SIMILARIDADE DE GRAFOS VIA HASHING

CARLOS HENRIQUE DE CARVALHO TEIXEIRA

SIMILARIDADE DE GRAFOS VIA HASHING

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Wagner Meira Júnior

Belo Horizonte

Maio de 2011

Teixeira, Carlos Henrique de Carvalho
T266s Similaridade de Grafos via Hashing / Carlos Henrique de Carvalho Teixeira. — Belo Horizonte, 2011
xxii, 82 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Wagner Meira Júnior

1. Computação – Teses. 2. Mineracão de Dados (computação) – Teses. 3. Teoria dos Grafos – Teses. I. Orientador. II. Título.

CDU 519.6*72(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Similaridade de grafos via hashing

CARLOS HENRIQUE DE CARVALHO TEIXEIRA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. WAGNER MEIRA JÚNIOR
Departamento de Ciência da Computação - UFMG

PROF. ALEXANDRE PLASTINO DE CARVALHO
Instituto de Computação - UFF

PROF. ADRIANO ALONSO VELOSO
Departamento de Ciência da Computação - UFMG

PROF. SEBASTIÁN ALBERTO URRUTIA
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 13 de maio de 2011.

Agradecimentos

Aos meus amados pais, Antônia e Geraldo, minha eterna gratidão por todo amor e incentivo durante esta jornada. Jornada que certamente seria mais difícil sem o apoio incondicional dos meus queridos irmãos Luíza, Camila e Cristiano. O meu sincero obrigado ao meu cunhado Fabrício pelos valiosos conselhos dados durante toda essa caminhada. Não posso me esquecer das especiais manhãs de domingo ao lado da minha família, tios e primos, que sempre me trouxeram paz e alegria. Gostaria de agradecer também um companheiro especial, Guto, nosso poodle, que nos acordava nessas mesmas manhãs de domingo. Meus agradecimentos a minha turma, Adriane, Daniel, Erika, Geovane, José Fernandes, Luana, Roberto e Túlio. Ao lado de vocês, a tristeza não tem lugar.

Em Belo Horizonte, pude conviver com pessoas formidáveis. Dentre essas pessoas, destaco a minha primeira amizade criada na capital mineira, Breno Vitorino, a quem eu serei enternamente grato por toda ajuda e humanidade. É impossível não me lembrar, também, dos meus amigos e colegas de trabalho do laboratório E-Speed, Arlei, Charles, Orair, Fernando, Pedro, Macambira, Coutinho, Douglas e George. Foram anos inesquecíveis de aprendizado e vocês terão minha eterna gratidão! Em especial, Arlei, que foi meu companheiro em diversos trabalhos, Orair, que me deu apoio e incentivo durante meus primeiros passos na área acadêmica, e Coutinho, que sempre esteve disponível para ajudar em qualquer circunstância.

Durante o meu mestrado, passei cerca de quatro meses nos Estados Unidos, na universidade do estado de Ohio (OSU). Lá, pude encontrar pessoas extremamente qualificadas e solidárias como o Prof. Srini e seus alunos, Shirish, Venu, Ye e Xintian. A eles, o meu obrigado por toda ajuda oferecida para que eu me adaptasse rapidamente aos EUA. Em especial, ao Prof. Srini, que foi meu orientador durante esse período de grandes conquistas.

Não seria possível concluir essa etapa da minha vida acadêmica sem a assistência social da Fundação Universitária Mendes Pimentel, que me axiliou com a moradia universitária e bolsa durante a minha graduação. Agradeço, também, aos profissionais

do Departamento de Ciência da Computação da UFMG. Especialmente a Sônia, por sua atenção e carinho com relação aos alunos de graduação.

Por fim, gostaria de agradecer o meu orientador, Prof. Wagner, que me indicou o caminho a ser trilhado nesse longo período de parceria e colaboração. Muito obrigado pelos conselhos e intervenções nas horas corretas, e por tonar possível essa dissertação.

"O mundo da realidade tem seus limites; o mundo da imaginação é ilimitado."
(Jean-Jacques Rousseau)

Resumo

Grafo é uma estrutura de dados universal capaz de representar objetos e conceitos. Nas últimas décadas, o interesse por essa estrutura tem sido impulsionado pela grande quantidade de dados disponível, modelados naturalmente como grafos. Exemplos são vistos desde a Web com repositórios XML e dados sobre redes sociais até Bioinformática e Química com dados estruturais relativos a proteínas e moléculas. Assim, tem-se uma necessidade de se manejar, consultar e analisar de forma eficiente esses dados, que estão aumentando em termos de volume continuamente.

O objetivo deste trabalho é comparar ou calcular a similaridade entre dois grafos quaisquer de forma eficiente e eficaz, facilitando consultas e análises de grandes bases de dados. Podemos dividir nosso método em duas etapas principais: (1) função de transformação e (2) função de assinatura. Na primeira, decompomos os grafos em subestruturas definidas nesse trabalho como caminhos aproximados. Diferente dos caminhos simples, os caminhos aproximados permitem saltos entre os vértices (*gaps*) sendo uma estrutura mais flexível capaz de descrever relacionamentos indiretos sobre os grafos. A similaridade entre dois grafos é, então, calculada em função do número de subestruturas compartilhadas através de um *kernel* de grafos. Visto que o conjunto de estruturas gerado para representar um grafo pode ser grande, a ideia da função de assinatura é reduzi-lo a um conteúdo fixo e pequeno através de técnicas de *hashing*. Além de tornar possível a análise de grandes bases de dados em memória principal, as assinaturas estimam a similaridade entre os conjuntos de forma eficiente, com qualidade assegurada.

Nós executamos o método proposto em diversas bases reais e sintéticas, avaliando-o em diferentes aplicações tais como recuperação de dados similares e classificação de grafos. Os resultados mostram que caminhos aproximados podem ser utilizados de forma eficiente, obtendo ganhos em relação às técnicas encontradas na literatura.

Abstract

A graph is a universal data structure, useful to represent several objects and concepts. In the recent decades, the interest in graphs has been driven by a large amount of data available. Examples include XML repositories, social networks, biological networks, and chemical graphs. Therefore, it is necessary to manage, query and analyze such large graph data efficiently.

The central problem of this thesis is the computation of the similarity between graphs in an efficient and effective manner. The proposed approach may be divided into two parts: (1) a transformation function, and (2) a signature function. A transformation function decomposes the input graph into approximate paths, which are substructures presented by this work. Approximate paths differ from simple paths by allowing gaps between nodes. Such flexible substructures are able to describe direct and indirect relationships in graphs. The similarity between two graphs is computed through a kernel function based on the number of substructures shared by them. Since the number of substructures that represent a graph may be large, a signature function applies a hashing technique in order to provide a short descriptor for a set of substructures. The signatures are short enough to fit into the main memory and may estimate the similarity between the sets efficiently, with theoretically guaranteed effectiveness.

We have evaluated the proposed method using several real and synthetic datasets, from different application scenarios, such as information retrieval and classification. The results show that approximate paths may be used efficiently and achieve gains w.r.t. the techniques from the literature.

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Sumário

Capítulo 1

Introdução

O rápido desenvolvimento das técnicas de armazenamento digital permitiu o surgimento de grandes bases de dados. Diariamente, dados oriundos de diversas fontes tais como empresas, governos, organizações e indivíduos são criados, armazenados e frequentemente disponibilizados para análise em múltiplos formatos. Grande parte desses são mantidos em bancos de dados em formatos (semi-)estruturados – grafos e árvores.

Alvo de estudos da Matemática há centenas de anos e, recentemente, da Ciência da Computação, grafo é uma estrutura de dados universal capaz de representar objetos e conceitos. Em uma representação baseada em grafo, vértices indicam objetos ou partes de um objeto, enquanto arestas descrevem relações entre eles. Grafos possuem diversas características interessantes: podemos transladar, rotacionar ou mesmo transformar um dado grafo G em sua imagem-espelho, que ainda assim teremos o mesmo grafo. Tais propriedades invariantes, juntamente com o fato de grafos serem adequados para modelar objetos e seus relacionamentos, fazem com que eles sejam utilizados em inúmeras aplicações e serviços. De fato, até mesmo filósofos argumentam que um grafo é a melhor forma para descrever, matematicamente, o mundo [Dipert, 1997].

Nas últimas duas décadas, o interesse por grafos na Ciência da Computação foi impulsionado por duas razões principais. Primeiro, o aumento do poder de processamento das máquinas, uma vez que as operações sobre grafos são complexas e custosas computacionalmente. Segundo, a grande quantidade de dados disponível, modelados naturalmente como grafos – bancos de dados biológicos [Stockham et al., 2002] e redes sociais [Tsvetovat et al., 2004] crescem vertiginosamente rápido. Outros exemplos são oriundos da Web com repositórios XML [Brazma et al., 2003], química com dados estruturais relativos às moléculas [Stockham et al., 2002] e até visão computacional e reconhecimento de padrões, onde utilizam-se modelos estruturais para representar objetos do mundo real [Samet, 1990; Sanfeliu et al., 2002]. Temos, então, uma neces-

sidade clara de organizar, consultar e analisar de forma eficiente tais dados, que estão em contínuo crescimento.

1.1 O problema

Uma operação fundamental para análise e organização de dados é o *cálculo de similaridade* ou *comparação* entre entidades. Sendo amplamente utilizada em áreas como inteligência artificial, mineração de dados e aprendizado de máquina, o estudo dessas operações será o foco desse trabalho. Em nosso caso, dados dois grafos quaisquer, utilizamos uma função de comparação para mensurar a similaridade (ou a diferença) estrutural[1] entre eles.

Os métodos de comparação de grafos têm sido aplicados em diversos domínios e são uma importante ferramenta para solucionar problemas como recuperação de grafos similares, classificação de grafos, descoberta de padrões e caracterização de dados estruturais. Para citarmos alguns cenários de aplicações possíveis, dividimos os problemas em quatro categorias: redes sociais, redes de informação, redes tecnológicas e redes biológicas [Newman, 2003].

- **Redes sociais :** Uma rede social é formada por um grupo de pessoas (representadas por vértices num grafo) que se relacionam ou interagem (representados por arestas num grafo). Tais relacionamentos podem ser, por exemplo, amizade – ou trabalho – entre indivíduos ou relacionamentos de negócio entre empresas e companhias [Rapoport & Horvath, 1961; Mizruchi, 1982]. Na última década, houve uma expansão bastante significativa desse domínio, originando diferentes tipos de redes sociais. Podemos citar como possível aplicação dessa área, o estudo e análise dos grafos de influência dos usuários [Leskovec et al., 2007]. Em outras palavras, cada usuário propaga informações na rede seguindo um padrão. Estudar e analisar tais grafos é uma tarefa importante para entendermos os diferentes padrões de propagação nas redes sociais e criarmos modelos analíticos capazes de descrever esse fenômeno.

- **Redes de informação :** Também conhecidas como redes de conhecimento, essa classe de grafos é representada pelas redes de citação. Duas redes clássicas dessa categoria são as redes de citações de artigos científicos e as redes constituídas por páginas da Web [Egghe & Rousseau, 1990; Huberman, 2001; White et al., 2004].

[1] Nesse trabalho, a função de similaridade é baseada na topologia dos grafos, isto é, nos relacionamentos entre os vértices.

O nome "redes de informação" vem do fato de que estrutura do grafo mantém informações guardadas nos seus nodos. Obviamente, existem aspectos sociais nas redes de citação também. Note ainda que os relacionamentos possuem uma orientação, e portanto tais redes possuem arestas direcionadas ou arcos. Grafos da Web, capturados pelas máquinas de busca, são essenciais para monitorar a evolução da rede e computar propriedades globais tal como *PageRank* das páginas. Esse monitoramento constante envolve medir a quantidade e o impacto das mudanças que ocorreram na Web em períodos subsequentes, o que pode ser conhecido a partir da similaridade de grafos [Papadimitriou et al., 2008].

- **Redes tecnológicas :** As redes tecnológicas são redes projetadas pelos humanos para transportar mercadorias ou distribuir recursos e informações. Constituem exemplos dessa classe as redes de energia elétrica, rodovias e a Internet (rede física usada para transportar as informações)[Kalapala et al., 2006; Faloutsos et al., 1999]. Outro exemplo comumente estudado são os fluxos de controle de programas. Os fluxos de controle formam grafos (*control flow graph* - CFG) que descrevem os caminhos que podem ser percorridos por um programa durante sua execução. Um problema dessa área é a detecção de erros (*bugs*) em códigos [Lo et al., 2009]. Assim, o conceito de similaridade de grafos pode ser aplicado para obter padrões comuns de erros ou mesmo identificar padrões anômalos que representem potenciais erros.

- **Redes biológicas e químicas:** A quarta categoria de redes é formada por grafos provindos das áreas de biologia e química. Na biologia, podemos citar as redes que mapeiam as interações físicas entre proteínas [Ito et al., 2001; Jeong et al., 2001]. Tais grafos, também chamados de redes de interação de proteínas, são fundamentais para se conhecer e entender os genomas. Note ainda que uma proteína, por si só, pode ser representada como um grafo. Redes químicas, por sua vez, representam compostos químicos ou moléculas. Os vértices desses grafos descrevem elementos químicos (hidrogênios, carbonos, etc) enquanto as arestas evidenciam as interações entre eles. Classificação e recuperação de proteínas e compostos químicos são problemas que envolvem comparação de estruturas e cálculo de similaridade [Murzin et al., 1995].

Logicamente, para atingir resultados satisfatórios, as funções de similaridade devem possuir dois requisitos principais: (1) eficácia e (2) eficiência. No primeiro, precisamos garantir que o valor de similaridade encontrado seja coerente, isto é, os casos de falso-positivos (grafos estruturalmente diferentes, onde a função retorna um alto

grau de similaridade) ou falso-negativos (grafos com alto grau de similaridade, mas que a função nos dá um baixo valor) devem ser mínimizados. Em segundo lugar, é necessário desenvolver técnicas eficientes capazes de processar grandes bases de dados. Todavia, por serem estruturas bastante flexíveis (podem possuir tamanhos variáveis), a comparação entre grafos pode se tornar proibitiva, devido a seu elevado tempo de processamento. Assim, a tarefa de análise de dados através do cálculo de similaridade em bases formadas por grafos constitui um desafio de pesquisa, sendo alvo de vários trabalhos recentes.

1.2 Um arcabouço para comparação de grafos

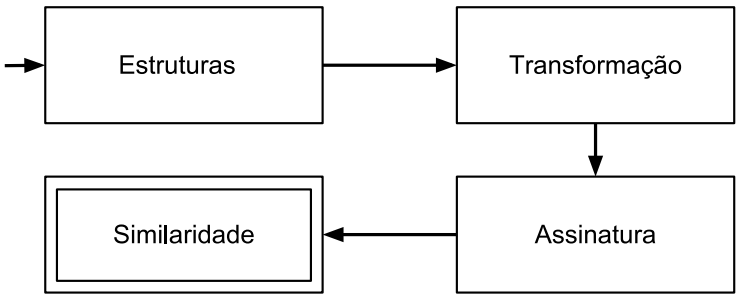


Figura 1.1: Diagrama de execução do arcabouço

Neste trabalho, investigamos e propomos uma nova técnica para enfrentar o problema de similaridade de dados (semi-)estruturados ou grafos considerando dois desafios principais: (1) eficácia e (2) eficiência. Nosso objetivo é comparar ou calcular a similaridade entre dois grafos quaisquer de forma eficiente e eficaz, facilitando consultas e análises de grandes bases de dados. Para tanto, construímos um arcabouço para comparação de grafos. Nosso arcabouço (figura 1.1) recebe os grafos como entrada e transformam esses grafos em assinaturas, facilitando, assim, sua comparação. As etapas do arcabouço estão descritas a seguir:

- Estruturas : Grafos são extraídos da base de dados. Como mencionamos, existem vários dados disponíveis que são naturalmente modelados como grafos. Nessa dissertação focamos em grafos rotulados, entretanto, podemos estender facilmente nosso método para qualquer outro tipo de representação baseada em grafo.

- Transformação : Nessa fase, empregamos as funções de mapeamento (ϕ) que transformam as estruturas em conjuntos de subestruturas (subgrafos, caminhos etc). Essa fase possibilita a definição das funções responsáveis pelo cálculo de similaridade entre dois grafos, sendo fundamental para o arcabouço. Normalmente, essa parte é também responsável pela maior parte do custo de processamento do arcabouço.

- Assinatura : Aqui, convertemos os conjuntos de subestruturas em assinaturas que podem ser alocadas em memória principal. As assinaturas podem ser vistas como uma sumarização dos grafos originais, o que possibilita a comparação eficiente entre os dados estruturados. Apesar de fornecer um valor aproximado para a similaridade, as assinaturas possuem garantias teóricas de qualidade.

- Similaridade : Nessa parte, calculamos os valores de nossa função de comparação. A comparação pode ser feita de duas formas. Primeiro, através do número de subestruturas compartilhadas pelos conjuntos e, alternativamente, através das assinaturas dos respectivos conjuntos. Note que neste, temos uma comparação eficiente em termos desempenho.

1.3 Contribuições dessa dissertação

Podemos sumarizar as principais contribuições dessa dissertação em:

1. Definimos uma nova subestrutura de grafos chamada de caminhos aproximados e, consequentemente, uma nova forma de decompor os grafos em subpartes. Diferentemente dos caminhos simples, os caminhos aproximados permitem a ocorrência de saltos na sequência de vértices, isto é, dois vértices consecutivos na sequência não são necessariamente vizinhos diretos entre si. De fato, um caminho aproximado é uma estrutura flexível, capaz de capturar relacionamentos indiretos entre as entidades de um grafo.

2. Funções para comparação entre dois grafos, também conhecidas como métodos *kernel*, são discutidas nesse trabalho. Foram propostas duas funções *kernel* para comparação de estruturas através da decomposição de caminhos aproximados. Ou seja, transformamos os grafos em um conjunto de subestruturas (caminhos aproximados) e a similaridade entre dois grafos é dada de acordo com o número de subestruturas comuns entre eles. Mostramos que as funções *kernel* são equivalentes ao bem conhecido método de *Jaccard* para comparação de conjuntos.

3. Para enfrentar esse problema de eficiência, exploramos técnicas de *hashing*. Dado um grafo qualquer, nossa proposta é utilizar uma técnica de *hashing*, conhecida como *Min-hash*, sobre seu (multi-)conjunto de subestruturas, gerando um conteúdo ou uma assinatura de tamanho fixo e pequeno. A comparação entre os grafos é feita, então, através das assinaturas com qualidade teoricamente assegurada. Essa estratégia consegue reduzir significativamente o consumo de memória principal utilizada, o que torna possível a análise de grandes bases de dados.

4. Avaliamos a eficiência e a eficácia de nossa abordagem em diferentes domínios de aplicação, incluindo consultas e recuperação de grafos similares, além de classificação de grafos em bancos de dados químicos. Na primeira aplicação, mostramos que, em geral, nossa estratégia retorna o grafo alvo nas primeiras posições do *ranking* de similaridade. Quanto à classificação de grafos, o método *kernel* proposto alcançou resultados superiores às técnicas encontradas na literatura (ganhos de 1% a 7%).

É importante ressaltar que nossa abordagem não impõe restrições quanto ao tipo de representação de grafo usada, permitindo obter assinaturas e, consequentemente, o cálculo de similaridade de grafos (não-)direcionados, (não-)rotulados, (não-)ponderados, dentre outras alternativas.

1.4 Organização dessa dissertação

Essa dissertação está dividida em mais cinco capítulos. O capítulo 2 apresenta conceitos e definições da Teoria de Grafos, além de algoritmos e estruturas de dados empregados na análise de dados estruturados. Técnicas tradicionais para comparação de grafos também são discutidas nessa parte do trabalho. O capítulo 3 esclarece questões sobre os métodos *kernel* tradicionais e métodos *kernel* de convolução que são aplicados a grafos em geral. Ainda nesse capítulo, propomos uma nova função *kernel* baseada em caminhos aproximados e discutimos outras técnicas para o cálculo de similaridade entre grafos que exploram outros tipos de sequência de vértices nos grafos (por exemplo, caminhos aleatórios, simples e mínimos). A seguir, o capítulo 4 apresenta as abordagens de *hashing* aplicadas para sumarizar os grafos gerando assinaturas. Isso possibilita realizar uma comparação eficiente dos grafos, com qualidade teoricamente assegurada. Os resultados empíricos dessa dissertação são relatados no capítulo 5, seguido da nossa conclusão (capítulo 6).

Capítulo 2

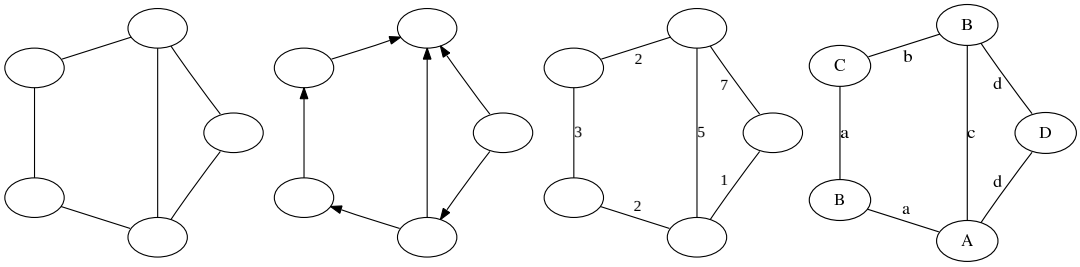
Conceitos preliminares e definições

Aqui, descrevemos conceitos e definições necessários para o entendimento desta dissertação. Primeiro, discutimos aspectos teóricos das estruturas baseadas em grafos. Dentre os temas mencionados estão os tipos de grafos existentes, subestruturas dos grafos (subgrafos, caminhos, etc.) e, até o clássico problema de isomorfismo. Algoritmos e estruturas de dados básicos para pesquisa e análise de grafos também são discutidos nessa seção. Já a segunda parte desse capítulo relata as abordagens tradicionais propostas para comparação de grafos: técnicas que, usualmente, exploram conceitos provindos da Teoria de Grafos.

2.1 Princípios básicos da Teoria de Grafos

Nesta seção apresentamos conceitos e definições da teoria de grafos, que serão utilizados ao longo desta dissertação. A tabela 2.1 descreve algumas notações.

2.1.1 Tipos de grafos



(a) Grafo simples (b) Grafo direcionado (c) Grafo ponderado (d) Grafo rotulado

Figura 2.1: Modelos de grafos

$V(G)$	conjunto de vértices de um grafo G
$E(G)$	conjunto de arestas de um grafo G
$L(G)$	conjunto de rótulos de um grafo G
$\|V(G)\|$	número de vértices de um grafo G
$\|E(G)\|$	número de arestas de um grafo G
$\|L(G)\|$	número de rótulos de um grafo G
$N(v)$	conjuntos de vértices vizinhos do vértice v em G
$\|N(v)\|$	número de vértices vizinhos (ou grau) do vértice v em G

Tabela 2.1: Notações simples

Existem várias formas de modelarmos um problema utilizando grafos. Cada forma ou tipo possui um conjunto de características e ferramentas necessário para representar adequadamente um certo problema. Os exemplos básicos mais conhecidos são apresentados na figura 2.1 : grafos simples, direcionados, ponderados e rotulados.

Definição 1 ***(Grafo simples)*** *Um grafo G é constituído por dois conjuntos, $G = (V, E)$, onde V e E representam os vértices e arestas de G, respectivamente. Cada aresta $e \in E$ conecta um par de vértices (v, u) de V, ou seja, $E \subseteq V \times V$. Dado um par de vértices v e u, apenas uma aresta (v, u) pode existir e não existe aresta de um vértice para ele mesmo, isto é, $(v, v) \notin E$.*

Definição 2 ***(Grafo direcionado)*** *Um grafo direcionado G_D é um grafo simples, onde os pares de arestas $e = (v, u)$ e $t = (u, v)$ são considerados distintos. Em outras palavras, o par (v, u) indica que aresta está direcionada de v para u, enquanto, o par (u, v) indica o oposto.*

A definição 1 descreve o modelo de grafos ilustrado na figura 2.1a. Por sua simplicidade, esse modelo é amplamente utilizado. De fato, os demais tipos foram propostos a partir de extensões sobre o modelo de grafo simples. A figura 2.1b, por sua vez, mostra um exemplo de grafo direcionado (definição 2). Esse tipo de modelo é empregado em problemas onde as relações entre as entidades envolvidas possuem uma orientação, um sentido.

Definição 3 ***(Grafo ponderado)*** *Um grafo ponderado G_P é um grafo simples, cujas arestas possuem pesos ou valores reais. Os valores são definidos por uma função de mapeamento $P : E \to \mathbb{R}$, sendo $\mathbb{R}$ o conjunto dos números reais.*

Definição 4 ***(Grafo rotulado)*** *Um grafo rotulado G_R é um grafo simples, onde os vértices e as arestas possuem tipos ou rótulos. Os rótulos são definidos por uma função de mapeamento $R : V \cup E \to L$, sendo L o conjunto de rótulos.*

Grafos ponderados (definição 3) também são bastante comuns. Uma característica importante dessa classe de grafos é a capacidade de mensurar a intensidade das relações entre os vértices do grafo, como pode ser visto na figura 2.1c. De uma forma similar, os grafos rotulados (definição 4) atribuem rótulos para os conjuntos de vértices e arestas (ou ambos). A figura 2.1d representa uma instância desse caso, na qual vértices e arestas são tipados. É importante ressaltar que outros tipos de modelos de grafos podem ser criados, combinando as características das definições anteriormente apresentadas [Gross & Yellen, 1999].

2.1.2 Subgrafos, caminhos e ciclos

Definição 5 ***(Subgrafo)*** *Um subgrafo G' de G tem seus vértices e arestas formados por subconjuntos de vértices e arestas de G. Em outras palavras, $V(G') \subseteq V(G)$ e $E(G') \subseteq E(G)$, onde as arestas de $E(G')$ devem possuir as duas extremidades em $V(G')$.*

Um grafo G pode ser visto como a concatenação de várias estruturas ou grafos menores. Tais estruturas são conhecidas por subgrafos (definição 5). Baseado nessa definição, até mesmo um vértice do grafo, apenas, é considerado um subgrafo. Os subgrafos podem ser categorizados de acordo com suas formas e características particulares. A seguir, destacamos algumas classes de subgrafos: subgrafos induzidos, caminhos e ciclos.

Definição 6 ***(Subgrafo induzido)*** *Um subgrafo G'_I induzido por $I \subset V(G)$ tem seu conjunto de vértices $V(G') = I$. Ademais, $E(G'_I)$ é formado por todas as arestas em $E(G)$ que tenham as duas extremidades em I.*

Os subgrafos induzidos, descritos pela definição 6, são criados ou induzidos a partir de um conjunto de vértices – obviamente, podemos induzir subgrafos também por arestas. Note que, diferentemente de um subgrafo comum, um subgrafo induzido pelos vértices I a partir de G deve, *necessariamente*, conter todas as arestas de G que incidem em dois vértices do conjunto I.

Definição 7 ***(Caminho)*** *Um caminho P de tamanho l em G é um subgrafo formado por uma sequência de vértices e arestas alternados, $(v_1, e_1, v_2, e_2, v_3, ..., e_{l-1}, v_l)$, tal que cada aresta é incidente aos vértices anterior e posterior no caminho, $e_i = (v_i, v_{i+1})$.*

Definição 8 ***(Ciclo)*** *Um ciclo C de G é um caminho, $(v_1, e_1, v_2, e_2, v_3, ..., e_{l-1}, v_l)$, onde $v_1 = v_l$. Além disso, os demais vértices e arestas de $V(C)$ e $E(C)$ devem ser distintos.*

A definição 7 descreve, formalmente, o que é um caminho em um grafo. Essa definição permite a repetição de vértices e arestas, entretanto, um caminho onde todos os vértices e arestas são diferentes é chamado de *caminho simples*. Um ciclo (definição 8), por sua vez, possui uma estrutura similar a um caminho simples, porém, com o primeiro e o ultimo vértice da cadeia iguais. Existem, ainda, outras estruturas bastante conhecidas como o *caminho hamiltoniano* e o *caminho euleriano* que são um caminho simples que passa por todos os vértices e por todas as arestas de um grafo, respectivamente.

2.1.3 Isomorfismo de grafos e subgrafos

Definição 9 ***(Isomorfismo)*** *Dois grafos G e H são isomorfos se existe uma função bijetora f de $V(G)$ em $V(H)$ tal que a aresta $(v_i,v_j) \in E(G)$ se e somente se $(f(v_i), f(v_j)) \in E(H)$.*

Isomorfismo de grafos é um problema intensivamente estudado na literatura. Informalmente, podemos dizer que dois grafos G e H são isomorfos se eles podem ser desenhados pelo mesmo diagrama, ou seja, se existe um casamento (*matching*) perfeito entre os vértices e arestas dos dois grafos. Um conjunto de grafos isomorfos entre si é chamado de classe de isomorfismo de grafos. Dessa forma, podemos dizer que o problema de isomorfismo particiona um conjunto de grafos quaisquer em classes de equivalência. Na definição 9, os grafos são entendidos como grafos simples, isto é, não direcionados e não rotulados. No entanto, o conceito de isomorfismo pode ser aplicado a todos os outros modelos de grafos.

Definição 10 ***(Isomorfismo de Subgrafo)*** *Seja $S = \{G_1, G_2, ..., G_k\}$ o conjunto de todos os subgrafos de G. Dizemos que H é isomorfo a um subgrafo de G se e somente se H for isomorfo a algum subgrafo G_i, onde $G_i \in S$.*

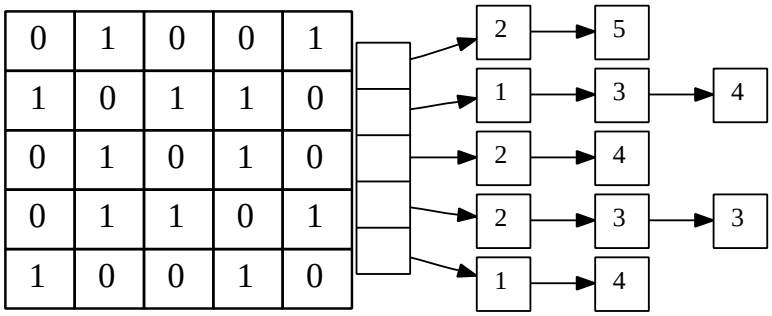
Além do problema de isomorfismo entre grafos, existe também o problema de isomorfismo de subgrafos (definição 10). Este último busca decidir se um dado grafo H é isomorfo a um dos subgrafos de G. Tal problema é conhecidamente NP-difícil, diferentemente do problema de isomorfismo de grafos [Garey & Johnson, 1990; Gross & Yellen, 1999].

2.1.4 Estrutura de dados e algoritmos

A seguir são apresentadas duas estruturas de dados básicas para representação de grafos em geral: (1) matriz de adjacência e (2) listas de adjacência. O emprego dessas

estruturas interfere diretamente no tempo de processamento dos métodos de comparação entre grafos. Posteriormente, algoritmos para pesquisa de vértices e computação de caminhos mínimos no grafo são descritos e analisados. Esses algoritmos são extremamente úteis e são utilizados por diversos métodos para cálculo de similaridade entre estruturas.

2.1.4.1 Estruturas



(a) Matriz de adjacência (b) Lista de adjacência

Figura 2.2: Estrutura de dados para grafos

Matriz de adjacência e listas de adjacência são estruturas de dados básicas usadas para representar grafos. A figura 2.2 ilustra como seriam as representações para um mesmo grafo utilizando tais estruturas.

Como podemos perceber, a matriz de adjacência (figura 2.2a) é uma estrutura bastante simples. A estrutura consiste numa matriz $N \times N$, onde N é um número de vértices do grafo. O valor 1 na posição (i, j) da matriz nos diz que os vértices i e j estão ligados. Note que, alternativamente, podemos usar as posições da matriz para alocarmos os pesos ou rótulos das arestas. São algumas das vantagens dessa estrutura de dados: verificar se dois vértices são adjacentes, adicionar ou remover ligações – complexidade $O(1)$. Todavia, temos um desperdício de memória, especialmente em grafos esparsos, pois, a matriz mantém informações sobre as arestas existentes e não existentes do grafo (valores 1 e 0, respectivamente). Além disso, para listar as arestas incidentes a um dado vértice devemos percorrer todos os vértices – complexidade $O(N)$.

Por outro lado, as listas de adjacência limitam-se a guardar somente as informações das arestas existentes do grafo, como mostrado na figura 2.2b. Duas vantagens importantes da representação baseada em listas são: (1) verificar todas as arestas incidentes num dado vértice pode ser feito em tempo linear em relação ao número de arestas[1] e (2) baixo desperdício de memória. Apesar disso, representar um grafo denso

[1]Essa vantagem é importante especialmente quando trabalhamos com grafos esparsos.

com o uso de listas de adjacência é inviável, pois, pode consumir uma quantidade maior de memória principal comparada ao uso de matriz. Ademais, o uso dessa representação é limitado em cenários onde as estruturas dos grafos não são alteradas frequentemente – remover arestas do grafo possui custo linear.

2.1.4.2 Algoritmos

Descrevemos, a seguir, três algoritmos fundamentais para análise de grafos. São eles : (1) busca em largura , (2) busca em profundidade e (3) algoritmo de Dijkstra.

Algoritmo 1: Busca em largura

```
Entrada: Um grafo G e o vértice v a ser examinado
Saída: Os vértices alcançados a partir de v
início
    para cada u ∈ V(G) − v faça
        verif[u] = −1;
    verif[v] = 0;
    S = S ∪ v;
1   enquanto S ≠ ∅ faça
        Selecione u ∈ S de acordo com a ordem de inserção;
2       S = S − u para cada n ∈ N(u) faça
            se verif[n] < 0 então
                verif[n] = verif[u] + 1;
3               S = S ∪ n
```

O algoritmo 3 descreve o método de busca em largura (*breadth-first search*), utilizado para realizar buscas num grafo. Nesse método, a busca é iniciada a partir de um vértice raiz e expandida, incrementalmente, para a vizinhança. Por exemplo, dado um vértice v como raiz, o algoritmo explora primeiro os vértices mais próximos de v, $N(v)$, e, posteriormente, os vizinhos de $N(v)$ (caso eles não tenham sido examinados). Esse processo é repetido até que o vértice alvo seja atingido ou todos os vértices do grafo sejam examinados. O algoritmo apresentado visita cada vértice e cada aresta no máximo uma vez – considerando a representação baseada em listas de adjacência. Assim, o algoritmo executa a busca num tempo de $O(V + E)$, além da complexidade de espaço de $O(V)$.

Um método recursivo para busca de profundidade (*depth-first search*) é descrito no algoritmo 2. Tal como o algoritmo de busca em largura, o método de busca em profundidade é usado para pesquisar vértices num grafo. Esse último, porém, emprega uma estratégia diferente, isto é, vértices mais distantes são expandidos antes dos vér-

Algoritmo 2: Busca em profundidade

```
Entrada: Um grafo G e o vértice v a ser examinado
Saída: Os vértices alcançados a partir de v
Verifica (vértice u)
1 início
    para cada n ∈ N(u) faça
        se verif[n] < 0 então
            verif[n] = verif[u] + 1;
            Verifica (n);
Busca ()
2 início
    para cada u ∈ V(G) − v faça
        verif[u] = −1;
    verif[v] = 0;
    Verifica (v);
```

tices mais próximos. O algoritmo inicia-se examinando um vértice vizinho u de v, onde v é o vértice raiz. A seguir, um vértice vizinho de u, ainda não explorado, é selecionado. Caso o vértice corrente não possua vizinhos ou tenha toda sua vizinhança analisada, voltamos para o último vértice examinado com vizinhos inexplorados (procedimento conhecido como *backtracking*), senão continuamos a busca escolhendo um de seus vizinhos. Como no caso anterior, o algoritmo de busca em profundidade tem complexidades de espaço e tempo de $O(V)$ e $O(V + E)$, respectivamente.

Algoritmo 3: Algoritmo de Dijkstra

```
Entrada: Um grafo G e o vértice v a ser examinado
Saída: Distancia mínima do vértice v ao demais vértices de G
início
    para cada u ∈ V(G) − v faça
        dist[u] = ∞;
    dist[v] = 0;
    S = S ∪ v;
1   enquanto S ≠ ∅ faça
        Selecione u ∈ S com o menor dist;
2       S = S − u para cada n ∈ N(u) faça
            se dist[n] > dist[u] + (u, n) então
                dist[n] = dist[u] + (u, n);
3               S = S ∪ n
```

O algoritmo de Dijkstra é um dos métodos mais famosos para calcular os caminhos de custo mínimo entre vértices de um grafo. Escolhido um vértice como raiz da busca, este algoritmo calcula o custo mínimo (ou distância mínima) deste para todos os demais vértices do grafo. Inicialmente, todos os nodos tem um custo infinito, exceto v (a raiz da busca) que tem valor 0. A seguir, os vizinhos de v são examinados e, caso suas distâncias para v sejam atualizadas (ou seja, distâncias menores para v foram encontradas), eles são colocados em uma fila. O próximo vértice a ser expandido é aquele cuja distância para v é a menor possível, dentre os vértices da fila. Uma vez expandido, o vértice é removido da fila. Esse processo continua enquanto existir vértice na fila. Note que o algoritmo de Dijkstra é similar à estratégia de busca em largura, porém, ele emprega uma estratégia para selecionar o próximo vértice a ser expandido[2]. Dijkstra pode ser implementado de forma eficiente usando uma estrutura *heap*, sendo executado em $O(|E|log|V|)$.

2.2 Estratégias para comparação de grafos

Definição 11 ***(Problema da comparação de grafos)*** *Seja Γ um espaço qualquer de representação dos grafos. Dados dois grafos quaisquer, $G \in \Gamma$ e $H \in \Gamma$, o problema da comparação de grafos consiste em determinar uma função de similaridade $S : \Gamma \times \Gamma \rightarrow \mathbb{R}$. Em outras palavras, $S(G, H)$ mensura a similaridade ou diferença entre G e H.*

Nessa dissertação, estamos interessados no problema de similaridade ou comparação de grafos (definição 11). É importante ressaltar que esse problema, apesar de relacionado[3], é diferente do problema de casamento (*matching*) de grafos [Bunke, 2000]. O problema de casamento de grafos emprega conceitos de isomorfismo e, dessa forma, busca identificar regiões topologicamente idênticas entre os grafos. Todavia, muitas aplicações do mundo real exigem abordagens tolerantes a erros, isto é, métodos que calculem um valor de similaridade entre entidades. A seguir, serão discutidos algoritmos tradicionais para enfrentar o problema de comparação entre grafos.

2.2.1 Técnicas baseadas em isomorfismo

Uma forma intuitiva de calcular a similaridade entre dois grafos é verificá-los com relação às suas identidades topológicas, i.e., por isomorfismo. Por exemplo, podemos

[2] O algoritmo de busca em largura utiliza a estratégia *first in-first out*.

[3] Alguns métodos para comparação de grafos exploram conceitos de isomorfismo.

definir uma métrica binária de similaridade com base no problema de isomorfismo: 1, caso os grafos sejam isomorfos; 0 caso contrário. Obviamente, essa função é bastante limitada, pois sua escala de similaridade apresenta apenas dois valores, o que não quantifica de forma satisfatória a diferença entre os grafos. Assim, foram propostos métodos baseados em subgrafos isomorfos comuns.

Definição 12 ***(Máximo Subgrafo Comum, MaSC)*** *Seja G' um subgrafo comum aos grafos G e H, então G' é isomorfo por subgrafo a G e H. G' é considerado o máximo subgrafo comum se não existe outro grafo que seja isomorfo por subgrafo à G e H com um número maior de vértices.*

A definição 12 descreve as características necessárias para que um subgrafo comum a dois grafos, G e H, seja considerado máximo [Neuhaus & Bunke, 2007]. Note que podem existir mais de um MaSC para um mesmo par de grafos. Para isso, basta que eles tenham o mesmo tamanho em relação ao número de vértices.

Definição 13 ***(Mínimo Supergrafo Comum, MiSC)*** *G' é considerado supergrafo comum dos grafos G e H, se G e H são isomorfos por subgrafo a G'. Se G' é mínimo supergrafo comum, então, não existe outro supergrafo comum a G e H com um número menor de vértices.*

O mínimo supergrafo comum (definição 13) é, logicamente, o oposto do problema do MaSC. Intuitivamente, um MiSC de dois grafos quaisquer, G e H, é o grafo com o menor número de vértices possível que tenha G e H como subgrafos. Como o MaSC, o MiSC também pode possuir mais de uma solução.

$$sim(G,H) = \frac{|MaSC(G,H)|}{max(|G||H|)} \tag{2.1a}$$

$$dist(G,H) = |MiSC(G,H)| - |MaSC(G,H)| \tag{2.1b}$$

$$sim(G,H) = 1 - \frac{dist(G,H)}{|MiSC(G,H)|} \tag{2.1c}$$

Assim, utilizando as definições de MiSC e MaSC foram propostas alternativas para o cálculo de similaridade entre grafos (equações 2.1) [Bunke & Shearer, 1998]. A equação 2.1a emprega apenas a definição MaSC e retorna um valor de similaridade entre [0,1]. Já a segunda obtém um valor de distância, maior do que zero, com base nos

dois conceitos apresentados. Note que a equação 2.1b pode ser facilmente modificada para retornar um valor de similaridade, como mostrado na equação 2.1c.

Definição 14 ***(Invariante de Grafo)*** *Sejam G e H dois grafos isomorfos e σ uma função que mapeia um grafo para um espaço d-dimensional, $\sigma : \Gamma \rightarrow \mathbb{R}^d$. Se $\sigma(G) = \sigma(H)$, então, σ é uma invariante de grafo.*

Infelizmente, não é conhecido nenhum algoritmo com tempo de execução polinomial para solucionar problemas relacionados ao isomorfismo de grafos [Garey & Johnson, 1990]. Entretanto, métodos eficientes para casamento de estruturas exploram o conceito de invariantes de grafos. De acordo com a definição 14, o tamanho dos grafos (número de vértices) é um invariante, pois, se dois grafos são isomorfos então eles devem possuir o mesmo tamanho. Assim, o objetivo é utilizar as invariantes antes de testar o isomorfismo de grafos para obter uma computação eficiente.

2.2.2 Técnicas baseadas em distância de edição

Distância de edição é uma medida de similaridade bastante usada em cadeia de caracteres (*strings*). Dadas duas cadeias de caracteres, A e B, a ideia central dessas técnicas é calcular o número mínimo de alterações necessárias para transformar A em B, ou vice-versa. Esse conceito pode ser aplicado a grafos de maneira similar e representa a abordagem mais poderosa dentro dos métodos tolerantes a erros para casamento de grafos [Bunke & Allermann, 1983]. Distância de edição em grafos envolve operações básicas tais como remoção, adição, ou substituição (alteração de rótulo) de vértices e arestas.

Definição 15 ***(Distância de edição em grafos)*** *A distância de edição entre dois grafos quaisquer, $d(G, H)$, é obtida através de uma sequência de operações S, cujo custo seja mínimo. Formalmente, $d(G, H) = min(custo(S))$, para todo S, onde S é uma sequência de operações que transformam G em H, ou vice-versa.*

Apesar de ser bastante intuitivo e poderoso, o cálculo exato da distância de edição (definição 15) é custoso computacionalmente, pois, trata-se de um problema NP-difícil [Garey & Johnson, 1990]. Além disso, cada operação (remoção, inserção, substituição etc) sobre vértices e arestas podem possuir pesos diferentes que variam de acordo com o domínio de aplicação. Por essa razão, a parametrização (atribuir pesos às operações distintas) das técnicas baseadas em distância de edição é difícil, o que torna o problema ainda mais complexo.

2.2.3 Técnicas baseadas em descritores topológicos

A terceira classe de técnicas usadas para comparação de grafos é bastante estudada na área de química [Todeschini et al., 2000]. Tais abordagens buscam representar os grafos através de vetores que sumarizam as características topológicas dos mesmos. O desafio aqui é encontrar um conjunto de descritores que descrevam de forma adequada a topologia dos grafos e apresentem resultados satisfatórios quanto aos valores de similaridade. Métodos populares para encontrar representações vetoriais de grafos são baseados na teoria espectral [Chung, 1997]. Outra forma encontrada na literatura é utilizar invariantes (definição 14) de grafos como descritores.

A vantagem dessa classe de algoritmos é que o uso de vetores permite o emprego de diversos métodos bem conhecidos e eficientes de indexação e busca no espaço Euclideano. Contudo, o tempo gasto para o cálculo de alguns descritores topológicos apresenta uma ordem de complexidade exponencial [Köbler & Verbitsky, 2008]. Além disso, o conjunto de descritores que melhor representa os grafos é dependente do domínio de aplicação, mesmo problema enfrentado pelas técnicas baseadas em distância de edição.

2.3 Discussão

Como foi visto, vários trabalhos e métodos foram propostos para comparação de grafos. Alternativas interessantes ainda surgem nas áreas de aprendizado de máquina e mineração de dados. Na primeira área, foram desenvolvidos métodos para determinar automaticamente os parâmetros para o cálculo da distância de edição [Neuhaus & Bunke, 2005, 2007]. Já na segunda, algoritmos eficientes para mineração de padrões frequentes são usados para determinar o valor de similaridade entre grafos (são considerados grafos similares aqueles que compartilham muitos padrões) [Kramer et al., 2001; Yan et al., 2004, 2006]. Apesar dessas propostas se mostrarem promissoras em diversos domínios e aplicações, elas ainda apresentam certas limitações: os métodos são ineficientes para grandes grafos ou eles utilizam representações simples dos grafos (ignorando a topologia).

Recentemente, surgiram estudos sobre a aplicação de métodos de *kernel* em dados estruturados. Técnicas baseadas em *kernel* apresentam características importantes para a análise de dados estruturados. Inicialmente, os grafos são divididos em várias subestruturas simples e essas subestruturas são, então, mapeadas num espaço *Hilbert* (usualmente Euclideano), representando cada grafo através de um vetor. A medida de similaridade é calculada sobre os vetores, através da quantidade de subestruturas

comuns (isomorfismo sobre as subestruturas simples). *Kernel* de grafos é uma alternativa recente e promissora, pois é uma abordagem tolerante a erros e apresenta uma complexidade polinomial, diferente dos outros trabalhos. Tais métodos serão também estudados nesse trabalho.

Capítulo 3

Kernel de grafos

A ideia por trás dos métodos *kernel* foi proposta há cerca de 40 anos [Kimeldorf & Wahba, 1971]. Basicamente, esses métodos transformam o espaço de representação dos dados através de uma função mapeamento ϕ e permitem a análise dos elementos de entrada nesse novo domínio (usualmente, um espaço Euclidiano). A comparação de dados passa a ser, então, realizada empregando operações simples como, por exemplo, o produto escalar. Os métodos *kernel* se consolidaram como um importante tópico na área de aprendizado de máquina devido à duas razões principais: (1) pelos trabalhos sólidos sobre *Support Vector Machines* [Boser et al., 1992; Cortes & Vapnik, 1995] e (2) por se mostrarem eficientes em diversos cenários e aplicações [Cristianini & Shawe-Taylor, 2000; Scholkopf & Smola, 2001].

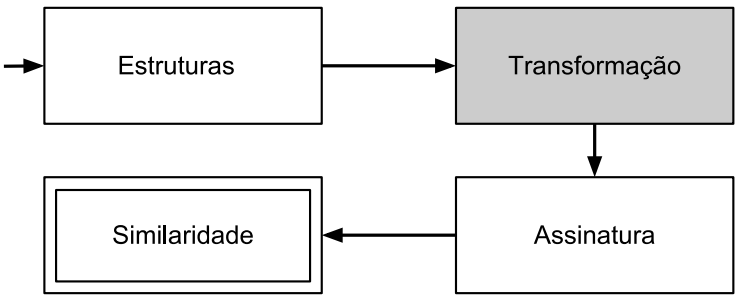


Figura 3.1: Fase de construção do *kernel* de grafos

Diferente dos métodos *kernel* tradicionais (que trabalham no espaço euclidiano), o desafio aqui é propor funções de transformação ϕ e *kernels* aplicados a grafos, ou seja, funções *kernel* aplicáveis a dados (semi-)estruturados, com tamanhos variáveis. Dessa

forma, apresentamos nesse capítulo um novo método *kernel* baseado em caminhos aproximados para grafos em geral. O *kernel* proposto explora a flexibilidade dessas subestruturas para capturar interações diretas e indiretas entre os vértices. Como será discutido, nossa função *kernel* é equivalente à similaridade de *Jaccard* entre conjuntos simples e também multiconjuntos.

A figura 3.1 destaca a fase de *transformação* no diagrama de execução do nosso método. Tal fase, descrita nesse capítulo, transforma os grafos de entrada num conjunto de subestruturas (caminhos aproximados) que serão usadas no cálculo de similaridade (função *kernel*).

Antes de detalharmos nossa abordagem, discutimos a importância dos métodos de *kernel* para o problema da classificação de dados e seus algoritmos, em particular, Support Vector Machines (SVMs). Posteriormente, introduzimos o conceito de *kernel* de convolução, que define uma classe de *kernels* que pode ser aplicada a dados (semi-)estruturados. Os trabalhos relacionados e outros *kernels* de grafos são discutidos no final desta seção.

É importante ressaltar que os conceitos apresentados na seção 3.1 – além de outros estudos avançados sobre tema de *kernel* e classificação de dados – podem ser encontrados em diversos livros e artigos científicos da área de aprendizado de máquina [Schölkopf et al., 1999; Scholkopf & Smola, 2001].

3.1 Conceitos preliminares

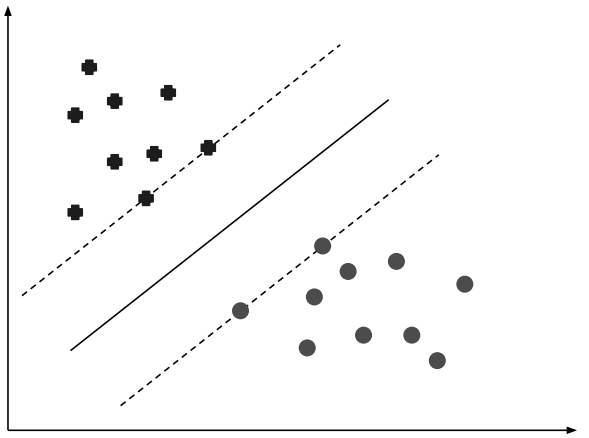


Figura 3.2: Classificação binária utilizando SVM com maximização da margem de separação

Usualmente, os métodos de *kernel* são utilizados para solucionar problemas de classificação de dados. Em nosso cenário, tal problema consiste em determinar a(s) categoria(s) de uma nova estrutura baseado em conhecimentos pré-estabelecidos (por exemplo, grafos previamente classificados). Nesta seção, exploramos o problema de classificação de grafos para introduzir os conceitos e definições dos *kernels*.

3.1.1 Kernel e Suport Vector Machines (SVMs)

SVMs constituem uma classe de algoritmos muito importante na área de aprendizado de máquina. SVMs tradicionais lidam com o problema da classificação binária[1], isto é, existem um conjunto de elementos e duas categorias, às quais esses estão assinalados. Uma SVM tem a tarefa de criar um modelo de classificação utilizando os exemplos previamente conhecidos e, baseando-se nesse modelo, inferir as classes de outros elementos. Formalmente, sejam $X = \{x_1, x_2, ...x_n\}$ o conjunto de elementos num espaço euclideano d-dimensional ($x_i \in X \subseteq \mathbb{R}^d$) e $Y = \{+1, -1\}$ o conjunto formado por duas categorias. Dados os pares $\{(x_1, y_1), (x_2, y_2), ...(x_n, y_n)\}$, formados pelos elementos e suas respectivas classes, nosso problema é definir uma função de mapeamento $f : X \to Y$.

A figura 3.2 mostra um exemplo de classificação binária utilizando uma SVM linear com maximização da margem de separação. Note que um hiperplano é introduzido no espaço de maneira que os elementos pertencentes à classe +1 fiquem separados dos elementos da classe -1.

$$f(x) =< w, x > +b = 0 \tag{3.1}$$

Um hiperplano pode ser descrito de acordo com a equação 3.1, onde $w \in X$ representa o vetor normal ao hiperplano e $b \in \mathbb{R}$. Como já mencionado, ele separa os elementos de acordo com sua classe, ou seja, divide o espaço em duas regiões ($f(x) < 0$ e $f(x) > 0$). Portanto, podemos definir um classificador simples, $g(x)$, de acordo com a equação 3.2, que captura o sinal dos valores retornados por $f(x)$.

$$g(x) = sinal(f(x)) = sinal(< w, x > +b) = \begin{cases} +1 & \text{se } f(x) > 0, \\ -1 & \text{se } f(x) < 0. \end{cases} \tag{3.2}$$

Note que existem vários hiperplanos possíveis que satisfazem a separação entre as classes, em particular, definidos por $y_i(< w, x_i > +b) > 0, \forall i \in \{1, 2, ...n\}$. Dentre

[1]Existem SVMs voltados para classificação multi-classe.

esses, escolhemos aquele que maximiza a distancia mínima[2] entre o hiperplano e os pontos mais próximos das duas classes. Tal plano pode ser ser obtido pela minimização de $||w||$, expresso pelo seguinte problema de otimização:

$$\begin{aligned} &\underset{w,b}{\text{minimizar}} && \frac{1}{2}||w||^2 \\ &\text{sujeito a} && y_i(< w, x_i > +b) \geq 1,\ \forall i \in \{1, 2, ...n\} \end{aligned} \tag{3.3}$$

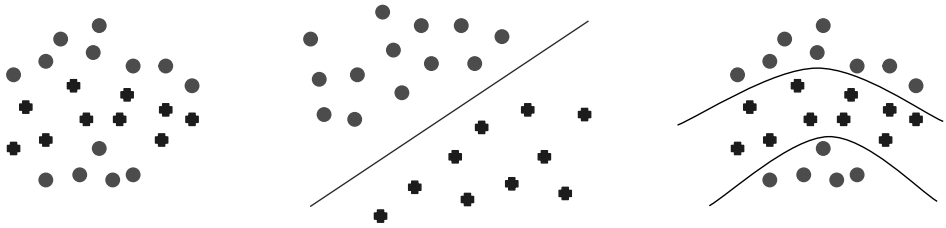
As restrições impostas no problema de otimização descrito pela equação 3.3 garantem que não haja dados mapeados entre as margens de separação das classes. Por esse motivo, a SVM obtida é também chamada de SVM com margens rígidas. Para solucionarmos esse problema de otimização, existem técnicas tradicionais que utilizam funções Lagrangianas e resolvem o problema dual (o problema original é referenciado como forma primal) :

$$\begin{aligned} &\underset{\alpha}{\text{maximizar}} && -\frac{1}{2}\alpha^T H\alpha + \sum_{i=1}^{n} \alpha_i \\ &\text{sujeito a} && \sum_{i=1}^{n} \alpha_i y_i = 0 \text{ e } \alpha_i \geq 0,\ \forall i \in \{1, 2, ...n\} \end{aligned} \tag{3.4}$$

onde a matriz $H \in \mathbb{R}^{n \times n}$, $H_{ij} = y_i y_j < x_i, x_j >$ e $w = \sum_{i=1}^{m} \alpha_i y_i x_i$. É interessante observar que o vetor solução possui apenas os dados de treinamento e os seus rótulos. Podemos, então, reescrever a função de decisão (equação 3.2) da seguinte forma :

$$g(x) = sinal\left(\sum_{i=1}^{n} \alpha_i y_i < x_i, x > +b\right) \tag{3.5}$$

Note que a forma dual (equação 3.4) apresenta restrições mais simples e permite a representação do problema de otimização em termos de produtos internos (ou produtos escalares, no espaço Euclideano) entre dados. Além disso, as variáveis α_i's são diferentes de zero exclusivamente para aqueles elementos x_i que satisfazem as restrições do problema primal, $y_i(< w, x_i > +b) \geq 1$, com igualdade. Estes pontos são chamados de vetores de suporte, pois, definem o hiperplano pelo fato de seus valores α_i's serem não nulos.



(a) Dados de entrada não separáveis linearmente

(b) Dados mapeados através da função ϕ

(c) Dados de entrada separados por superfícies

Figura 3.3: O método *kernel*

3.1.2 O método kernel

As SVMs lineares são eficazes na classificação de conjuntos de dados linearmente separáveis ou que possuam uma distribuição aproximadamente linear, como mostrado anteriormente. Contudo, mesmo utilizando um SVM com margens suaves[3] há problemas de classificação complexos (figura 3.3) em que dividir dados através de hiperplanos não apresenta solução viável. O objetivo do método *kernel* é contornar essa limitação aplicando uma transformação não linear sobre os dados originais, projetando-os em um espaço de características $\mathcal{H}$. Denotamos por ϕ a função que mapeia o espaço de entrada X para um espaço $\mathcal{H}$, $\phi : X \to \mathcal{H}$. A escolha apropriada de ϕ faz com que o conjunto de elementos mapeados possa ser separado por uma SVM linear simplesmente alterando $< x_i, x_j >$ por $< \phi(x_i), \phi(x_j) >$. Podemos definir, então, uma função *kernel* k com a seguinte propriedade :

$$k(x, x') =< \phi(x), \phi(x') > \tag{3.6}$$

Assim, obtemos uma função de decisão descrita pela equação :

$$g(x) = sinal\left(\sum_{i=1}^{n} \alpha_i y_i < \phi(x_i), \phi(x) > +b\right) = sinal\left(\sum_{i=1}^{n} \alpha_i y_i k(x_i, x) + b\right) \tag{3.7}$$

[2]Essa distância é também conhecida como margem de separação de dados.

[3]SVMs com margens suaves, ao contrário dos SVMs com margens rígidas, são menos sensíveis a ruídos e exceções nos dados.

e ainda temos o seguinte problema de otimização :

$$\begin{aligned} \underset{\alpha}{\text{maximizar}} \quad & -\frac{1}{2}\sum_{i=0}^{n}\sum_{j=0}^{n}\alpha_i\alpha_j y_i y_j k(x_i, y_j) + \sum_{i=1}^{n}\alpha_i \\ \text{sujeito a} \quad & \sum_{i=1}^{n}\alpha_i y_i = 0 \text{ e } \alpha_i \geq\in 0,\ \forall i \in \{1, 2, ...n\} \end{aligned} \tag{3.8}$$

Isso significa que estamos movendo nosso problema de classificação para um espaço com um número maior de dimensões, $\mathcal{H}$, e solucionando-o sem aplicar explicitamente a função de mapeamento ϕ sobre os elementos de entrada. Esse procedimento é conhecido como método *kernel*.

3.1.2.1 Funções kernel válidas

Nem toda função *kernel* pode ser empregada como produto interno num espaço de características. Para que uma função *kernel* seja válida, é preciso respeitar certas definições e lemas: matriz Gram, matriz positiva definida, *kernel* positivo definido e o lema de *Mercer*. Os quais são descritos a seguir.

Definição 16 ***(Matriz Gram)*** *Seja uma função $k : X \times X \to \mathbb{R}$. Uma matriz K cujos elementos são dados por $k_{ij} = k(x_i, x_j)$ é chamada matriz Gram de k com relação ao conjunto de elementos $x_i \in X$, $\forall i \in \{1, 2, ..., n\}$.*

Definição 17 ***(Matriz positiva definida)*** *Uma matriz K é considerada positiva definida se para todo vetor $c \in \mathbb{R}^n$,*

$$\sum_{i=0}^{n}\sum_{j=0}^{n} c_i\overline{c_j}k_{ij} \geq 0$$

Uma matriz Gram (definição 16) é uma matriz $N \times N$, também conhecida como matriz *kernel*. Para um dado conjunto de elementos $X = \{x_1, x_2, ...x_n\}$, K é chamada de matriz Gram de k de acordo com X. Por sua vez, uma matriz positiva definida[4] (definição 17) é uma matriz Gram onde todos seus autovalores são não-negativos. Note ainda que uma matriz positiva definida com valores reais é simétrica.

Definição 18 ***(Kernel positivo definido)*** *Sejam X um espaço não-vazio e k uma função contínua e simétrica sobre $X \times X$. k é um* kernel *positivo definido se e somente*

[4]Essa matriz é também conhecida por matriz positiva semi-definida.

se para todo $n \in \mathbb{N}$ *e para todo* $x_1, x_2, ...x_n \in X$*, a matriz quadrada* K *com* $k_{ij} = (k(x_i, x_j))$*, é positiva (semi-)definida.*

Lema 1 ***(Propriedade de Mercer)*** *Para toda função* kernel *positiva definida* $k \in X \times X$*, existe um mapeamento* $\phi : X \to \mathcal{H}$*, sendo* $\mathcal{H}$ *um espaço Hilbert habilitado com a operação de produto interno. Assim,* $k(u, v) =< \phi(u), \phi(v) >$ *para todo* $u, v \in X$*.* $\mathbb{R}$

Dado um *kernel* positivo definido descrito por uma função k (definição 18), podemos construir um espaço de características, $\mathcal{H}$, no qual k pode ser usado como a operação de produto interno aplicando o lema 1. Por essa razão, os *kernels* positivos definidos são também chamados de *kernels* de Mercer, pois possuem a propriedade referente a este. Tal classe de *kernels* possui ainda propriedades de fechamento interessantes:

- Se k é *kernel* e α é um escalar maior que zero, então, αk é também um *kernel*;
- Se k_1 e k_2 são *kernels*, então, $k_1 + k_2$ é também um *kernel*;
- Se k_1 e k_2 são *kernels*, então o *kernel* definido por $k_1 k_2(x, x') = k_1(x, x')k_2(x, x')$ é válido.

As propriedades de fechamento apresentadas são particularmente importantes para a criação de novas funções *kernel* a partir de outras previamente conhecidas. Considerando a classe de *kernels* positivos definidos, as funções *kernel* mais populares são: *kernel* delta, *kernel* polinomial e *kernel* gaussiano.

Aqui, definimos o método *kernel* que pode ser empregado num espaço *Hilbert* qualquer. Na próxima seção, mostramos como estender esse conceito para dados estruturados, em particular, grafos.

3.1.3 Kernel de convolução R

Recentemente, *kernel* se tornou, também, uma ferramenta importante no contexto de dados não vetoriais [Schölkopf et al., 1999; Gaertner et al., 2003; Schölkopf et al., 2004]. Nos exemplos apresentados até aqui, os dados foram projetados em um espaço euclidiano d-dimensional ($X = \mathbb{R}^d$), entretanto, X pode ser um espaço formado por cadeias de caracteres (*strings*) ou mesmo grafos. Assim, o conceito de *kernel* pode ser aplicado mesmo nesses espaços e, para isso, basta encontrar uma função de transformação (ϕ) que projete os dados estruturados num espaço de características ($\mathcal{H}$) habilitado com produto interno. Em outras palavras, é necessário apenas definirmos uma função *kernel* válida sobre pares de elementos em X.

A classe de *kernels* com foco em dados estruturados foi definida como *kernel* de convolução R [Haussler, 1999]. A ideia central dessa classe é dividir os dados em componentes mais simples e definir *kernels* para tais componentes. Com isso, um *kernel* de convolução busca medir a similaridade de dados estruturados através dos *kernels* definidos sobre os componentes. Formalmente, seja $x \in X$ um dado estruturado e seus n componentes $x' = \{x_1, x_2, ...x_n\}$. Cada parte $x_i \in x'$ pertence ao espaço de representação X_i. Ademais, existe uma função, $R(x, x')$, responsável por determinar se um conjunto de componentes qualquer, x', é uma possível decomposição da estrutura x:

$$R(x, x') = \begin{cases} 1 & \text{se } x_i \in x,\ 1 \leq i \leq n \\ 0 & \text{caso contrário} \end{cases} \tag{3.9}$$

De fato, $R(x, x')$ retorna verdadeiro se e somente se x_1, x_2, x_n são partes de x. Considere, então, uma função $R^{-1}(x)$ que retorna todas as possíveis decomposições de x:

$$R^{-1}(x) = \{x' | R(x, x')\} \tag{3.10}$$

Sejam x e y dois objetos estruturados em X e $R^{-1}(x)$ e $R^{-1}(y)$ os conjuntos das decomposições de x e y, respectivamente. Assumindo que temos definida uma função *kernel* k_i para cada parte $x_i, y_i \in X_i$ e que $R^{-1}(x)$ é enumerável, a similaridade entre x e y pode ser calculado através de um *kernel* de convolução, em termos das funções $k_1, k_2, ...k_n$:

$$k(x, y) = \sum_{\substack{x' \in R^{-1}(x) \\ y' \in R^{-1}(y)}} \mu(x', y') \prod_{i=1}^{n} k_i(x_i, y_i) \tag{3.11}$$

onde $\mu(x, y)$ retorna um valor finito que garante a convergência da função. É sabido que a função $k(x, y)$ é semi-definida positiva sendo, portanto, um *kernel* válido [Haussler, 1999]. Com isso, é possível concluir que as funções k_i's também são válidas com base nas propriedades de fechamento anteriormente apresentadas. Note que, explorando o conceito de convolução R podemos criar variadas funções *kernel* alterando, apenas, os métodos de decomposição das estruturas.

Nesta dissertação estamos interessados em kernels aplicados a grafos, ou, simplesmente, *kernel* de grafos. Uma forma natural e intuitiva de utilizarmos o conceito de convolução nesse cenário é decompor os grafos em subgrafos. Em outras palavras, a similaridade entre dois grafos, G e H, seria dada em função do número comum de

subgrafos entre eles. Este *kernel* de grafos baseado em subgrafos pode ser descrito como:

$$k_{subgrafo}(G, H) = \sum_{G' \subseteq G} \sum_{H' \subseteq H} k_{iso}(G', H') \tag{3.12a}$$

$$k_{iso}(G, H) = \begin{cases} 1 & \text{se G é isomorfo à H,} \\ 0 & \text{caso contrário} \end{cases} \tag{3.12b}$$

Obviamente, a computação da função $k_{subgrafo}$ é NP-difícil, pois esse *kernel* envolve também o problema de isomorfismo de subgrafo (definido pela função *kernel* k_{iso}). Dessa forma, várias propostas surgiram com a intenção de reduzir a complexidade das funções kernel através de decomposições mais simples ou mesmo parciais dos grafos.

Na próxima seção, propomos um novo método de *kernel* baseado em caminhos aproximados. Nosso método tem uma importante vantagem em relação às outras abordagens, pois ele é capaz de capturar relacionamentos indiretos sobre os vértices de um grafo. A computação de todos os caminhos aproximados pode ser custosa por sua alta complexidade. Dessa forma, apresentamos um algoritmo parametrizado para a extração de uma fração dessas subestruturas. Isso torna o processamento dos grafos, isto é, a extração das subestruturas uma tarefa viável e polinomial.

3.2 Kernel baseado em caminhos aproximados

Nesta seção, propomos uma função *kernel* que explora o conceito de convolução, mapeando os grafos para um espaço formado por subestruturas representativas. Diferentemente dos trabalhos anteriores, as subestruturas consideradas aqui são caminhos aproximados nos grafos. Em outras palavras, transformamos um grafo G em um conjunto de caminhos que permitem saltos (*gaps*) entre dois vértices da sequência, capturando relacionamentos diretos e indiretos entre os elementos do grafo. Dois grafos são considerados similares se compartilham um grande número de caminhos aproximados, enquanto grafos distintos serão aqueles que não possuem subestruturas comuns.

A seguir, definimos formalmente o que são caminhos aproximados e apresentamos nossa função de transformação ϕ de grafos que explora tais subestruturas. Por fim, mostramos nossa função *kernel* capaz de calcular a similaridade entre dois grafos quaisquer e sua relação com a similaridade de *Jaccard*.

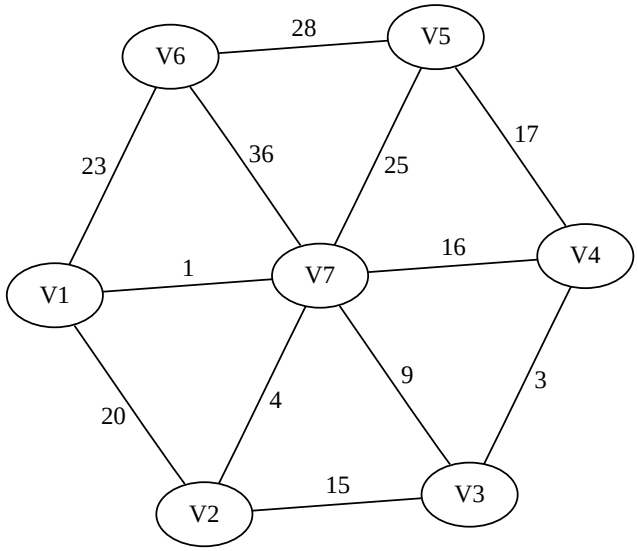


Figura 3.4: Um grafo rotulado G

(a) Sequência S_1 - A,B,C e D (b) Sequência S_2 - A,B,E e D

Figura 3.5: Grafos formados por caminhos simples

3.2.1 Caminhos aproximados

Como descrevemos na seção 2.1, um caminho simples é uma sequência de vértices não repetidos $(v_0, v_1, ...v_k)$, onde há uma aresta entre os vértices v_i e v_{i+1}, para $0 \leq i \leq k-1$. Note que as arestas também são, necessariamente, distintas. Em outras palavras, caminhos são formados por cadeias de vértices e constituem um componente conexo no grafo.

Definição 19 ***Caminho aproximado*** Δ *: Um caminho aproximado* Δ *é uma sequência de vértices distintos* $(v_0, v_1, ...v_k)$*, onde a distância (ou saltos) entre nodos* v_i *e* v_{i+1} *é menor que* Δ*,* $d(v_i, v_{i+1}) \leq \Delta$*, para* $0 \leq i \leq k-1$*.*

Caminhos aproximados (definição 19), propostos nessa dissertação, são sequências de nodos onde são permitidos saltos (*gaps*) entre dois vértices. Obviamente, caminhos simples formam um subconjunto de caminhos aproximados. Por exemplo, na figura 3.4, a sequência de nodos (V1, V2, V3) é um caminho e, também, um caminho aproximado (onde $\Delta = 0$) no grafo. Por outro lado, (V1, V2, V4) não é um caminho simples, mas forma um caminho aproximado 1 em G.

Caminhos aproximados oferecem uma importante vantagem com relação aos caminhos simples: flexibilidade. Observando a figura 3.5, verificamos que utilizando

Caminhos	S_1	S_2
Simples	A, B, C, D AB, BC, CD ABC, BCD ABCD	A, B, E, D, AB, BE, ED, ABE, BED, ABED
Aproximados	A, B, C, D, AB, AC, AD, BC, BD, CD, ABC, ABD, ACD, BDC, ABCD	A, B, E, D, AB, AE, AD, BE, BD, ED, ABE, ABD, AED, BDE, ABED

Tabela 3.1: Comparação entre caminhos simples e caminhos aproximados

caminhos $\Delta = 1$ conseguimos capturar o relacionamento entre os vértices (A,B) e D, através da sequência (A, B, D). Tal relacionamento pertence, claramente, aos dois grafos (sequências) apresentados, porém, essa similaridade não é capturada quando utilizamos caminhos simples. Em outras palavras, os caminhos aproximados conseguem capturar os relacionamentos diretos e indiretos dos vértices de um grafo. A tabela 3.1 mostra uma comparação entre os caminhos simples e os caminhos aproximados encontrados nas sequências apresentadas.

Considere um caminho aproximado qualquer $\rho = (v_1, v_2, ...v_\alpha)$ sob um grafo G. Os vértices v_1 e v_α são ditos nodos extremos do caminho, enquanto os vértices internos, $(v_2, ...v_{\alpha-1})$, são chamados nodos alcançáveis comuns de v_1 e v_α. Os caminhos aproximados evidenciam a associação entre v_1 e v_α baseando nas relações de alcançabilidade entre os vértices que formam a sequência, isto é, os vértices internos fazem parte do caminho de ligação entre os nodos extremos. Por exemplo, o grafo G da figura 3.4 possui diversos caminhos aproximados com $\alpha = 3$ tais como (V1, V2, V3) e (V1, V4, V3) utilizando V1 e V3 como v_1 e v_α, respectivamente. Contudo, existe uma diferença entre eles: o segundo é um caminho aproximado que descreve um relacionamento indireto no grafo, ao contrário do primeiro (caminho simples). Visto que os nodos consecutivos podem não ser vizinhos diretos entre si, um caminho aproximado é de fato uma estrutura ou pivô livre em G.

A seguir definimos as funções de mapeamento (ϕ) entre os grafos e caminhos aproximados.

3.2.2 Função de transformação ϕ

A motivação fundamental das funções de transformação ϕ em grafos é conservar ou manter, no novo espaço de representação, as relações entre os vértices de um grafo. Assim, podemos comparar duas diferentes estruturas baseando nos relacionamentos

conservados e calcular um valor de similaridade de forma eficaz, isto é, com um número baixo de casos falso-positivos e falso-negativos.

Nossa função de transformação ϕ mapeia uma estrutura baseada em grafo num multiconjunto de caminhos aproximados ou pivôs livres – a partir desse momento utilizaremos o termo *pivô*, sempre que possível, para referenciarmos os caminhos aproximados. Como mostramos, os pivôs capturam relacionamentos específicos de alcançabilidade sobre os vértices presentes no grafo. Em nosso trabalho, tais pivôs são descritos em função dos rótulos dos vértices. Como alguns vértices podem ter o mesmo rótulo, um pivô ρ pode ocorrer mais de uma vez em um mesmo grafo. De fato, todos os pivôs possíveis de tamanho l, considerando um conjunto de rótulos R é dado por:

$$A_{R,l} = \{(r_1, r_2, ...r_l) | r_i \in R, 1 \leq i \leq l\} \tag{3.13}$$

Sejam $\mathcal{P}(G)$ o conjunto formado por todos os potenciais pivôs em um grafo G e $\mathcal{P}$ o conjunto de todos os potenciais pivôs (considerando todos os grafos do domínio em questão). Esses conjuntos de pivôs são determinados pelas combinações de rótulos segundo as equações:

$$\mathcal{P}(G) = \bigcup_{l=1}^{|L(G)|} A_{L(G),l} \tag{3.14a}$$

$$\mathcal{P} = \bigcup_{G \in \Gamma} \mathcal{P}(G) \tag{3.14b}$$

Com isso, podemos definir uma função de transformação ϕ que mapeia os grafos no espaço formado por $\mathcal{P}$ da seguinte forma:

$$\phi(G) = \bigcup_{\forall \rho \in \mathcal{P}} \phi_\rho(G) \tag{3.15a}$$

$$\phi_\rho(G) = \begin{cases} 1 & \text{se } \rho \in G \\ 0 & \text{caso contrario} \end{cases} \tag{3.15b}$$

Em outras palavras, dado um grafo G, criamos um vetor binário $\phi(G)$, onde a posição do vetor cabível ao pivô ρ, $\phi_\rho(G)$, recebe o valor 1 se ρ está presente em G e 0, caso contrário. Como mencionado, o conjunto de todos os pivôs existentes em grafo G pode conter repetições, isto é, um mesmo pivô pode ocorrer mais de uma vez. Logo, podemos, alternativamente, estender a representação de vetores binários para vetores multi-valorados (no espaço $\mathbb{Z}^d$) atribuindo a frequência do pivô ρ em G ao invés do valor 1. Para diferenciarmos os dois tipos de transformação, denotamos essas funções

de transformação pelos símbolos ϕ^{bin} (vetores binários) e $\phi^{\mathbb{Z}}$ (vetores multi-valorados).

3.2.3 Função de similaridade kernel

Utilizando funções de mapeamento ϕ definidas anteriormente, podemos criar kernels que retornem um valor de similaridade entre dois grafos quaisquer. Os kernels apresentados a seguir são chamados de *kernel* binário e *kernel* multiconjunto. Enquanto o primeiro explora o espaço de características binário criado pela função ϕ^{bin}, o segundo aplica a transformação $\phi^{\mathbb{Z}}$. Como veremos, tais kernels exploram algumas operações básicas da teoria de conjuntos: interseção e união.

$$k_{bin}(G,H) = \frac{k_c(G,H)}{k_c(G,G) + k_c(H,H) - k_c(G,H)} \tag{3.16a}$$

$$k_c(G,H) =< \phi^{bin}(G), \phi^{bin}(H) >= \sum_{\substack{\rho \in G \\ \rho' \in H}} k_p(\rho,\rho') \tag{3.16b}$$

$$k_p(\rho,\rho') = \begin{cases} 1 & \text{se } \rho = \rho' \\ 0 & \text{caso contrário} \end{cases} \tag{3.16c}$$

O *kernel* binário, descrito pela equação 3.16a, é composto pela combinação de quatro outras funções (k_c, *kernel* contador). A função *kernel* k_c (equação 3.16b) realiza um produto interno sobre os vetores binários no espaço dado por $\mathcal{P}$. De fato, ele retorna o número de posições com valores iguais a 1 em ambos os vetores $\phi(G)$ e $\phi(H)$, ou seja, ele obtém o tamanho do conjunto interseção entre os pivôs presentes em G e H. Note que, enquanto a parte superior da equação 3.16a retorna o número de pivôs comuns nos grafos G e H, o denominador nos dá o tamanho do conjunto união. Com isso, é fácil perceber que a função *kernel* (equação 3.16a), calcula a similaridade de *Jaccard* entre os conjuntos de pivôs dos grafos G e H, respectivamente.

$$sim(G,H) \approx Jaccard(\phi(G),\phi(H))) = \frac{|\phi(G) \bigcap \phi(H)|}{|\phi(G) \bigcup \phi(H)|} \tag{3.17}$$

Claramente, a função *kernel* k_{bin} foi desenvolvida considerando a representação dos grafos em vetores binários. Entretanto, desenvolvemos uma outra função explorando a transformação $\phi^{\mathbb{Z}}$. O *kernel* k_m (equação 3.18) é calculado através dos valores mínimo e máximo das frequências dos pivôs nos grafos G e H que, por sua vez, são

definidos pelos vetores multi-valorados $\phi^{\mathbb{Z}}(G)$ e $\phi^{\mathbb{Z}}(H)$:

$$k_m(G,H) = \frac{\sum_{\forall \rho \in \mathcal{P}} \min(\phi_\rho(G), \phi_\rho(H))}{\sum_{\forall \rho \in \mathcal{P}} \max(\phi_\rho(G), \phi_\rho(H))} \tag{3.18}$$

De fato, k_m generaliza as definições de interseção e união entre conjunto simples (k_{bin}) para multiconjuntos. Ademais, por considerar a frequência dos pivôs, k_m é mais apropriado para comparação grafos de tamanhos diferentes.

Podemos verificar que k_{bin} e k_m são relacionados por, pelo menos, duas formas. Primeiro, considerando vetores com representação binária, k_{bin} e k_m produzem, exatamente, o mesmo valor. Segundo, podemos mapear os vetores de inteiros em vetores binários e, em seguida, aplicarmos a função k_{bin} produzindo o mesmo resultado. Formalmente, seja f_{max} o valor de frequência máxima entre todos os pivôs. Cada vetor d-dimensional $\phi^{\mathbb{Z}} \in \mathbb{Z}^d$ será projetado num espaço de características binário com $d \times f_{max}$ dimensões, onde a posição i do vetor de inteiros é descrito pelas posições $i, i+1, ..., i+f_{max}$ no vetor binário correspondente. Logicamente, o número posições $(i, i+1, ..., i+f_{max})$ com valores iguais a 1 será dado em função do valor da posição i no vetor original. Assim, aplicando o *kernel* k_{bin} nesses vetores binários estendidos, teremos o mesmo valor retornado pela função k_m sobre os vetores de inteiros.

A prova de que k_{bin} e k_m são kernels positivos definidos se baseia, para qualquer inteiro d e qualquer conjunto de vetores binários $x_1, x_2, ...x_l$ ($x_i \in \{0,1\}^d$), a matriz *kernel* (ou matriz de similaridades) K com valores $K_{ij} = k_{bin}(x_i, x_j)$ é positiva semi-definida [Gower & Legendre, 1986]. Portanto, k_{bin} é positivo definido e um *kernel* de Mercer. Transformando os vetores de inteiros em vetores binários estendidos, temos que k_m é equivalente a k_{bin} e, assim, podemos concluir que k_m é também um *kernel* de Mercer [Scholkopf & Smola, 2001].

3.2.4 Algoritmos para extração dos pivôs

O algoritmo 4 descreve um procedimento recursivo para extração os caminhos aproximados dado um grafo G e um vértice inicial $v \in V(G)$ que será, necessariamente, um dos nodos extremos do caminho. Em outras palavras, todos os caminhos gerados por esse algoritmo devem iniciar a sequência de rótulos utilizando o nodo v. A variável c representa a cadeia de rótulos dos vértices do caminho encontrado até um determinado momento da execução e C armazena todas as subestruturas (c's) encontradas.

Enquanto gerar o conjunto de todos os caminhos simples (sem saltos) apresenta

Algoritmo 4: Algoritmo para extração de caminhos aproximados

Entrada: Um grafo G e um vértice $v \in V(G)$ para ser um dos nodos extremos
Saída: O conjunto de todos os pivôs livres gerados a partir de v
Incrementa (vértice u, caminho c, conjunto de caminhos C)

```
1 início
    C = C ∪ c;
    para cada n ∈ N(u) faça
        se dist[n] < 0 então
            dist[n] = dist[u] + 1;
            Incrementa (n, c + L(n), C);
            Incrementa (n, c, C);
        dist[n] = 0;
```

GeraCaminhos ()

```
2 início
    para cada u ∈ V(G) − v faça
        dist[u] = −1;
    dist[v] = 0;
    C = ∅;
    c = L(v);
    Incrementa (v, c, C);
```

um custo de $O(N^N)$ – considerando um grafo com N vértices – , obter todos caminhos aproximados possui uma complexidade de $O(2^N \times N^N) = O((2N)^N)$. Essa análise pode ser confirmada pelo algoritmo 4, onde são realizadas duas chamadas recursivas para cada vértice que é testado no caminho. Uma das chamadas cria caminhos com a presença do vértice em teste e a outra sem a ocorrência dele, caracterizando um salto. A complexidade desse método é $O((2N)^{N-1})$, no pior caso. Note que, para gerarmos todos os caminhos aproximados de um grafo devemos executar o algoritmo $|V(G)|$ vezes, isto é, devemos alterar o vértice de origem. Dessa forma, a computação do conjunto de todos os pivôs livres de um grafo G pode ser infactível mesmo para grafos com um número pequeno de vértices. Contudo, podemos extrair, de maneira polinomial, um subconjunto de pivôs limitando o número de vértices testados.

O algoritmo 5 apresenta uma alternativa para computação de um subconjunto de pivôs livres de um grafo G, limitando os parâmetros α (tamanho dos caminhos) e Δ (tamanho dos saltos). Esses valores limites definem os tipos de pivôs que serão gerados a partir de G. Por exemplo, colocando α_{max} e Δ_{max} iguais a $|V(G)|$, estamos voltando ao algoritmo 4, no qual são extraídos todos os caminhos aproximados iniciados com o vértice v. Por outro lado, considerando o valor limiar Δ_{max} igual a 1 obtemos apenas

Algoritmo 5: Algoritmo para extração de caminhos aproximados parametrizados

Entrada: Um grafo G, um vértice $v \in V(G)$ a ser examinado, o tamanho máximo dos pivôs (α_{max}), o salto máximo permitido (Δ_{max}) e o tamanho máximo dos caminhos ($dist_{max}$)

Saída: O conjunto de todos os pivôs livres gerados a partir de v

Incrementa (vértice u, caminho c, conjunto de caminhos C, tamanho do salto corrente s)

1 **início**

- **se** $|c| > \alpha_{max} \vee s + 1 > \Delta_{max}$ **então**
  - **retorna**;
- $C = C \cup c$;
- **para cada** $n \in N(u)$ **faça**
  - **se** $dist[n] < 0 \wedge dist[u] + 1 < dist_{max}$ **então**
    - $dist[n] = dist[u] + 1$;
    - **Incrementa** (n, c + L(n), C, s);
    - **Incrementa** (n, c, C, s+1);
  - $dist[n] = 0$;

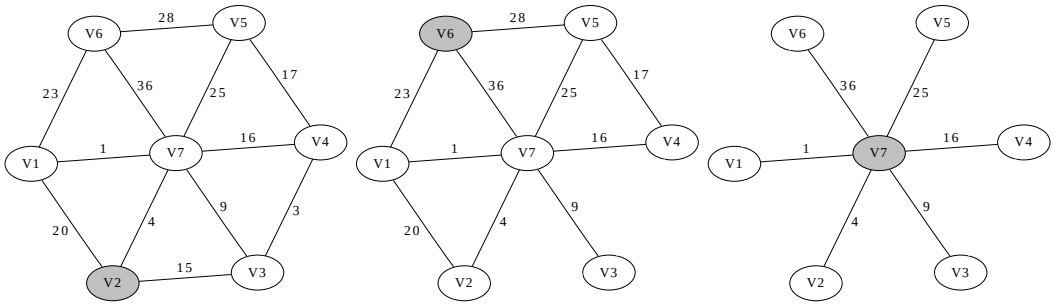
GeraCaminhos ()

2 **início**

- **para cada** $u \in V(G) - v$ **faça**
  - $dist[u] = -1$;
- $dist[v] = 0$;
- $C = \emptyset$;
- $c = L(v)$;
- $s = 0$;
- **Incrementa** (v, c, C, s);

caminhos simples. Note que, ao contrário da primeira abordagem, onde a extração de pivôs inclui todos os vértices e arestas, o algoritmo 5 explora apenas um subgrafo $G_v \subseteq G$. Tal subgrafo, chamado de projeção ou visão de G a partir do vértice de origem v, é delimitado pelo parâmetros $dist_{max}$ que indica a distância máxima permitida entre o vértice de origem e outro vértice qualquer presente na projeção e, consequentemente, nos pivôs. A figura 3.6 mostra três diferentes projeções do mesmo grafo G, considerando como origem os vértices em destaque $V2$, $V6$ e $V7$, respectivamente.

Seja $Proj(G) = \{G_{v_1}, G_{v_2}, G_{v_3}...G_{v_n}\}$ o conjunto de n projeções de um grafo G, dado os vértices $\{v_1, v_2, ...v_n\}$ como origem. O (multi-)conjunto de pivôs associados à projeção a partir de v, G_v, é denotado por $S(G_v)$. Note que, esse conjunto contém somente pivôs livres que possuem o vértice v como um dos extremos. Logo, o conjunto total de pivôs é definido em função do número de diferentes projeções adquiridas de



(a) Visão baseada em V2, $dist_{max} = 3$ (b) Visão baseada em V6, $dist_{max} = 2$ (c) Visão baseada em V7, $dist_{max} = 1$

Figura 3.6: Diferentes visões de G

G. Se consideramos todos os casos possíveis, teríamos $|V(G)|$ projeções, onde $|V(G)|$ é o número de vértices distintos. Então, o subconjunto de pivôs extraídos de G, $S(G)$ pode ser definido como :

$$S(G) = \bigcup_{G_v \in Proj(G)} S(G_v) \tag{3.19}$$

A complexidade do algoritmo 5 é da ordem de $O((2N)^{dist_{max}})$ no pior caso, onde N é o número de vértices em G_v.

3.3 Outras alternativas

Existem vários outros *kernels* propostos para dados estruturados. A seguir, apresentamos o estado da arte com relação aos métodos *kernel* de grafos baseados em caminhos.

3.3.1 Kernel baseado em random walks

Existem dois tipos de *kernel* baseados em *random walks*: (1) *kernel* baseados em produtos de grafos e (2) *kernel* baseado em margens. Tais métodos buscam decompor os grafos em caminhamentos (walks) que permitem a repetição de nodos e arestas e, então, os grafos são comparados considerando o número de caminhos que eles compartilham. Por questão de simplicidade, assumimos, sem perda de generalidade, que todos os grafos a serem analisados possuem o mesmo tamanho com relação ao número de vértices.

Kernel baseados em produtos de grafos: A ideia central dessa técnica é contar o número de caminhos comuns entre os dois grafos envolvidos no cálculo de similaridade [Gaertner et al., 2003].

Definição 20 ***(Produto de grafos)*** *Um produto de dois grafos G e H é denotado por $G_\times = G \times H$. O conjunto de vértices e arestas do grafo produto são dados por:*

$$V(G_\times) = \{(v_i, v'_j) | v_i \in V(G) \wedge v'_i \in V(H) \wedge L(v_i) = L(v'_j)\} \tag{3.20a}$$

$$E(G_\times) = \{((v_i, v'_j), (v'_{i'}, v'_{j'})) | (v_i, v'_j) \in E(G) \wedge (v'_{i'}, v'_{j'}) \in E(H) \wedge L((v'_i, v'_j)) = L((v'_{i'}, v'_{j'}))\} \tag{3.20b}$$

Obviamente, o grafo produto resultante possui $O(|V(G)||V(H)|)$ vértices e $O(|E(G)||E(H)|)$ arestas e usando-o podemos definir o *kernel* baseado em *random walks*:

$$k_\times = \sum_{i,j=1}^{|V(G_\times)|} [\sum_{w=0}^{\infty} \lambda_w A_\times^w]_{ij} \tag{3.21}$$

onde $A_\times$ é a matriz de adjacência do grafo produto $G_\times$ e λ_i é um valor real positivo ($\lambda \in \mathbb{R}$) para todo $i \in \mathbb{N}$. Note que cada posição da matriz A_{ij}^w retorna o número de caminhos que existem entre os vértices i e j. Assim, a função retorna um valor correspondente à soma ponderada do número de sequências de rótulos que ocorrem tanto em G quanto no grafo H. Essa função é considerada um *kernel* canônico a partir do qual são desenvolvidos outras funções tais como o *kernel* exponencial e o *kernel* geométrico para grafos. O algoritmo inicialmente proposto possuía uma complexidade de tempo de $O(|V(G_\times)|^6|)$. Entretanto, usando estruturas de dados avançadas foi mostrado que é possível computar em $O(|V(G_\times)|^3|)$ um *kernel* particular baseado em produto de grafos [Vishwanathan et al., 2006].

Kernel baseado em margens: Diferente do *kernel* baseado em produto de grafos, essa função explora o conceito de cadeias de Markov sobre os grafos a serem comparados [Kashima et al., 2003]. Cadeias de Markov de primeira ordem são descritas por uma matriz T que contém os valores de probabilidade das transições entre os vértices do grafo. Por exemplo, o valor da matriz T_{ij} nos dá a probabilidade de ocorrer uma transição do vértice i para o vértice j. Existe ainda um vetor t cujos valores representam as probabilidades de iniciarmos os caminhos em um determinado vértice.

A forma geral para o *kernel* baseado margens é:

$$k_m = \sum_{\substack{h \in G \\ h' \in H}} k_s(h, h') p(h|G) p(h', H) \tag{3.22}$$

onde h e h' são caminhamentos aleatórios (*random walks*), k_s é um *kernel* es-

pecífico para sequência de rótulos (*string*) e $p(h, G)$ é a probabilidade de gerarmos o caminho h a partir de G dado a matriz de transição (T) e as probabilidades iniciais (t). O custo de execução do *kernel* baseado em margens é de $O(N^6)$, onde N é o número de linhas e de colunas das matrizes de probabilidades referentes aos dois grafos comparados.

3.3.2 Kernel baseado em caminhos simples

Nesse *kernel* a similaridade é calculada com relação ao número de caminhos simples compartilhados [Ralaivola et al., 2005]. Todavia, como já mencionamos, obter todos os caminhos simples de um grafo é custoso ($O(N^N)$, para um grafo com N vértices). Para contornar esse problema utilizou-se um algoritmo de busca em profundidade (alg. 2) para extrair um subconjunto de caminhos simples : à medida que atingimos um novo vértice durante a busca, um novo caminho é encontrado e armazenado. Dessa forma, os caminhos encontrados num grafo G qualquer podem possuir um número de nodos variando de $[1, |V(G)|]$ [5]. Foram propostos três *kernels* baseados em caminhos simples: (1) *kernel* tanimoto, (2) *kernel* min-max e (3) *kernel* híbrido. Os dois primeiros são similares aos *kernels* binário e multiconjunto (discutidos anteriormente), respectivamente. Já o terceiro *kernel* é dado por:

$$k_h(G, H) = \frac{1}{3}((2 - \theta)k_t(G, H) + (1 + \theta)\ \not{k}_t(G, H)) \tag{3.23}$$

onde $\not{k}_t(G, H)$ é o *kernel* tanimoto inverso, isto é, retorna a similaridade baseando nos caminhos que não ocorrem nos dois grafos. Podemos dizer que o *kernel* híbrido (k_h) é combinação de dois *kernels* que medem o numero de caminhos comuns e o número de caminhos perdidos entre as estruturas comparadas. Note que, colocando θ com o valor -1 temos o *kernel* tanimoto. Apesar de ser uma ideia interessante, os autores concluíram que k_h não consegue resultados superiores aos demais.

3.3.3 Kernel baseado em caminhos mínimos

Como o próprio nome já nos diz, esse *kernel* explora um subconjunto de caminhos cujas distâncias são mínimas [Borgwardt & Kriegel, 2005]. Isto é, a similaridade entre dois grafos é dada pela matriz de distâncias mínimas entre os vértices do grafo. Assim, dados dois grafos G e H, criamos duas outras estruturas G' e H' cujas representações são derivadas das matrizes de distância mínima de G e H, respectivamente. O *kernel*

[5] Nos experimentos apresentados pelos autores, os caminhos simples possuem até 10 vértices.

de caminhos mínimos é definido como:

$$k_{cm}(G,H) = \sum_{e \in E(G')} \sum_{e' \in E(H')} k_w^1(e,e') \tag{3.24}$$

onde k_w^1 é um *kernel* sobre arestas que retorna 1 se e e e' existem em ambos grafos, G' e H'. Podemos definir um *kernel* k_w^2, alternativo à k_w^1, que retorna o valor 1 se e somente os pesos das arestas e e e' forem idênticos. O custo computacional dessa função é de $O(N^4)$, onde N^2 é o número máximo de arestas de cada grafo G' e H'.

3.4 Discussão

Como foi visto, vários métodos *kernel* voltados para dados estruturados foram propostos. As abordagens apresentadas nessa seção exploram, de alguma maneira, caminhos sobre os grafos. Além do custo elevado de processamento, tais métodos possuem outras limitações. Por exemplo, as técnicas baseadas em *random walks* (*kernels* baseados em produto de grafos e os *kernels* baseado em margens) possuem duas limitações principais. Primeiro, os caminhamentos sobre o grafo, que são aleatórios, podem pertencer a uma mesma região da topologia, ou seja, não temos garantia de cobrir toda a estrutura e caminhos. Já o segundo problema tem relação com o fator λ presente nas funções *kernel* que atribuem pesos menores aos caminhos mais longos. Isso faz com que os caminhos menores dominem o valor de similaridade entre dois grafos. As técnicas baseadas em caminhos simples e caminhos mínimos, por sua vez, são *kernels* que buscam apenas relações diretas entre os vértices de um grafo, ou seja, apresentam limitações quanto à flexibilidade dos padrões extraídos das estruturas.

Existem ainda outras funções como o *kernel* baseado em árvores e o *kernel* baseado em ciclos [Ramon & Gaertner, 2003; Horváth et al., 2004]. Tais métodos representam os grafos através de subestruturas no formato de árvores e ciclos, respectivamente. Ambos *kernels* podem possuir uma complexidade até mesmo maior do que as técnicas descritas anteriormente, sendo que o *kernel* baseado em ciclos tem sua aplicação voltada para cenários específicos. De fato, em banco de dados químicos os ciclos são bastante importantes para análise das funções moleculares.

Apesar do alto custo computacional das funções *kernel*, muitas vezes o domínio de aplicação fornece grafos pequenos tornando viável o cálculo de similaridade entre dois grafos quaisquer. Podemos, por exemplo, indexar os grafos através dos caminhos que os compõem e realizar as tarefas de interseção e união facilmente. Todavia, considerando grandes bases de dados (um grande conjunto de grafos), essa estrutura de índice pode

consumir muito espaço em disco, seja pelo grande número grafos, seja pelo grande número de caminhos distintos.

Apresentamos na próxima seção, um método de *hashing* capaz de sumarizar o conjunto de subestruturas de um dado grafo num vetor (assinatura) de tamanho n. Podemos, então, selecionar um tamanho tão pequeno quanto desejado a ponto de analisar todas as assinaturas em memória principal. O valor de similaridade entre os grafos é calculado através de suas assinaturas e possui garantias estatísticas de aproximação.

Capítulo 4

Assinatura de grafos via hashing

Sabendo que o número de pivôs extraídos de um grafo por nossa função de transformação é potencialmente $(2N)^{N-1}$ (N é o número de vértices do grafo), os cálculos de interseção e união que são necessários para a comparação entre os grafos podem exigir um alto poder de processamento. Ainda nesse cenário, a quantidade memória principal pode ser insuficiente para armazenar os dados, o que reduz consideravelmente o desempenho das aplicações.

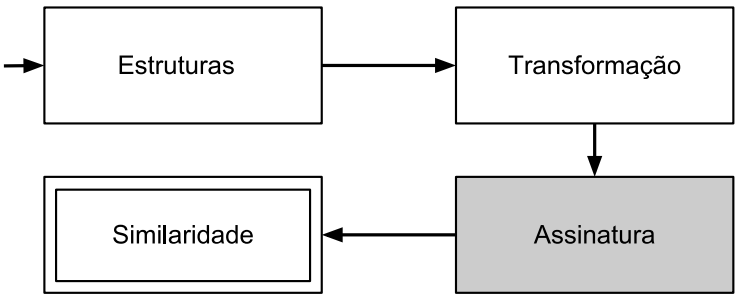


Figura 4.1: Fase para geração das assinaturas

Nessa parte do trabalho, utilizamos funções de assinatura para alcançar eficiência durante a análise de similaridade entre os grafos. As funções de assinatura convertem o (multi-)conjunto de pivôs em uma quantidade fixa de informação. Podemos pensar nas funções de assinatura como um operador de sumarização (*fingerprint*) sobre os grafos. A ideia é produzir assinaturas que são pequenas suficiente para serem colocadas em memória principal e, mais importante, que o valor de similaridade entre duas assinaturas seja o mesmo (ou mais próximo possível), considerando a similaridade dos conjuntos de

pivôs dos grafos correspondentes (Eq. 3.17). A figura 4.1 destaca a fase de assinatura no diagrama de execução do nosso método. Tal fase, descrita nesse capítulo, converte os conjuntos de pivôs em assinaturas, que podem ser alocadas em memória principal.

A ideia chave da função de assinatura é um esquema de *hashing*. *Hashing* é uma operação fundamental, em vários domínios, tais como sistemas de banco de dados, mineração de dados, visão computacional e ciência de redes. Tal estratégia mapeia conteúdos de informação – geralmente grandes e de variáveis magnitudes – em dados pequenos e de tamanho fixo aplicando-se funções *hash*. *Hashing* é largamente usado como ferramenta para altear a eficiência de pesquisas, por exemplo, buscar por itens específicos num banco de dados [Agrawal et al., 2002], detectar registros duplicados [Elmagarmid et al., 2007; Ke et al., 2004], e também localizar dados relacionados a um outro previamente conhecido [Buehrer & Chellapilla, 2008]. Neste trabalho, adotamos o método *Min-hash* para geração das assinaturas dos grafos. Através desse método, conseguimos estimar a similaridade de *Jaccard* entre conjuntos.

É importante ressaltar que o emprego de assinaturas no contexto de grafos e dados (semi-)estruturados é explorado em trabalhos encontrados na literatura [Ralaivola et al., 2005; Tatikonda & Parthasarathy, 2010], como será discutido na seção 4.2. Contudo, ainda não existem pesquisas dedicadas ao estudo de assinaturas baseadas em *Min-hash* aplicadas à grafos. A seguir, discutimos a abordagem *Min-hash* além de outras alternativas para sumarização de conjuntos.

4.1 O método Min-hash

Existem variadas formas de se obter assinaturas de conjuntos, sendo uma delas a técnica *Min-hash* [Cohen, 1997; Broder et al., 1998; Cohen et al., 2001]. Existem duas razões principais para a escolha desse técnica: (1) *Min-hash* vem sendo empregada com êxito em diversos contextos[1], (2) possui garantias teóricas de qualidade e (3) pode ser usada para obter um valor aproximado para a similaridade de *Jaccard* entre conjuntos.

A função das assinaturas é estimar a similaridade entre conjuntos de forma precisa e eficaz. Em outras palavras, dois conjuntos são parecidos se, e somente se, suas respectivas assinaturas forem também similares. Podemos, então, estimar a similaridade de dois grafos (entre os conjuntos de pivôs) através de suas assinaturas:

$$sim(G,T) = \frac{|S(G) \bigcap S(T)|}{|S(G) \bigcup S(T)|} \approx \frac{|sig(G) \bigcap sig(T)|}{|sig(G) \bigcup sig(T)|} \tag{4.1}$$

[1]Mineração de padrões frequentes e similaridade de árvores são alguns desses contextos [Cohen et al., 2001; Tatikonda & Parthasarathy, 2010].

Uma grande vantagem das assinaturas é que podemos mantê-las em memória principal e realizar operações de comparação de forma eficiente, pois elas possuem um tamanho significativamente menor do que o conjunto original de elementos. A seguir descrevemos o processo de geração dessas assinaturas e mostramos que as estimativas dadas por esse método são coerentes, ou seja, possuem poucos casos falso-positivos e falso-negativos.

4.1.1 Comparação eficiente entre grafos

Dado um grafo G, primeiro extraímos o (multi-)conjunto de pivôs $S(G)$ de acordo com a seção anterior. Construímos então uma assinatura para G, $sig(G)$, usando *Min-hash*. Para isso, realizamos um *hash* de cada pivô $p \in S(G)$ usando a seguinte função *hash*:

$$ph(\rho) = a_1 \cdot v_1 + a_2 \cdot v_2 + ...a_k \cdot v_k (mod\ P) \tag{4.2}$$

onde p é o pivô livre de tamanho k, $v_1, v_2, ...v_k \in L(G)$, P é um número primo grande, e $a_1, a_2, ...a_k \in \mathbb{Z}_P$. É importante ressaltar que os rótulos são usados na equação acima, portanto, transformamos rótulos alfanuméricos para números. Essa função *hash* é suficientemente aleatória e nos retorna uma baixa probabilidade de colisão [Gionis et al., 1999]. Concluindo, convertemos cada subestrutura pivô a um número entre 0 e $P-1$ transformando o conjunto de pivôs, $S(G)$, em um conjunto de valores inteiros, I_G.

A ideia fundamental no esquema de *Min-hash* é permutar aleatoriamente o universo de inteiros (em nosso caso, valores entre 0 e $P-1$) e o valor de *hash* de um conjunto de inteiros I ($h(I)$) é definido pelo primeiro valor da permutação que esteja presente em I. Em outras palavras, o índice do primeiro inteiro retornado pela permutação π_i e que está presente em I é produzido como seu valor de *Min-Hash* $h_i(I)$. Assim, temos que a probabilidade de dois conjuntos de pivôs produzirem o mesmo valor de *hash* é igual à similaridade de *Jaccard* desses conjuntos[Broder et al., 1998; Cohen et al., 2001]:

$$Pr[h(I) = h(I')] = sim(I, I') = \frac{|I \cap I'|}{|I \cup I'|} \tag{4.3}$$

Note que, por ser probabilístico, podemos ter casos falso-positivos e falso-negativos. Entretanto, esses casos podem ser eliminados repetindo o processo k vezes, resultando em *k-Min-Hashes*:

$$sig(G) = \{h_i(I_G) = \min_{\forall p \in I_G}(\pi_i(p))\},\ 1 \leq i \leq k \tag{4.4}$$

onde π_i são funções de permutação aleatórias sobre o universo de inteiros. Para universos de tamanho grande, a construção explícita dessas permutações podem ter um custo computacional elevado. Contudo, quando o universo em questão possui elementos da forma $\{0, 1, 2, ...P-1\}$ podemos considerar uma família de funções permutação da forma [Broder et al., 1998]:

$$\pi_i(x) = a_i \cdot x + b_i (mod\ M)$$

onde $a_i \in \mathbb{Z}_M^*$, $b_i \in \mathbb{Z}_M$ e M é um número primo que não é menor que o tamanho do universo de inteiros, logo, $M \geq P$. O valor de similaridade entre dois grafos G e H pode, então, ser estimado utilizando suas respectivas assinaturas:

$$sim(G, H) \approx sim(sig(G), sig(H)) = \frac{|\{i | 1 \leq i \leq k \ \wedge \ h_i(I_G) = h_i(I_H)\}|}{k} \tag{4.5}$$

Em outras palavras, a similaridade entre as assinaturas é estimado pelo número de *hashes* comuns entre os conjuntos de inteiros, I_G e I_H. Mostraremos agora que, apesar de ser uma solução aproximada, a técnica de *Min-hash* nos dá garantias sobre sua efetividade. Seja s um limiar para dizermos que dois grafos possuem um alto grau de similaridade, ou seja, se $sim(G, H)) \geq s$, então, G e H são muito parecidos. Considere, ainda, uma constante c como limite inferior de s. Podemos concluir que os casos de falso-positivos e falso-negativos são infrequentes quando usamos a função $sim(sig(G), sig(H)))$ através do seguinte lema [Charikar, 2002]:

Lema 2 *Seja* $0 < \lambda < 1$, $\mu > 0$ *e* $k \geq 2\lambda^{-2}c^{-1}log\mu^{-1}$. *Então, dado dois grafos quaisquer,* G *e* H, *temos duas propriedades :*

1. *Se* $sim(G, H) \geq s \geq c$, *então,* $sim(sig(G), sig(H)) \geq (1 - \lambda)s$ *com probabilidade mínima de* $1 - \mu$;

2. *Se* $sim(G, H) \leq c$, *então,* $sim(sig(G), sig(H)) \leq (1 - \lambda)c$ *com probabilidade mínima de* $1 - \mu$.

O lema 2 nos diz que, considerando um valor k (tamanho das assinaturas) suficientemente grande, se o valor da similaridade entre os conjuntos de subestruturas de dois grafos é alto ($sim(G, H) \geq s$), então, esses conjuntos terão uma grande fração de valores *Min-hash* idênticos. Da mesma forma, se o valor de similaridade for baixo ($sim(G, H) \leq c$), então, apenas uma pequena fração das assinaturas será idêntica.

Note que a estimativa dada pela equação 4.5 está ligada à função kernel k_{bin} (descrita pela equação 3.16a). Entretanto, assim como o conjunto de pivôs, $S(G)$, o conjunto de inteiros, I_G, também pode possuir valores repetidos. Podemos, então, armazenar o valor de multiplicidade do valores de *hash*, $h_i(\cdot)$, para calcular as operações interseção e união das assinaturas usando o conceito de multiconjunto, retornando assim uma estimativa para a função kernel k_{multi}:

$$sim(G,H) \approx sim(sig(G), sig(H)) = \frac{\sum_{1 \leq i \leq k} \min(|h_i(I_G)|, |h_i(I_H)|)}{\sum_{1 \leq i \leq k} \max(|h_i(I_G)|, |h_i(I_H)|)} \tag{4.6}$$

Algoritmo 6: Algoritmo para geração de assinatura

Entrada: Um conjunto de subestruturas $S(G)$ de um grafo G
Saída: Uma assinatura para um grafo G
GeraAssinatura ()

```
1 início
    para cada i ∈ 1, 2, ...k faça
        ass[i] = ∞;
    para cada c ∈ S(G) faça
        para cada i ∈ 1, 2, ...k faça
            se ass[i] > h_i(Int(c)) então
                ass[i] = h_i(Int(c));
    retorna ass;
```

O algoritmo 6 descreve o método para geração das assinaturas dado um grafo G qualquer. A função Int mapeia uma subestrutura num valor inteiro como mostrado pela equação 4.2. Assim, primeiro convertemos as subestruturas em um conjunto de números inteiros (I_G) e, depois, aplicamos cada função h_i sobre I_G e capturamos os valores mínimos de *hash* formando a assinatura. Considerando o algoritmo 5 para extração dos pivôs, temos, no pior caso, $|I_G| = O((2N)^{d_{max}})$. Logo, estabelecendo k funções *hash*, temos uma complexidade de tempo de $O(k(2N)^{d_{max}})$ no pior caso. Quanto à questão de espaço utilizado, é fácil perceber que podemos manter n assinaturas em memória principal por um custo de $O(nk)$ de espaço. Visto que k é um valor usualmente pequeno (entre 16 e 1024), podemos computar em memória principal diversos algoritmos baseados na operação de comparação entre grafos. Obviamente, a ideia é manter a assinatura dos grafos como um metadado sem a necessidade de processamento futuro. Ao chegar novas estruturas na base, podemos gerar suas assinaturas

simplesmente conhecendo os valores primos das funções *hash* utilizados anteriormente.

4.2 Outros algoritmos

As próximas seções discutem outros algoritmos de *hashing* utilizados para sumarizar dados. Esses algoritmos podem ser também aplicados sobre o conjuntos de pivôs, porém, a técnica de *Min-hash* foi selecionada por motivos já citados.

4.2.1 Assinatura binária simples

Talvez, a forma mais fácil de gerarmos uma assinatura é explorando uma técnica que chamamos de assinatura simples [Ralaivola et al., 2005]. De forma similar ao nosso trabalho, convertemos cada estrutura em um inteiro, através de uma função *hash*:

$$ph(s) = Int(s)(mod\ P) \tag{4.7}$$

onde s é uma estrutura, Int é uma função que retorna um valor inteiro ($\mathbb{Z}$) sobre s e P é um número primo. Feito isso, podemos representar um grafo qualquer G através de um conjunto de inteiros, I_G. Define-se, então, uma função $m : \mathbb{Z} \rightarrow \{1, 2, ..., l\}$, onde l é o tamanho das assinaturas a serem geradas. Aplicamos a função m sobre cada valor em I_G e estabelecemos o valor 1 às posições retornadas pela $m(\cdot)$. Uma vez que uma posição da assinatura é atingida por m, ela não pode ser mais alterada.

Uma função m intuitiva e simples é aplicar o módulo l sobre os valores de *hash*. Por outro lado, podemos ainda definir um conjunto de funções m_i (usualmente, $1 \leq i \leq 4$) e, assim, cada valor presente no conjunto I_G é responsável por determinar o valor de mais de uma posição na assinatura de G. A complexidade tempo dessa função é $O(i * |I_G|)$ ou $O(i * (2N)^{N-1})$, assumindo um total de $(2N)^{N-1}$ estruturas e *hash* perfeito sobre elas. Note que, se garantirmos que as funções m são uniformes, temos a técnica de filtro *bloom* (bloom filter) [Kirsch & Mitzenmacher, 2006].

4.2.2 Locality sensitive hashing

Locality sensitive hashing (LSH) constitui uma família de métodos baseados em funções *hash*. LSH's têm sido usadas em dois contextos relacionados: (1) busca por vizinhos próximos, onde dois objetos possuem o mesmo valor de *hash* se eles estão próximos no espaço de características e (2) comparação de objetos, onde as probabilidades de colisão são usadas para estimar o valor de similaridade entre os dados [Indyk & Motwani, 1998; Charikar, 2002]. Estamos interessados no segundo caso.

Definição 21 ***(Locality sensitive hashing - LSH)*** *A estratégia LSH é dada por uma família $\mathcal{F}$ de funções hash aplicadas sobre objetos de um domínio X tal que:*

$$Pr_{h\in\mathcal{F}}[h(x) = h(x')] = sim(x, x') \tag{4.8}$$

onde $x, x' \in X$ e $h \in \mathcal{F}$.

A estratégia LSH é descrita pela definição 21. De fato, podemos concluir que o método *Min-hash* anteriormente apresentado é um caso particular de LSH correspondente à similaridade de *Jaccard*. É possível estimar outras medidas de similaridade como, por exemplo, distância de cosseno ($sim(x, x') = x^T x'$, onde x e x' são vetores), usualmente aplicadas para análise de documentos, ou distância EMD (*earth mover distance*), comum nas áreas de computação gráfica e visão computacional) [Charikar, 2002].

Lema 3 *Se uma função de similaridade qualquer $sim(x, x')$ pode ser estimada por uma família de funções LSH (definição 21), então a função de distância dada por $sim'(x, x') = 1 - sim(x, x')$ satisfaz a desigualdade triangular.*

Contudo, nem todas as métricas de distância ou similaridade admitem o uso de funções LSH. Para que isso seja possível, a métrica deve respeitar o lema 3 [Charikar, 2002]. O método *Min-hash* claramente segue esse lema.

4.3 Discussão

Embora a ideia de *hashing* seja antiga – mais de 50 anos atrás – *hashing* de conjuntos de tamanhos variáveis é um tópico recente de pesquisa. Métodos de *hashing* têm sido usados com sucesso em diferentes cenários tais como agrupamento de páginas Web, mineração de padrões em documentos, comparação de sequências e árvores e análise de objetos geométricos e imagens [Broder et al., 1997; Cohen et al., 2001; Aho et al., 1996; Tatikonda & Parthasarathy, 2010; Wolfson & Rigoutsos, 1997; Kulis & Grauman, 2009]. Infelizmente, pouco esforço foi realizado em consideração a *hashing* em dados estruturados – principalmente grafos – o que ainda não é bem entendido na comunidade científica.

Nesse trabalho, propomos o uso das técnicas de *hashing*, em particular, a técnica *Min-hash*, para calcular a similaridade de grafos de forma eficiente. *Min-hash* pertence à família de algoritmos *locality sensitive hashing*. Elas podem estimar a similaridade entre conjuntos de dados para diferentes métricas como, por exemplo, a similaridade

de *Jaccard* e cosseno. Uma vantagem dessas técnicas em relação às abordagens mais simples é o processo de estimativar a similaridade entre os dados a partir das assinaturas possui uma teoria sólida, que garante uma aproximação mínima com relação à solução ótima.

Capítulo 5

Avaliação experimental

Neste capítulo, apresentamos um conjunto de avaliações empíricas, onde medimos a eficiência e eficácia dos diferentes operadores de transformação (algoritmo para extração dos pivôs) propostos. Tais operadores exploram as subestruturas definidas como caminhos mínimos (capítulo 3). Além disso, investigamos as funções de assinatura baseadas em esquemas *hashing* (capítulo 4).

Iniciamos nosso estudo experimental mostrando a eficácia das assinaturas para estimar o valor de similaridade entre os conjuntos de pivôs dos grafos a serem comparados. Para isso, medimos os valores das funções *kernel* utilizando os conjuntos de pivôs diretamente e comparamos com os valores aproximados dados pelas assinaturas (foram examinadas as similaridades dos grafos da base de dados entre si). De fato, o coeficiente de *Pearson* nos indica que esses dois conjuntos de valores são altamente correlacionados [Weiss, 2005]. Posteriormente, aplicamos nosso algoritmo sobre dois problemas bem conhecidos : (1) recuperação de grafos similares e (2) classificação de grafos. No primeiro caso, avaliamos o desempenho das funções de similaridade propostas na recuperação de grafos similares e mostramos que, em geral, nossa estratégia retorna o grafo alvo nas primeiras posições do *ranking* de similaridade. Já no segundo, testamos nosso método em cenários reais para classificação automática de grafos, isto é, as estruturas são organizadas, com base nos relacionamentos dos vértices, em categorias definidas por especialistas. Verificamos que nossa abordagem atinge resultados superiores (de 1% até 7% de ganho de acurácia) às técnicas de classificação encontradas na literatura.

Parâmetro	Descrição
$dist_{max}$	Profundidade máxima das projeções obtidas de um grafo
Δ	Salto máximo permitido nos caminhos aproximados
k	Tamanho do vetor assinatura de um grafo
c	Porcentagem de alteração sobre o grafo original

Tabela 5.1: Parâmetros avaliados

5.1 Metodologia

Nesta seção, descrevemos a metodologia utilizado para geração dos resultados e avaliação dos resultados.

5.1.1 Algoritmos e parâmetros

Os algoritmos estudados nesse capítulo são: (1) o algoritmo para extração dos pivôs dos grafos e (2) o algoritmo para geração das assinaturas. Nosso interesse é avaliar o impacto dos parâmetros nesses dois casos e suas relações com os valores de similaridade dos grafos provindos de diversos cenários. Tais técnicas estão descritas nos capítulos 3 e 4, respectivamente. Já os parâmetros e as bases de dados testados são discutidos a seguir.

Os parâmetros dos algoritmos investigados estão descritos na tabela 5.1. Como podemos perceber, $dist_{max}$ e Δ são parâmetros que pertencem ao algoritmo de extração de pivôs, enquanto, k está ligado ao algoritmo de geração de assinatura. O parâmetro c será empregado nos experimentos com relação à recuperação de grafos similares. Esse parâmetro é usado para definir a porcentagem de alterações realizadas nos grafos e será devidamente detalhado quando necessário. Note que atribuindo o valor zero ao parâmetro Δ, temos o método *kernel* baseado em caminhos simples [Ralaivola et al., 2005] que será usado como *baseline*.

É importante ressaltar que as configurações de parâmetros avaliadas foram executadas cincos vezes para aumentar a confiabilidade dos resultados. Além disso, utilizamos métodos de validação estatística (*test-t*), quando necessário, para demonstrar a superioridade ou inferioridade das abordagens (principalmente no contexto de classificação de grafos).

5.1.2 Base de dados

Ao todo, foram usadas quatro bases de dados reais em nossos testes. A seguir, descrevemos de forma sucinta cada uma delas. A tabela 5.2 contém um sumário das características principais de cada um das bases de dados.

- **Mutag :** Essa base de dados contém, originalmente, 230 compostos químicos aromáticos e heteromáticos [Debnath et al., 1992]. Esses compostos foram testados sobre a bactéria *Salmonella typhimurium*, e foram catalogados casos positivos e negativos de mutagenicidade. A tarefa de classificação é inferir se uma dada molécula exerce ou não um efeito mutagênico. Para isso, foram selecionados 188 compostos (125 casos positivos e 63 negativos) que são considerados adequados para a tarefa e aprendizado [Debnath et al., 1992]. Cada molécula é representada como um grafo, onde os vértices são átomos e as arestas representam interações entre eles.

- **PTC :** O conjunto de grafos PTC (*Predictive Toxicology Challenge*) reporta a carcinogenicidade de centenas de compostos químicos em ratos e camundongos [Helma et al., 2001]. Temos quatro bases de dados desse tipo : Camundongos Machos (CM), Camundongos Fêmeas (CF), Ratos Machos (RM) e Ratos Fêmeas (RF). A tarefa de classificação se resume em predizer se um dado composto provoca um efeito carcinogênico, ou seja, temos uma classificação binária para cada tipo de base. A representação de moléculas através de grafos é similar à base de dados Mutag.

- **PTE :** A base de dados PTE[1] (*Predictive Toxicology Evaluation*) é constituída de 340 compostos químicos classificados, de maneira similar à PTC, como carcinogênico ou não [Srinivasan et al., 1997; Kuramochi & Karypis, 2001]. Novamente, os testes foram aplicados em roedores.

- **NCI-CA :** Tais dados, disponibilizados pelo Instituto Nacional do Câncer dos Estados Unidos (*National Cancer Institute* - NCI), testam a ação de, aproximadamente, 40.000 compostos químicos contra o vírus HIV. Os compostos são então divididos de acordo com suas capacidades anti-HIV, assim, a tarefa de classificação é prever se as moléculas inibem ou não a expansão do vírus. Existem três categorias: (CA) comprovados ativos, (CM) comprovados moderadamente ativos e (CI) comprovados inativos. Neste trabalho, consideramos apenas a primeira categoria.

[1] http://www.comlab.ox.ac.uk/activities/machinelearning/PTE/

bases		#grafos	# médio de nós	# médio de arestas	#rótulos
Mutag		188	17,93	19,79	7
PTE		340	27,02	27,40	66
PTC	CM	336	25,03	25,39	21
	CF	349	25,25	25,62	19
	RM	344	25,56	25,96	19
	RF	351	26,08	26,53	20
NCI-CA		422	39,60	42,30	21
MCF-7		2294	60.00	63.00	27
MOLT-4		3140	57.00	60.00	34
OVCAR-8		2079	62.00	64.00	33

Tabela 5.2: Sumário das características das bases de dados

- **MCF-7, MOLT-4, OVCAR-8 :** O projeto *PubChem*[2] disponibilizou estas bases de dados formadas por compostos químicos ativos em testes com células cancerígenas (cancêr de mama, leucemia e cancêr de ovário, respectivamente).

5.2 Efetividade das assinaturas

O intuito do uso das assinaturas é reduzir o tempo de execução de comparação entre os grafos. Contudo, é importante que o valor de similaridade retornado pelas assinaturas esteja de acordo com o valor obtido através dos conjuntos de subestruturas ou pivôs.

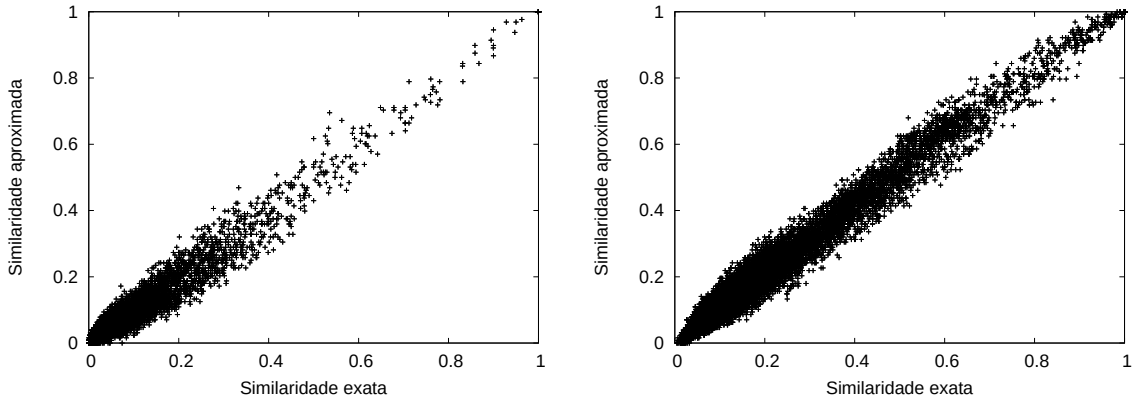
Para mostrar que nosso método de assinatura é eficaz, avaliaremos a dependência entre as similaridades ótimas – obtidas diretamente dos conjuntos de pivôs pelas funções *kernel* – e aproximadas – obtidas pelo método de *hashing*. Podemos dizer que as assinaturas conseguem substituir satisfatoriamente os conjuntos de subestruturas no processo de comparação de grafos se os valores de correlação entre as similaridades, ótima e aproximada, forem relevantes[3].

Nesta parte do trabalho, utilizamos as bases de dados PTE e NCI-CA. As figuras 5.1 e 5.2 mostram gráficos de dispersão que contrastam os valores de similaridades ótimas (eixo X) e os valores estimados pelas assinaturas (eixo Y). Note que os pontos tendem a formar uma reta crescente, evidenciando dependência entre as variáveis. Percebemos, ainda, que as assinaturas estimam melhor os valores retornados pelo *kernel* binário (figs. 5.1a e 5.1b) do que os resultados obtidos pelo *kernel* multiconjunto (figs. 5.2a e 5.2b), pois os pontos do primeiro estão mais concentrados na diagonal do gráfico.

[2]http://pubchem.ncbi.nlm.nih.gov

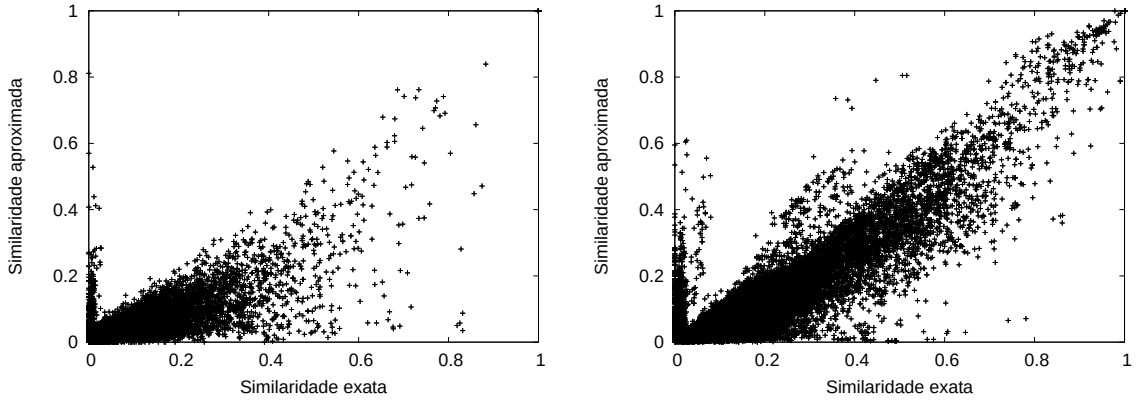
[3]O conceito de relevância nesse cenário depende do método de correlação utilizado.

Acreditamos que a razão para tal fato é que os multiconjuntos são frequentemente maiores do que os conjuntos simples (utilizados pelo *kernel* binário), sendo, portanto, mais difíceis de estimar. Tal fato pode ser facilmente contornado aumentando o tamanho das assinaturas. Os parâmetros utilizados para gerar esses resultados foram $dist_{max} = 10$, $\Delta = 0$ e $k = 128$ (tab. 5.1).



(a) PTE (b) NCI-CA

Figura 5.1: Comparação entre os valores de similaridade utilizando *kernel* binário



(a) PTE (b) NCI-CA

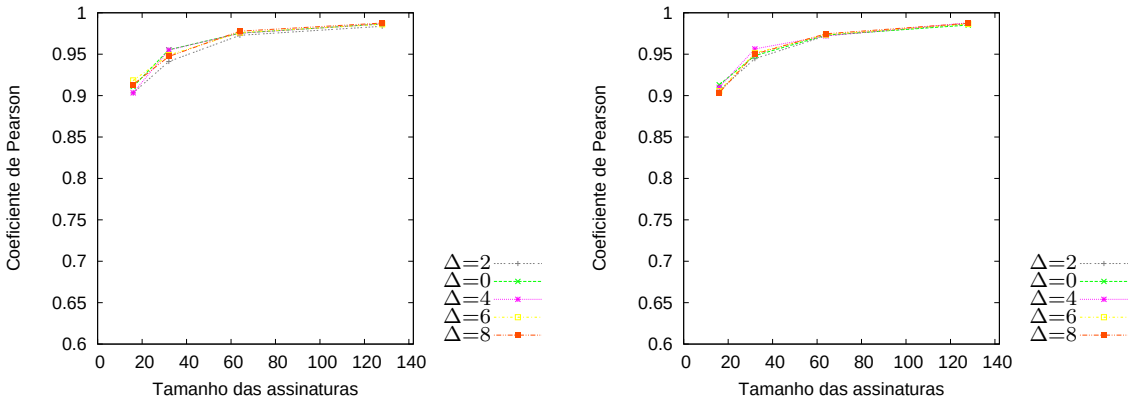
Figura 5.2: Comparação entre os valores de similaridade utilizando *kernel* multiconjunto

Existem outras técnicas usadas para mensurar relacionamentos estatísticos entre duas ou mais variáveis independentes ou valores observados. Nesse contexto, uma abordagem comumente empregada é o coeficiente de correlação de *Pearson*. Tal coeficiente captura relacionamentos lineares (se houverem) entre duas variáveis quaisquer. *Pearson* retorna um valor entre $[-1, 1]$: Um valor próximo das extremidades (1 ou -1) nos

Correlação	Negativa	Positiva
Nenhuma	-0,09 a 0,0	0,0 a 0,09
Pequena	-0,3 a -0,1	0,1 a 0,3
Média	-0,5 a -0,3	0,3 a 0,5
Alta	-1,0 a -0,5	0,5 a 1,0

Tabela 5.3: Categorias para os valores de correlação de *Pearson*

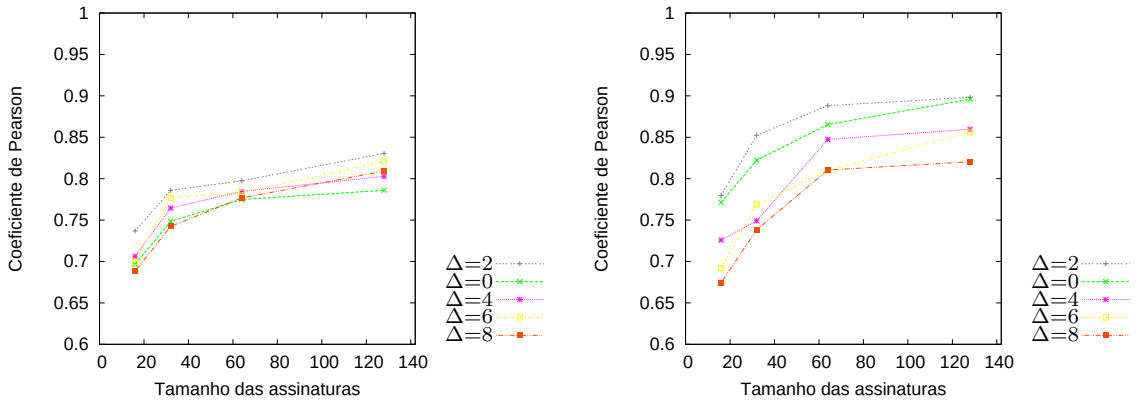
diz que existe uma alta correlação entre as funções e um valor próximo de zero indica que as variáveis são independentes. A tabela 5.3 apresenta uma categorização para os diferentes valores de *Pearson* [Weiss, 2005]. As figuras 5.3 e 5.4 mostram valores de *Pearson* para as bases PTE e NCI-CA, considerando os *kernels* binário e multiconjunto, respectivamente. O tamanho das assinaturas (k) e o tamanho dos saltos dentro das subestruturas (Δ) foram variados, fixando o parâmetro $dist_{max}$ em 10.



(a) PTE (b) NCI-CA

Figura 5.3: Valores de *Pearson* utilizando *kernel* binário

Considerando a classificação apresentada pela tabela 5.3 e as figuras 5.3 e 5.4, podemos afirmar que existe uma correlação alta e positiva entre os valores de similaridade dados pelos conjuntos de pivôs e pelas assinaturas. Além disso, observamos que o coeficiente de *Pearson* é sempre maior do que $0,5$, independentemente do tamanho das assinaturas (k) e dos saltos permitidos (Δ). Comparados ao *kernel* multiconjunto, os coeficientes obtidos pelo *kernel* binário foram superiores nas duas bases avaliadas – em geral, com uma diferença acima de $0,10$. Acreditamos que essa diferença é dada em razão dos tamanhos dos conjuntos, isto é, quando temos um número maior de elementos, como no caso de multiconjuntos, a estimativa pode perder precisão com assinaturas de tamanho pequeno. Note ainda que os valores de *Pearson* aumentam à medida que elevamos o tamanho das assinaturas. Por exemplo, na figura 5.3a o coeficiente aumenta



(a) PTE (b) NCI-CA

Figura 5.4: Valores de *Pearson* utilizando *kernel* multiconjunto

cerca de $0,05$ quando passamos o tamanho das assinaturas de 16 para 128 dimensões. Essa diferença chega a $0,15$ (para $\Delta = 4$) na figura 5.4b, evidenciando a importância do tamanho das assinaturas na qualidade da aproximação.

Com base nos resultados apresentados, concluímos que as assinaturas conseguem estimar de maneira satisfatória e com pouca perda de informação a similaridade entre conjuntos e multiconjuntos de subestruturas. Podemos, então, utilizar as assinaturas como uma alternativa eficiente para comparação de grafos.

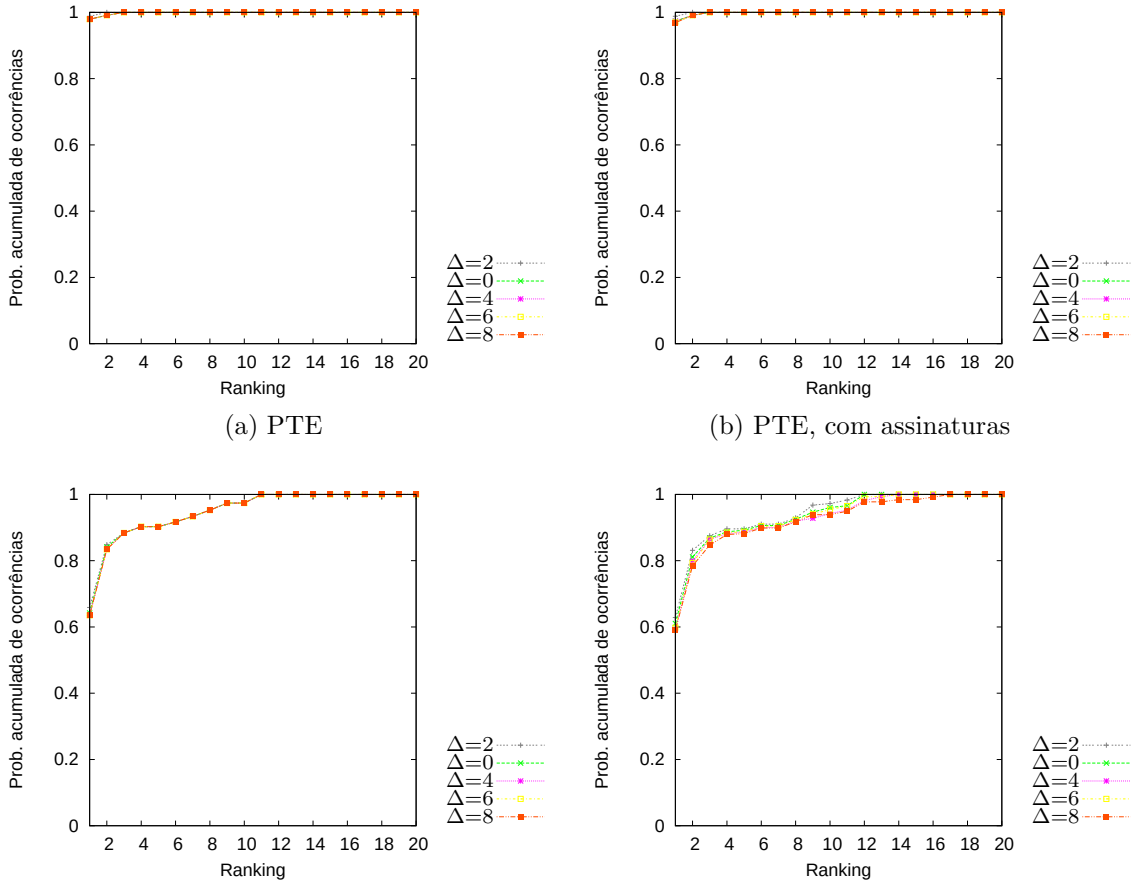
5.3 Aplicações

Nesta seção, aplicamos nosso algoritmo em dois cenários: (1) recuperação de grafos similares e (2) classificação de grafos. Os resultados revelam que a abordagem proposta pode ser utilizada com sucesso nesses dois casos.

5.3.1 Recuperação de grafos similares

Aqui, pretendemos verificar empiricamente a capacidade de nosso método de recuperar grafos similares a uma estrutura dada como consulta. O experimento realizado foi feito da seguinte forma: para cada grafo G da base de dados, criamos uma réplica G' do mesmo, e inserimos no conjunto de grafos. Feito isso, calculamos a distância entre o grafo G e toda a base de dados, incluindo G', e realizamos uma ordenação decrescente entre os valores de similaridades calculados. Logicamente, esperamos que o maior valor de similaridade encontrado seja entre G e G', ou seja, esperamos que G' esteja nas primeiras posições do nosso *ranking*.

A ideia é estudar o impacto de pequenas mudanças na precisão do método e ainda comprovar a recuperação de grafos similares. Com isso, criamos um parâmetro c que indica a porcentagem de alteração de G' em relação ao grafo original. Tal parâmetro assumiu os valores 0% (grafo idêntico ao original), 1%, 3% e 5%, através da remoção de arestas ou substituição de rótulos. Por exemplo, dados um grafo para consulta G e o parâmetro c igual a 3%, temos que G' possui 97% das arestas do grafo original G. No caso da substituição de rótulos, teríamos que G' com $(100-c)$% dos rótulos originais de G, sendo o restante alterado. A escolha das arestas a serem removidas e dos rótulos a serem substituídos são feitos de forma aleatória. Além disso, as configurações avaliadas possuem projeções com tamanho máximo igual a 10 ($dist_{max} = 10$).



(a) PTE (b) PTE, com assinaturas

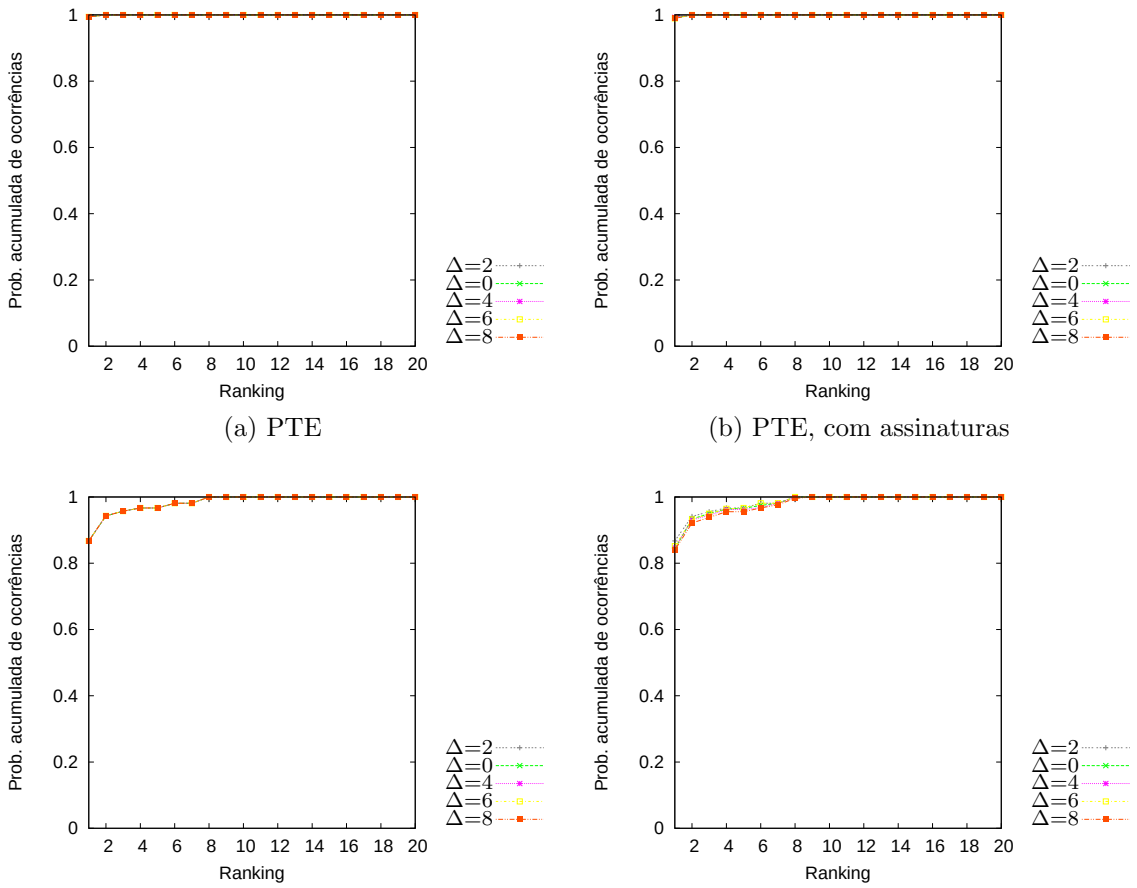
(c) NCI-CA (d) NCI-CA, com assinaturas

Figura 5.5: Recuperação de grafos idênticos com *kernel* binário

A figura 5.5 apresenta as curvas de precisão acumuladas na recuperação de grafos idênticos ($c = 0$) empregando o *kernel* binário nas bases PTE e NCI-CA. Em outras palavras, comparamos o grafo de consulta e todos os outros grafos da base de dados (incluindo o próprio grafo de consulta) e os ordenamos pelo valor de similaridade. A

precisão retornada pelo método reflete o número de grafos que possuem uma similaridade maior ou igual à similaridade do grafo alvo – que é idêntico ao grafo de consulta nesse experimento. Os gráficos 5.5a e 5.5c foram obtidos examinando os conjuntos de pivôs diretamente para o cálculo da função de similaridade, ao contrário das figuras 5.5b e 5.5d que utilizam as assinaturas.

Observe que na base de dados PTE, os resultados são bastante satisfatórios. Temos uma recuperação de 98% dos grafos idênticos (fig. 5.5a) na primeira posição do *ranking*, sendo que todos os grafos foram recuperados dentre as dez primeiras posições. Já na base NCI-CA (fig. 5.5c), a precisão do método aumenta de aproximadamente 60% para 98%, analisando dez posições do *ranking*. As figuras 5.5b e 5.5c reportam testes semelhantes utilizando as bases PTE e NCI-CA, respectivamente. Porém, neste último usamos as assinaturas no cálculo de similaridade. Como podemos perceber, os valores obtidos são bem próximos dos resultados descritos anteriormente, comprovando a efetividade das assinaturas.



(a) PTE (b) PTE, com assinaturas

(c) NCI-CA (d) NCI-CA, com assinaturas

Figura 5.6: Recuperação de grafos idênticos com *kernel* multiconjunto

A figura 5.6 mostra os valores de precisão acumulada obtidos na recuperação de grafos idênticos considerando o *kernel* multiconjunto. Intuitivamente esperamos que explorar a multiplicidade dos pivôs nos conjuntos irá auxiliar a função de comparação a distinguir melhor os grafos. De fato, o *kernel* multiconjunto melhorou a precisão da base NCI-CA em cerca de 25% já na primeira posição do *ranking* (fig. 5.6c), chegando a 87% (fig. 5.6d). Na base PTE (figs. 5.6a e 5.6b), não tivemos grandes alterações dado que a precisão do método já era perto de ótima, mesmo empregando o *kernel* binário.

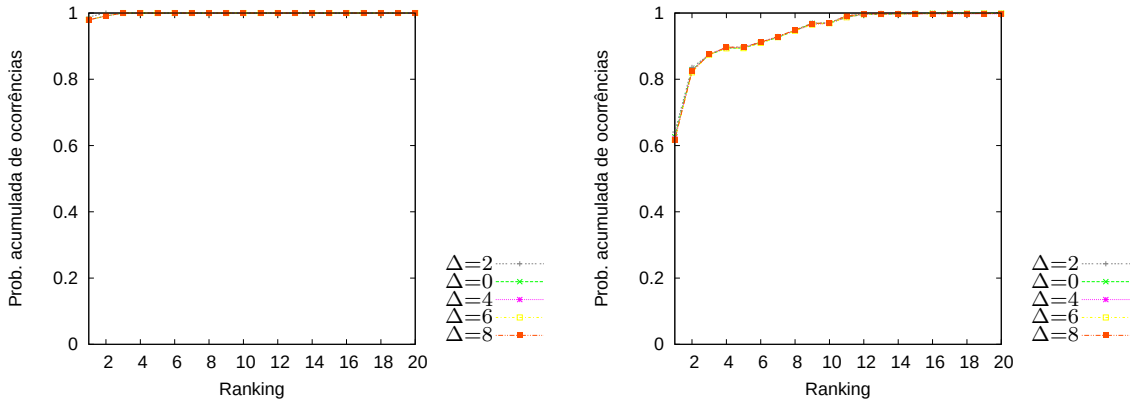
A seguir, estudamos o impacto causado pelas operações de remoção de arestas e substituição de rótulos na recuperação de grafos similares. Visto que as curvas de precisão obtidas com as assinaturas se aproximam das curvas originais ou exatas, apenas as figuras relacionadas ao segundo caso serão discutidas nas próximas seções.

5.3.1.1 Remoção de arestas

Aqui, avaliamos o impacto causado pela remoção de arestas nos grafos na recuperação de grafos similares. O parâmetro c é variado nos valores de 1, 3 e 5, enquanto, Δ é alternado na escala de [0,8].

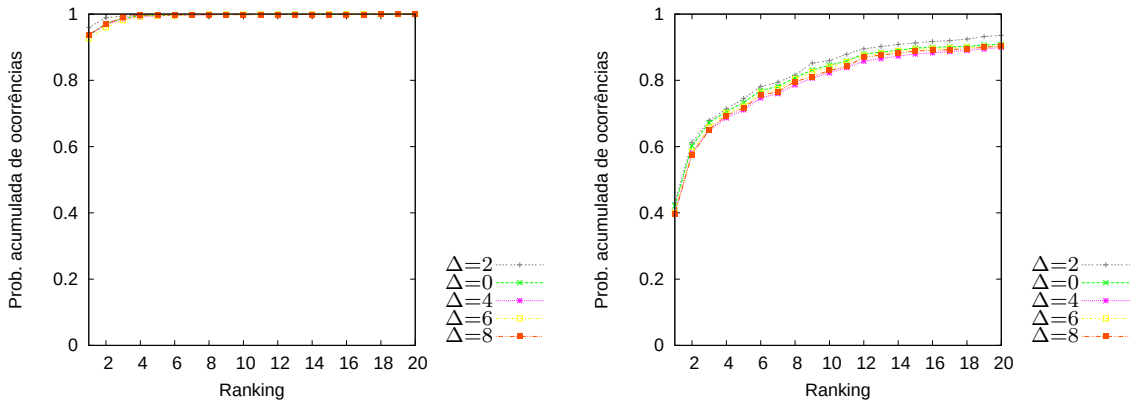
A figura 5.7 mostra os resultados obtidos empregando o *kernel* binário nas bases PTE e NCI-CA. Observe que a remoção de 1% das arestas dos grafos não causou mudanças significativas nas curvas de precisão. Nesta configuração, mantivemos uma recuperação de 98% dos grafos alvo (fig. 5.7a) na primeira posição posição do *ranking*. Todavia, após a remoção de 5% das arestas dos grafos (fig. 5.7e) temos um decréscimo desse valor, chegando a 87%. Em geral, todos os grafos foram recuperados dentre as dez primeiras posições do *ranking* independente dos valores dos parâmetros c e Δ.

Considerando a base de dados NCI-CA, verificamos que a precisão do método com $c = 1$ (fig. 5.7b) foi cerca de 60% e 90% examinando a primeira e as dez primeiras posições do *ranking*, respectivamente. Esses valores são reduzidos de maneira significativa à medida que removemos arestas. Por exemplo, para $c = 5\%$ (fig. 5.7f) temos uma precisão de 30% na primeira posição do *ranking* e, aproximadamente, 70% nas dez primeiras posições. Tal fato pode ser explicado por duas razões: (1) proporcionalmente foi removido um número maior de arestas desses grafos, visto que eles possuem mais arestas (tabela 5.2) e (2) os grafos dessa base de dados compartilham um grande número de rótulos, sendo bastante similares entre si. No primeiro caso, teríamos uma redução dos valores de similaridade entre G e G'. A figura 5.8 apresenta os valores de similaridade para $c = 5$, revelando os altos valores de similaridade (acima de 0,6). Assim, a primeira hipótese possui uma baixa influência nos resultados obtidos visto



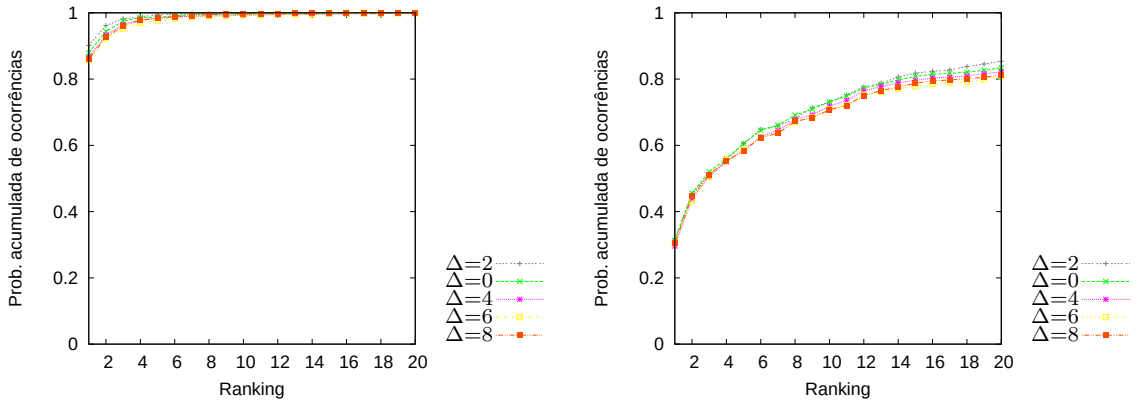
(a) PTE, c=1%

(b) NCI-CA, c=1%



(c) PTE, c=3%

(d) NCI-CA, c=3%

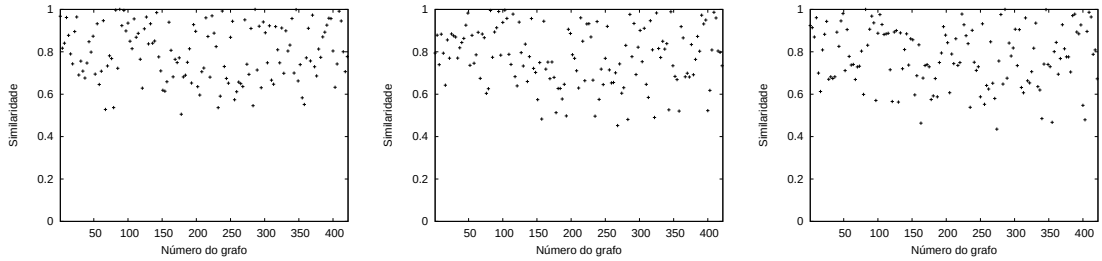


(e) PTE, c=5%

(f) NCI-CA, c=5%

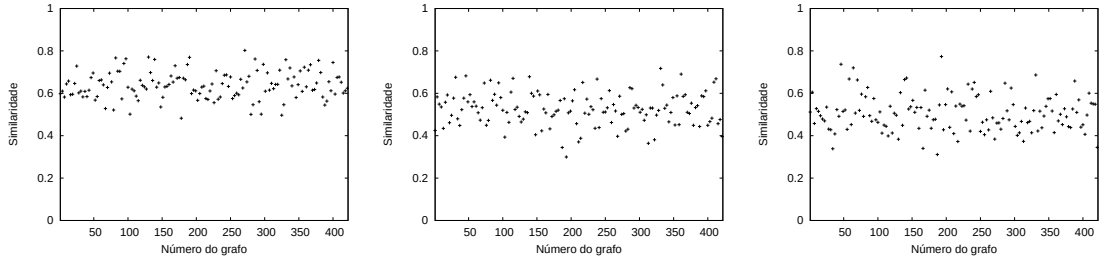
Figura 5.7: Recuperação de grafos similares com *kernel* binário

que os valores retornados pela função de comparação são elevados para ambos casos. Acreditamos que tal comportamento é causado pela segunda hipótese, ou seja, existem vários grafos semelhantes na base de dados e, após a remoção das arestas, essas estruturas passam a ser, de fato, mais similares ao grafo de consulta.



(a) Δ=0 (b) Δ=4 (c) Δ=8

Figura 5.8: Valores de similaridade com *kernel* binário na base NCI-CA



(a) Δ=0 (b) Δ=4 (c) Δ=8

Figura 5.9: Valores de similaridade com *kernel* multiconjunto na base NCI-CA

Os gráficos da figura 5.10 reportam a precisão do método quanto à recuperação de grafos similares utilizando o *kernel* multiconjunto. Para a base PTE verificamos valores de precisão próximos a 100% quando buscamos pelos grafos com remoção de 1% das arestas (fig. 5.10a) já na primeira posição do *ranking* de grafos. Esse valor chega a, aproximadamente, 78% e 65% quando removemos 5% das arestas dos grafos a serem recuperados ao observarmos o primeiro grafo retornado pelo método (fig. 5.10e). Por outro lado, temos uma precisão entre 84% e 98% ao verificarmos as dez primeiras posições no *ranking* nesse mesmo experimento.

Novamente, na base de dados NCI-CA tivemos uma redução significativa da precisão ao mudarmos os parâmetros c. Obtivemos uma precisão de 100% para as primeiras dez posições para $c = 0$, enquanto para $c = 5$ a eficácia do sistema varia entre 23% e 58% (figuras 5.10b e 5.10f, respectivamente). Na base NCI-CA verificamos uma diferença relevante nos valores de precisão ao investigarmos o parâmetros Δ. Por exemplo, na 5.10f a diferença entre a curvas de precisão obtidas por $\Delta = 0$ e $\Delta = 8$ chega a 36%. Observou-se que os valores de similaridade retornados pela função de comparação decrescem à medida que aumentamos o parâmetro Δ (fig. 5.9). Isso pode ser explicado pelo número de relacionamentos diretos e indiretos perdidos ao removermos arestas de um grafo. Em outras palavras, o número de relacionamentos indiretos é maior para

valores elevados de Δ (para $\Delta = 0$ temos apenas relacionamentos diretos). Logo, a remoção de arestas dos grafos faz com que essas configurações tenham uma perda significativa de pivôs e, consequentemente, apresentem um valor de similaridade menor. Note que esse impacto não é observado pelo *kernel* binário, pois ele não considera a multiplicidade ou o número de ocorrências dos pivôs nos conjuntos.

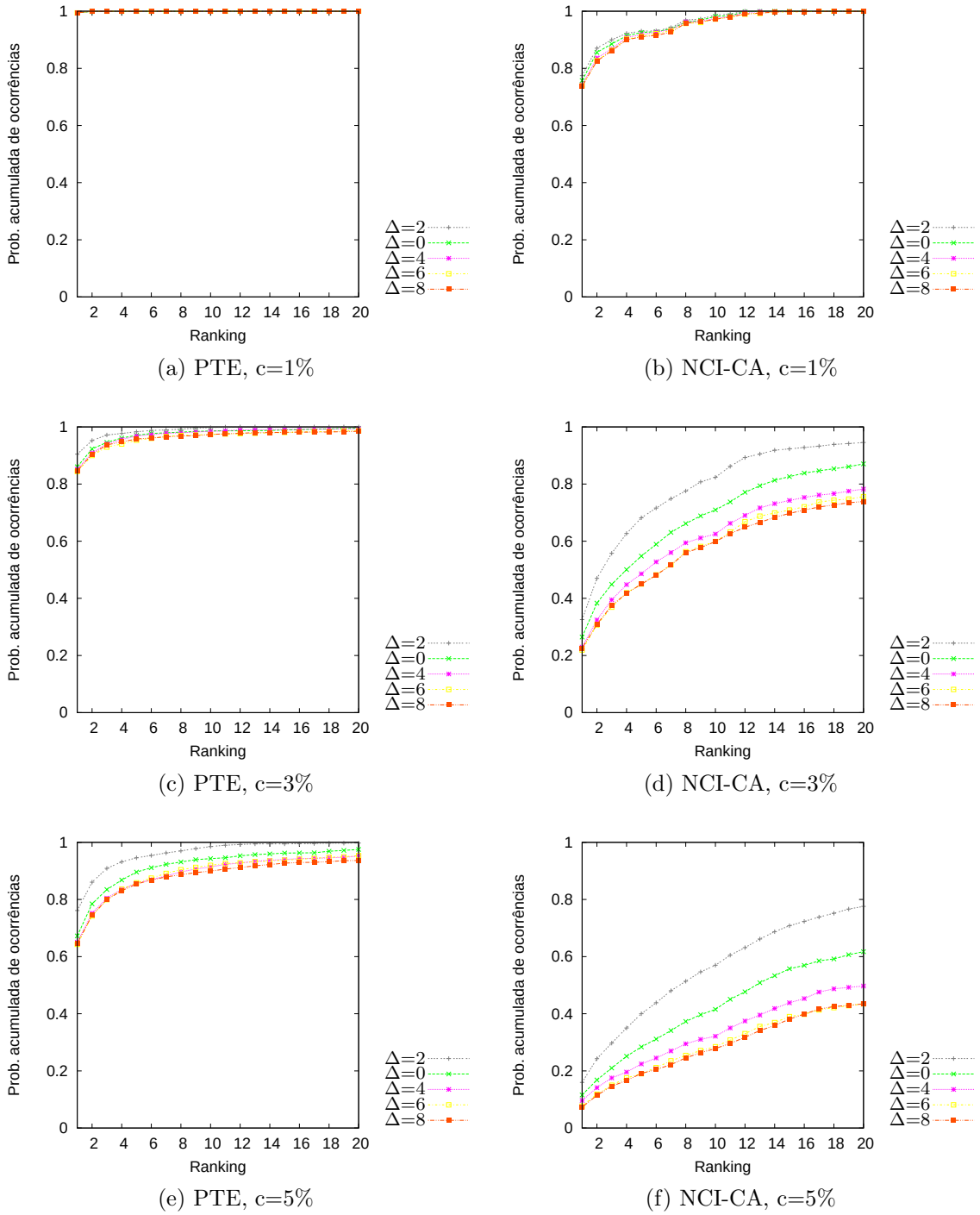
5.3.1.2 Substituição de rótulos

Nesta seção, avaliamos o impacto das operações de substituição de rótulos na recuperação de grafos similares. Ao contrário da remoção de arestas (que apenas diminui o número de caminhos aproximados extraídos), a substituição de rótulos altera os pivôs extraídos dos grafos mantendo o mesmo número de subestruturas.

As figuras 5.11 e 5.12 reportam os resultados de precisão obtidos nas bases PTE e NCI-CA. Como no caso anterior, a primeira figura emprega o *kernel* binário e a segunda o *kernel* multiconjunto.

Considerando os testes realizados com o *kernel* binário, verificamos uma precisão acima de 99% nas primeiras dez posições do *ranking* na base PTE, independentemente dos valores do parâmetro c. Uma diferença de aproximadamente 19% é encontrada entre os valores $c = 1$ e $c = 5$ (figs. 5.11a e 5.11e) avaliando apenas a primeira posição do *ranking*. Na base de dados NCI-CA, a variação do parâmetro c impacta significativamente na precisão do método. Note que o valor de precisão diminui de 87% para pouco mais de 40% considerando as dez primeiras posições do *ranking* de busca nas figuras 5.11b e 5.11f. A diferença de comportamento acentuada entre as bases de dados Mutag e NCI-CA pode ser explicada pelo número de rótulos existentes nos dois conjuntos juntamente com o fato de que o *kernel* binário não considera a multiplicidade dos pivôs. A Mutag possui um número de rótulos cerca de noves vezes maior do que a NCI-CA. Assim, a porcentagem de pivôs únicos alterados nos conjuntos nessa última (sem considerar a multiplicidade) é maior do que na base Mutag. Note que na remoção de arestas isso não aconteceu, pois o número de pivôs únicos alterados é menor.

Na figura 5.12, onde é executado é *kernel* multiconjunto, os resultados foram diferentes. Observe que, em geral, considerar a multiplicidade dos pivôs nos conjuntos é mais eficaz do que verificar a presença ou não dos mesmos. Na base de dados PTE, o valor de precisão foi aumentado em cerca de 3% e 11% em relação aos resultados obtidos *kernel* binário com o parâmetro $c = 3$ e $c = 5$ (figs. 5.12c e 5.12e), respectivamente. Para todos os valores de c avaliados, atingimos 100% de precisão observando as dez primeiras posições do *ranking* de similaridades na base Mutag. Já na base NCI-CA, os ganhos foram ainda mais expressivos – novamente, considerando como *baseline* o



(a) PTE, c=1%

(b) NCI-CA, c=1%

(c) PTE, c=3%

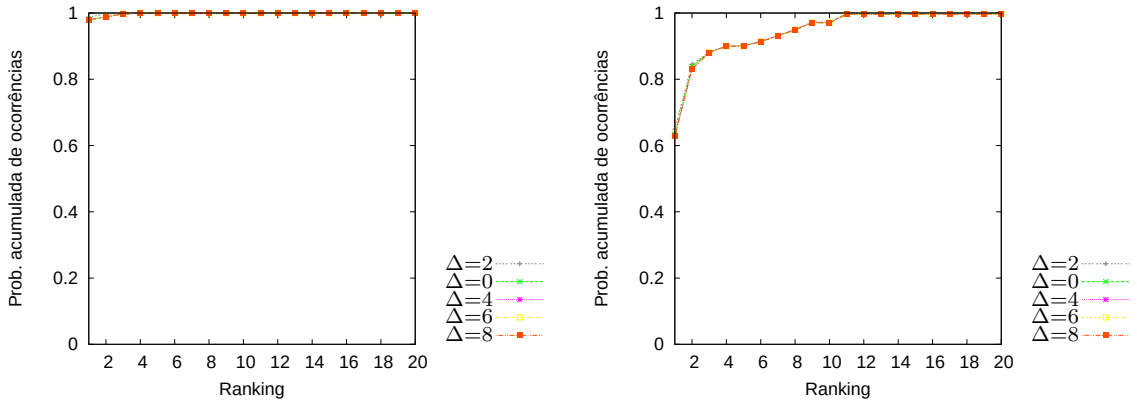
(d) NCI-CA, c=3%

(e) PTE, c=5%

(f) NCI-CA, c=5%

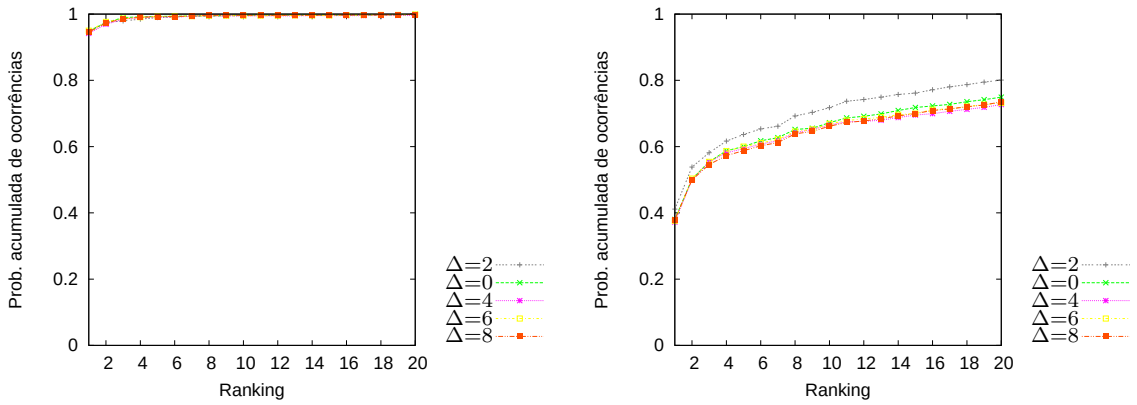
Figura 5.10: Recuperação de grafos similares com *kernel* multiconjunto

kernel binário. Com $c = 1$ (fig. 5.12b), recuperamos cerca de 88% já na primeira posição do *ranking*, aproximadamente 30% superior ao *kernel* binário. Esse ganho chega a quase 40% quando substituímos 5% dos rótulos do grafo, examinando as dez primeiras posições do *ranking* de similaridades. Comparando esses resultados com



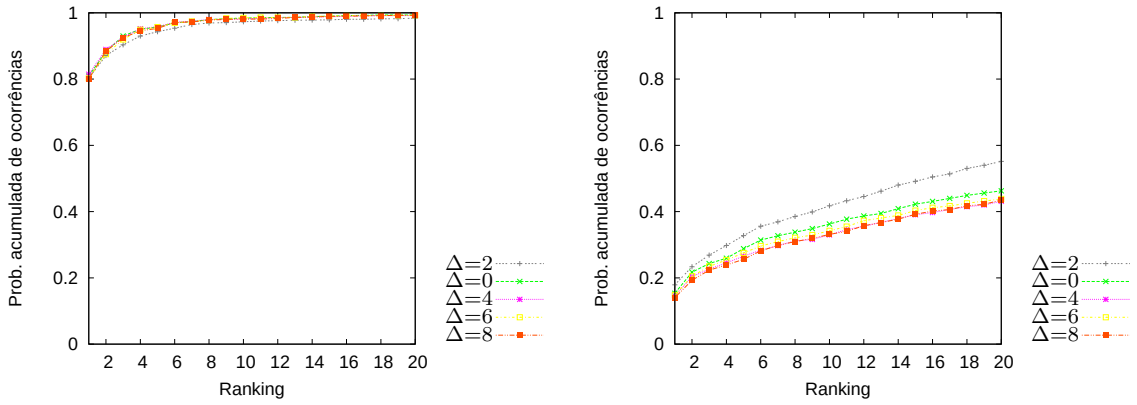
(a) PTE, c=1%

(b) NCI-CA, c=1%



(c) PTE, c=3%

(d) NCI-CA, c=3%

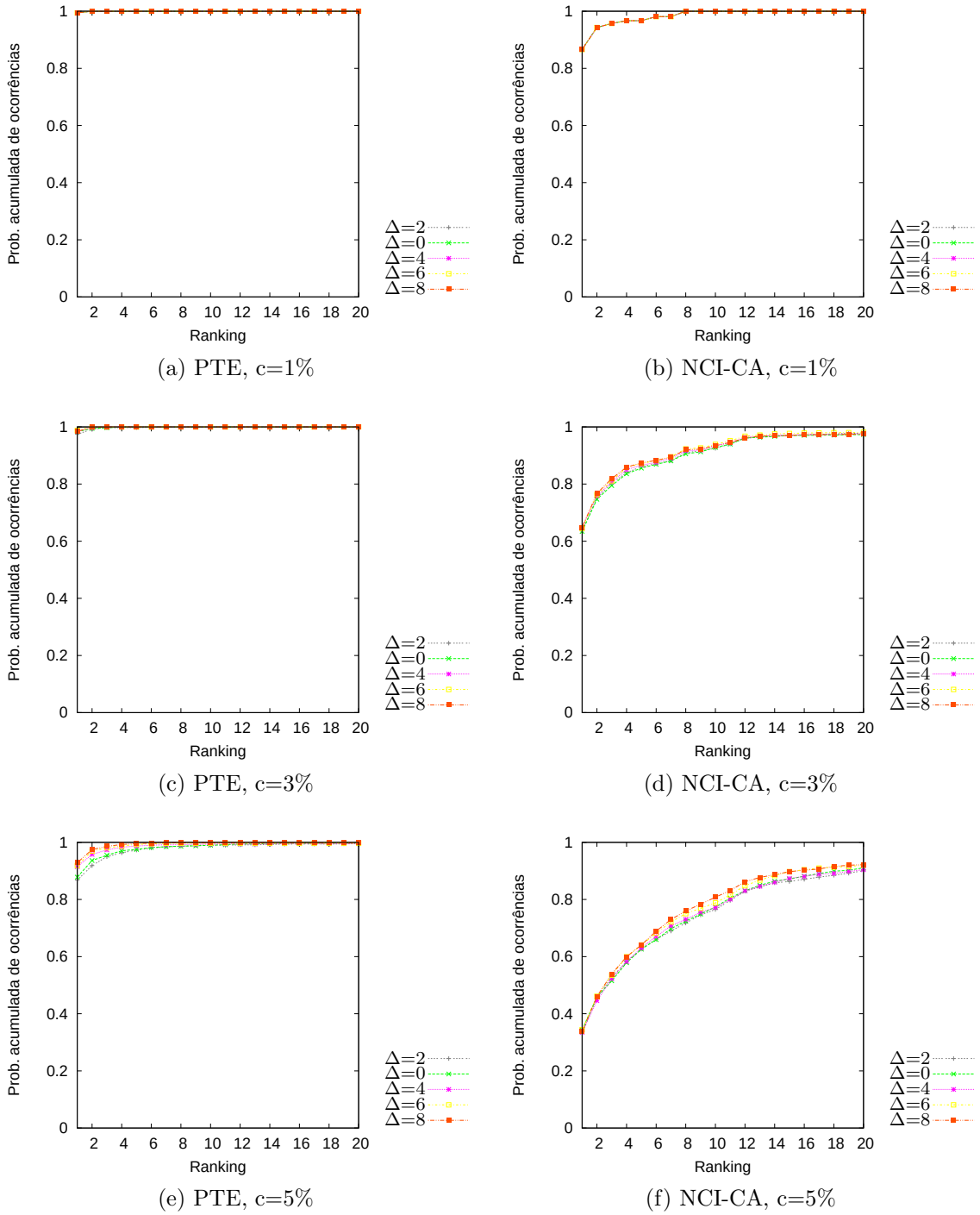


(e) PTE, c=5%

(f) NCI-CA, c=5%

Figura 5.11: Recuperação de grafos similares com *kernel* binário

aqueles obtidos na remoção de arestas, podemos concluir que, de fato, a substituição de rótulos altera um número menor de pivôs (considerando a multiplicidade). Por outro lado, o caso contrário surge quando o número de ocorrências das subestruturas é ignorado.



(a) PTE, c=1%

(b) NCI-CA, c=1%

(c) PTE, c=3%

(d) NCI-CA, c=3%

(e) PTE, c=5%

(f) NCI-CA, c=5%

Figura 5.12: Recuperação de grafos similares com *kernel* multiconjunto

5.3.2 Classificação de grafos

Uma das motivações mais importantes para o desenvolvimento de *kernels* voltados para dados estruturados é a classificação de grafos. Como já mencionamos, diversas bases de

kernel/método	Δ	Mutag	CM	CF	RM	RF
Kernel binário	0	81.91	65.17	**65.32**	**63.37**	**68.09**
	1	83.51	**67.26**	**65.32**	61.62	66.95
	2	84.04	**67.85**	**65.33**	**62.79**	**68.37**
	3	82.97	66.36	64.75	61.91	66.95
	4	83.51	66.07	64.46	63.08	67.23
	5	83.51	65.77	63.89	62.20	67.23
	6	83.51	65.47	63.89	62.50	67.23
	7	83.51	65.77	64.18	62.50	67.23
	8	83.51	65.77	64.18	62.50	67.23
Kernel multiconjunto	0	84.57	62.50	63.03	56.10	65.81
	1	83.51	63.09	62.75	57.26	65.52
	2	87.76	61.90	64.12	59.53	66.38
	3	87.23	62.20	63.03	58.25	65.52
	4	**88.29**	62.50	63.03	59.36	**68.37**
	5	**88.29**	62.20	62.75	58.54	65.52
	6	**88.29**	61.90	64.46	56.68	65.52
	7	**88.29**	62.50	63.03	59.65	65.52
	8	**88.82**	61.90	62.17	60.46	65.52

Tabela 5.4: Classificação automática de grafos

dados das áreas de biologia e química necessitam ser organizadas em categorias definidas por especialistas. Nosso objetivo, aqui, é empregar nosso *kernel* para classificação automática de grafos. A métrica usada para avaliação e comparação das abordagens foi a acurácia. Tal métrica consiste numa simples divisão entre o número de grafos classificados corretamente divido pelo número total de grafos. Note que quando temos apenas duas classes (como é o caso de nossas bases de dados) a acurácia é equivalente à métrica microF1. Ademais, a métrica macroF1 não foi avaliada visto que as classes são balanceadas.

Em nossa avaliação, utilizamos as bases Mutag e PTC's que são *benchmarks* bem estabelecidos na literatura. O algoritmo de classificação utilizado foi o SVM linear (disponível na biblioteca *libsvm* [Chang & Lin, 2001]), descrito no capítulo 3. Os resultados foram obtidos através da técnica de validação cruzada (*cross-validation*) de tamanho dez, ou seja, a base de dados é divida em dez partes, sendo que nove delas são usadas para treinamento e uma para teste. As tabelas 5.4 e 5.5 mostram os valores de acurácia do classificador utilizando as similaridades originais (matriz *kernel* criada diretamente dos conjuntos de pivôs) e os valores estimados (matriz *kernel* gerada pelas assinaturas). Note que aplicamos o teste-t e nossos resultados apresentam uma confiança de 90%: os valores de acurácia estatisticamente superiores estão em destaque.

Considerando a tabela 5.4 podemos notar que: (1) geralmente, as bases PTC's

kernel/método	Δ	Mutag	CM	CF	RM	RF
Kernel binário	0	82.76	66.24	**65,21**	62,09	**67,69**
	1	83,08	**66,66**	65,04	62,55	66,83
	2	82,97	**67,02**	65,04	63,25	67,17
	3	82,23	66,07	64,52	61,27	**67,86**
	4	82,76	66,13	**65,10**	61,68	66,78
	5	83,08	65,65	64,41	62,50	67,00
	6	82,87	65,77	64,46	62,44	67,17
	7	82,76	66,07	64,46	61,68	66,60
	8	82,87	65,71	64,58	61,91	66,83
Kernel multiconjunto	0	89,14	65,71	**65,50**	63,66	66,66
	1	89,36	65,95	**65,44**	**65,00**	66,49
	2	89,14	65,23	**65,27**	64,01	67,00
	3	89,14	64,46	64,58	62,55	67,35
	4	89,46	63,63	**65,21**	63,43	66,95
	5	89,89	63,75	**65,50**	62,15	67,00
	6	89,99	64,34	**65,15**	63,31	**67,63**
	7	**90,85**	64,28	64,87	62,79	**67,40**
	8	**90,21**	64,40	64,75	62,38	66,95

Tabela 5.5: Classificação automática de grafos com assinatura

são melhor classificadas usando o *kernel* binário, (2) o *kernel* multiconjunto obtém valores de acurácia satisfatórios na base de dados Mutag e (3) caminhos aproximados, parâmetro $d > 0$, geralmente apresentou melhores resultados de acurácia. No primeiro caso, a existência ou não de uma dada subestrutura no grafo é mais relevante do que o número de vezes que ela ocorreu. Além disso, nas bases PTC's podemos observar que pequenos valores de Δ geram maiores acurácias. Isso pode ser explicado pelo número de pivôs gerados – à medida que aumentamos o valor de Δ criamos um número maior de pivôs. Todavia, valores elevados de Δ podem, também, aumentar a similaridade entre os grafos que possuem o mesmo conjunto de rótulos, independentemente das ligações entre os vértices. Assim, como os rótulos nas bases PTC's estão presentes nas duas classes, o método de classificação perde acurácia. Já no caso da base Mutag, percebemos que a multiplicidade dos pivôs nos conjuntos é uma informação relevante para classificação. Note, ainda, que nessa base maiores valores de Δ geram melhores soluções indicando que os relacionamentos indiretos contribuem significativamente para o algoritmo de classificação.

Executamos, também, um experimento de forma semelhante ao anterior (validação cruzada de tamanho 10 e nas mesmas bases de dados) porém, aplicando as assinaturas para o cálculo de similaridade. Os valores de acurácia podem ser vistos na tabela 5.5. Note que, o uso das assinaturas aumenta os valores de acurácia com

relação ao resultado anterior em dois casos. Por exemplo, na base Mutag, o valor da acurácia passou de 88,82% para 90,85%. Um ganho de aproximadamente 2% também é observado na base PTC-RM. De fato, nos demais casos, as assinaturas apresentaram resultados bem próximos quando comparados com o experimento não aproximado. Acreditamos que essa melhora relativa que as assinaturas proporcionaram no valor de acurácia, considerando a base Mutag, está ligada a redução de dimensionalidade dos atributos (ou *features*) dos grafos. Isto auxiliou o classificador na criação do modelo de categorização, ou seja, o SVM foi capaz de separar melhor as classes no espaço formado pelas assinaturas.

5.3.2.1 Comparação com outros métodos

Nesta seção, comparamos nosso método com os outros disponíveis na literatura. Para isso, utilizamos os resultados reportados na literatura, onde foram executados experimentos sobre as mesmas bases de dados [Kashima et al., 2003; Ralaivola et al., 2005; Borgwardt & Kriegel, 2005].

A comparação entre os métodos de classificação foi dividida em duas etapas. Primeiro, utilizamos os melhores resultados reportados no trabalho que propõe o *kernel* baseado em caminhos mínimos [Borgwardt & Kriegel, 2005]. Nesse artigo científico são avaliadas duas funções *kernel*, caminhos mínimos e produto de grafos, nas bases Mutag e PTC-RM. É importante ressaltar que a comparação é feita através do método de validação cruzada de tamanho dez. O classificador utilizado foi o SVM linear disponível na biblioteca *libsvm* [Chang & Lin, 2001]. A tabela 5.6 contrasta os melhores resultados[4] obtidos pelo uso de nossa estratégia com o *kernel* baseado em caminhos mínimos e o *kernel* baseado em produto de grafos.

Os ganhos são consistentes nas duas bases avaliadas. Na base PTC-RM temos um aumento de aproximadamente 5% em termos de acurácia. Já na base Mutag temos um ganho 7% em relação ao kernel baseado em caminhos mínimos.

A segunda parte de nosso estudo de comparação utiliza os melhores resultados obtidos nos trabalhos relacionados ao *kernel* baseado em margens e *kernel* baseado em caminhos simples [Kashima et al., 2003; Ralaivola et al., 2005]. Os resultados apresentados nesses trabalhos foram gerados a partir de uma classificação simples (*leave-one-out*[5]), isto é, a cada momento um grafo é usado para teste e os demais para treino. O algoritmo utilizado para treino foi o classificador baseado em votação *Percep-*

[4]Note que não consideramos $\Delta = 0$, pois essa configuração é equivalente ao uso de caminhos simples.

[5]Esse método é equivalente a uma validação cruzada de tamanho N, onde N é o número total de grafos.

Tabela 5.6: Comparação entre os algoritmos de classificação

kernel/método	Mutag	PTC-RM
KCA[a]	88,8	63,0
KCA com assinaturas[b]	**90,8**	**65,0**
KPG[c]	78,9	59,8
KCM[d]	83,9	59,0

[a] Kernel baseado em caminhos aproximados
[b] Kernel baseado em caminhos aproximados com assinaturas
[c] Kernel baseado em produto de grafos
[d] Kernel baseado em caminhos mínimos

kernel/método	Δ	Mutag	CM	CF	RM	RF
Kernel binário	0	81,38	64,58	65,04	**64,53**	**68,37**
	1	83,51	**66,66**	63,89	63,37	**68,37**
	2	83,51	**66,66**	63,89	62,79	67,52
	3	82,97	65,77	63,89	60,75	66,66
	4	83,51	66,36	63,61	61,04	66,66
	5	83,51	65,47	63,89	60,46	66,66
	6	83,51	64,88	63,61	61,33	66,38
	7	83,51	64,88	63,61	60,75	66,38
	8	83,51	64,88	63,61	60,75	66,38
Kernel multiconjunto	0	84,57	61,60	62,46	56,10	66,09
	1	84,57	63,98	62,75	55,81	65,52
	2	87,76	61,60	**65,95**	55,81	67,97
	3	**88,82**	63,09	61,89	59,30	66,89
	4	**88,82**	62,79	63,83	58,72	65,52
	5	**88,82**	61,90	62,17	60,75	65,52
	6	88,29	63,45	63,61	60,17	65,52
	7	88,29	63,39	63,32	61,91	65,52
	8	87,76	61,60	63,03	59,18	65,52

Tabela 5.7: Classificação automática de grafos

tron [Freund & Schapire, 1999]. Tal classificador é constituído de vários classificadores lineares chamados de *perceptrons*[6]. Assim, os vários *perceptrons* votam e decidem a qual classe um determinado elemento pertence. O processo de treinamento realizado por esses algoritmos é bastante eficiente e os resultados de classificação são comparáveis ou até superiores com relação dos métodos SVM [Freund & Schapire, 1999].

[6]Note que as margens de separação utilizadas nesse métodos não são otimizadas, diferentemente dos classificadores SVM's.

kernel/método	Δ	Mutag	CM	CF	RM	RF
Kernel binário	0	82,34	65,77	64,06	63,02	**68,20**
	1	82,55	66,66	64,29	63,48	67,17
	2	83,08	**67,55**	64,12	61,91	67,86
	3	82,87	66,90	63,95	60,52	66,89
	4	83,51	66,25	64,01	61,04	67,12
	5	82,97	67,20	64,18	60,34	67,12
	6	82,76	66,01	64,06	62,50	67,06
	7	83,19	66,66	63,89	60,98	66,72
	8	82,65	65,77	63,78	61,51	67,46
Kernel multiconjunto	0	88,93	64,88	**65,50**	64,36	67,92
	1	89,14	65,59	**65,50**	64,70	67,63
	2	88,51	65,83	64,92	64,59	67,69
	3	89,14	65,05	**65,78**	64,70	67,23
	4	89,78	64,34	64,52	63,08	67,35
	5	89,57	65,35	**65,78**	62,44	67,74
	6	89,57	65,11	64,81	**65,34**	67,92
	7	**89,99**	64,58	65,15	64,65	67,86
	8	89,14	65,00	65,04	**65,52**	67,57

Tabela 5.8: Classificação automática de grafos com assinatura

As tabelas 5.7 e 5.8 mostram os resultados obtidos por nosso algoritmo através do método de avaliação *leave-one-out*. Os maiores valores foram selecionados para uma comparação e apresentados na tabela 5.9. Os valores de acurácia das demais abordagens foram obtidos dos artigos científicos encontrados na literatura [Kashima et al., 2003; Ralaivola et al., 2005]. Novamente, nossa abordagem conseguiu ganhos em relação aos demais, exceto para a base de dados PTC-RM. Nosso método apresenta uma acurácia superior em quatro bases, com ganhos de 1% até 3%. Por outro lado, o *kernel* proposto apresenta uma acurácia de 0,2% inferior em relação ao *kernel* baseado em caminhos simples. Considerando que o problema de classificação de grafos é um problema desafiador (com ganhos relativamente baixos), os resultados apresentados credenciam o método *kernel* proposto como uma contribuição significativa nas áreas de mineração de dados e aprendizado de máquina.

5.4 Eficiência

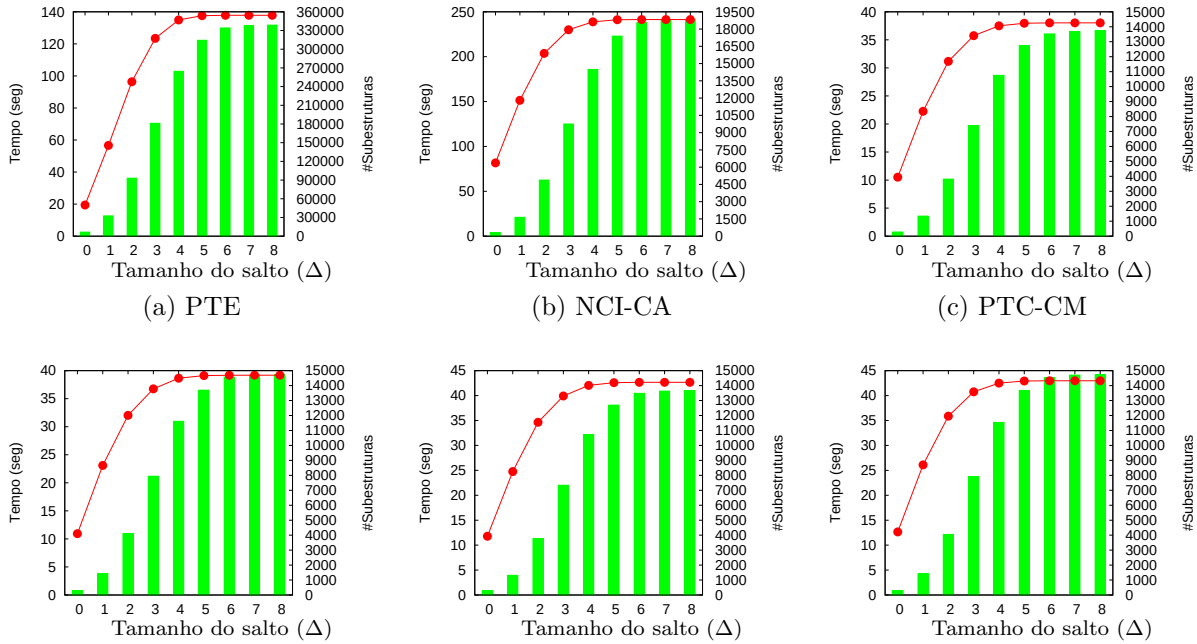
Como já foi mencionado, o tempo de execução do algoritmo para extração de pivôs é determinante para o desempenho do arcabouço de comparação de grafos. Nesta seção, medimos o tempo gasto dessa fase, incluindo, também, a geração das assinaturas.

Tabela 5.9: Comparação entre os algoritmos de classificação

kernel/método	Mutag	CM	CF	RM	RF
KCA[a]	88,8	66,6	**65,9**	63,3	**68,3**
KCA assinatura[b]	**89,9**	**67,5**	65,7	65,5	68,2
KPD[c]	89,1	61,0	61,0	62,8	66,7
KM[d]	85,1	64,3	63,4	58,4	66,1
KCS[e]	87,8	66,4	64,5	**65,7**	66,9

[a] Kernel baseado em caminhos aproximados
[b] Kernel baseado em caminhos aproximados com assinaturas
[c] Kernel baseado em padrões
[d] Kernel baseado margens
[e] Kernel baseado em caminhos simples

Obviamente, a computação necessária para extrair os conjuntos de pivôs dos grafos domina o tempo de execução para geração das assinaturas – o tempo gasto para extração dos pivôs seguido da geração da assinatura de um grafo possui uma complexidade da ordem de $O(k(2N)^{dist_{max}})$, considerando um grafo de tamanho N, com uma assinatura de tamanho k. Os experimentos mostrados na figura 5.13 mostram o desempenho desse processo nas bases de dados estudadas até aqui. Note que os valores dos parâmetros utilizados nesse experimento foram $dist_{max} = 10$ e $k = 128$.



(a) PTE (b) NCI-CA (c) PTC-CM

(d) PTC-CF (e) PTC-RM (f) PTC-RF

Figura 5.13: Tempo de execução para extração dos pivôs e geração das assinaturas

Os gráficos apresentados em 5.13 contrastam o tempo de processamento dos grafos com o número de pivôs extraídos dos mesmos. Existe uma dependência clara entre os fatores tempo e número de subestruturas em todas as bases. Podemos notar que, em geral, os resultados reportados pelas bases PTC's são bem próximos, o que é esperado visto que elas pertencem ao mesmo cenário. Essas bases geram em torno de quinze mil subestruturas quando assumimos $\Delta = 8$. Observe que os padrões de crescimento do número de pivôs extraídos são bastante parecidos nas bases avaliadas, mantendo-se aproximadamente constante para valores de Δ acima de 6. Isso acontece pois a profundidade máxima ($dist_{max}$) dos grafos avaliados é próxima de 6, ou seja, o tamanho dos pivôs é limitado. A base de dados PTE possui um número bem superior de subestruturas, sendo cerca de vinte vezes maior que a quantidade de pivôs das bases PTC's. Tal diferença é explicada pelo número de rótulos desses domínios.

Base	Subestruturas (Kb)	Assinaturas (Kb)	Tx. Compressão
MCF-7	6660.00	2240.80	2.97
MOLT-4	8724.00	3075.20	2.83
OVCAR-8	6044.00	2030.40	2.97

Tabela 5.10: Análise de compressão dos dados

Por fim, realizamos experimentos para medir a taxa de compressão de nosso método, isto é, comparamos o espaço de memória despendido (em *kbytes*) para armazenar os conjuntos de subestruturas extraídos dos grafos com espaço gasto pelas assinaturas. O experimento foi executado com os parâmetros $dist_{max} = 10$, $\Delta = 0$ e $k = 16$. A tabela 5.10 relata os espaços de memória necessários para armazenar as bases MCF-7, MOLT-4, e OVCAR-8 e suas respectivas taxas de compressão[7] – próxima de 3, em todas as bases avaliadas. Em outras palavras, conseguimos economizar cerca de 75% de espaço de memória utilizando as assinaturas. Note que, a taxa de compressão pode ser ainda maior se os valores dos parâmetros $dist_{max}$ e/ou Δ fossem incrementados. Isso porque, nesse caso, o número de subestruturas extraídas dos grafos pode ser consideravelmente maior. Uma outra forma de elevarmos a taxa de compressão é, obviamente, reduzir o valor do parâmetro k (tamanho das assinaturas).

5.5 Discussão

Neste capítulo, avaliamos como os parâmetros dos algoritmos para extração dos pivôs dos grafos e para geração das assinaturas impactam na eficácia do cálculo de similari-

[7]Note que a taxa de compressão é dada pela razão entre os espaços de memória gastos.

dade das estruturas. Ao todo, o método foi testado em cinco bases de dados, em dois contextos aplicação: (1) recuperação de grafos similares e (2) classificação de grafos.

A eficácia dos caminhos aproximados foi demonstrada através dos resultados de precisão na recuperação de grafos similares e dos valores de acurácia alcançados durante os experimentos de classificação de grafos. No primeiro, investigamos o comportamento do nosso método de comparação perante pequenas alterações na estrutura dos grafos (remoção de arestas e substituição de rótulos) e obtivemos altos valores de precisão (usualmente, acima de 90%) examinando as primeiras posições do *ranking* de busca. Na classificação de grafos, os resultados revelam que o método de *kernel* proposto apresenta valores de acurácia superiores (em quatro das cinco bases avaliadas) em relação aos demais algoritmos de classificação disponíveis na literatura.

Considerando os desafios da comparação de grafos citados nos capítulos anteriores – eficácia e eficiência –, verificamos que a abordagem proposta apresenta contribuições relevantes para as áreas de mineração de dados e aprendizado de máquinas. Quanto à eficiência, nossa estratégia explora as assinaturas dos grafos, baseando na sumarização dos dados. Foi mostrado nesse capítulo que as assinaturas, apesar de pequenas e de tamanho fixo, conseguem representar adequadamente os conjuntos de pivôs dos grafos. As assinaturas podem ser, então, armazenadas em memória principal para estimar as similaridades entre grafos com alto desempenho.

Na próxima seção, apresentamos nossas conclusões e os trabalhos futuros.

Capítulo 6

Conclusão e trabalhos futuros

Impulsionado pelos avanços tecnológicos nas últimas duas décadas, a análise de dados (semi-)estruturados (grafos e árvores) se tornou um importante tópico da Ciência da Computação. O grande volume de grafos disponível estabeleceu uma necessidade clara de organização, consultas e análises de forma eficiente sobre esses dados, que estão sob contínuo crescimento. Nesse contexto, uma operação fundamental é o *cálculo de similaridade* ou *comparação* entre entidades. Em nosso caso, dados dois grafos quaisquer usamos uma função de comparação para mensurar a similaridade (ou a diferença) estrutural entre eles. Todavia, por serem estruturas bastante flexíveis, a comparação de grafos é, de fato, um desafio de pesquisa em diversas áreas como mineração de dados e aprendizado de máquina.

Neste trabalho, desenvolvemos um arcabouço para comparação de grafos. Formado por quatro etapas, o arcabouço recebe como entrada as estruturas a serem avaliadas e extrai os conjuntos de subestruturas dos respectivos grafos. Posteriormente, os conjuntos de subestruturas são transformados em assinaturas pequenas e de tamanho fixo através de técnicas de *hashing*. Por fim, as assinaturas são utilizadas para comparar os grafos de forma eficiente. Destacamos como principais contribuições dessa dissertação:

- *Caminhos aproximados:* propomos uma nova subestrutura de grafos como alternativa às encontradas na literatura (grafos, caminhos simples etc). Os caminhos aproximados permitem saltos (*gaps*) entre os vértices da cadeia, ou seja, os vértices que constituem o caminho não precisam ser, necessariamente, vizinhos em si. Isso nos proporciona uma importante vantagem em relação as outras propostas: flexibilidade. A função de comparação, também conhecida como função *kernel*, proposta é baseada na similaridade de *Jaccard* entre conjuntos de caminhos apro-

ximados.

- *Técnicas* hashing *para sumarização de grafos:* Adotamos a técnica conhecida como *Min-hash* para a sumarização dos conjuntos de subestruturas (caminhos aproximados). *Min-hash* vem sendo aplicadas com êxito em diversos cenários como, por exemplo, mineração de padrões frequentes. As assinaturas geradas pela técnica *Min-hash* são usadas para estimar a similaridade de *Jaccard* entre os seus respectivos conjuntos, com qualidade estatisticamente garantida. Por serem pequenas e de tamanho fixo, as assinaturas podem ser colocadas em memória principal, permitindo uma análise eficiente dos dados.

Nós avaliamos o arcabouço proposto em cenários reais como, por exemplo, moléculas provindas de análises químicas. Os principais resultados são:

- Para mostrar que as assinaturas conseguem estimar com precisão a similaridade entre conjuntos, medimos a correlação de *Pearson* entre os valores de similaridade exatos e aproximados. Os resultados revelam que existe uma correlação alta e positiva entre os dados, acima de 0,6. Essa dependência entre os dados se torna ainda mais clara através dos gráficos de dispersão.

- Empregamos nossa estratégia para consultas e recuperação de grafos similares. Em geral, mostramos que nossa abordagem retorna o grafo alvo nas primeiras posições do *ranking* de similaridade. Considerando o grafo de consulta e a estrutura alvo (a ser retornada pela consulta) iguais, o método proposto tem uma precisão acima de 90% já na primeira posição do *ranking* de busca.

- Na classificação de grafos, o método *kernel* proposto alcançou resultados superiores às técnicas encontradas na literatura. Os ganhos variam entre 1% até 7%, em quatro das cinco base de dados testadas. Assim, os resultados apresentados credenciam o método *kernel* proposto como uma abordagem promissora nas áreas de mineração de dados e aprendizado de máquina.

Obviamente, existem vários trabalhos possíveis a serem realizados. Abaixo, listamos algumas alternativas interessantes para trabalhos futuros:

- Avaliar o algoritmo proposto e os caminhos aproximados em outras aplicações. Por exemplo, podemos empregar a função de similaridade para realizar tarefas de agrupamento (classificação não-supervisionada), detecção de exceções e até balanceamento de carga em sistemas distribuídos.

- Empregar o algoritmo em outros cenários. Redes sociais, Web e diagramas de fluxos de programas são opções interessantes. Podemos, ainda, estender o arcabouço criado para outros tipos de representações baseadas em grafos como grafos direcionados, sem rótulos ou multi-rotulados.

- Desenvolver e avaliar algoritmos paralelos para extração dos caminhos aproximados. Como foi visto, o processo de extração de caminhos aproximados é custoso e dependente de parâmetros. O uso de algoritmos paralelos proporcionaria a obtenção de um número maior de caminhos aproximados com um tempo menor de processamento.

- Avaliar a qualidade de diferentes modelos de assinaturas. Nesse trabalho, utilizamos a técnica *Min-hash* para geração das assinaturas. Contudo, existem diversas técnicas que podem ser utilizadas para essa tarefa. Estudar o impacto dessas abordagens em diferentes aplicações e serviços é uma direção promissora.

Referências Bibliográficas

Agrawal, S.; Chaudhuri, S. & Das, G. (2002). Dbxplorer: Enabling keyword search over relational databases. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, SIGMOD '02, pp. 627--627, New York, NY, USA. ACM.

Aho, A. V.; Sethi, R. & Ullman, J. D. (1996). *Compilers: Principles, Techniques and Tools*. Addison Wesley.

Borgwardt, K. M. & Kriegel, H.-P. (2005). Shortest-path kernels on graphs. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, ICDM '05, pp. 74--81, Washington, DC, USA. IEEE Computer Society.

Boser, B. E.; Guyon, I. M. & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pp. 144--152, New York, NY, USA. ACM.

Brazma, A.; Parkinson, H.; Sarkans, U.; Shojatalab, M.; Vilo, J.; Abeygunawardena, N.; Holloway, E.; Kapushesky, M.; Kemmeren, P.; Lara, G. G.; Oezcimen, A.; Rocca-Serra, P. & Sansone, S.-A. (2003). ArrayExpress – A public repository for microarray gene expression data at the EBI. *Nucleic Acids Research*, 31(1):68--71.

Broder, A. Z.; Charikar, M.; Frieze, A. M. & Mitzenmacher, M. (1998). Min-wise independent permutations (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pp. 327--336, New York, NY, USA. ACM.

Broder, A. Z.; Glassman, S. C.; Manasse, M. S. & Zweig, G. (1997). Syntactic clustering of the web. pp. 1157--1166.

Buehrer, G. & Chellapilla, K. (2008). A scalable pattern mining approach to web graph compression with communities. In *Proceedings of the 2008 International Conference*

on Web Search and Data Mining, WSDM '08, pp. 95--106, New York, NY, USA. ACM.

Bunke, H. (2000). Graph Matching: Theoretical Foundations, Algorithms, and Applications.

Bunke, H. & Allermann, G. (1983). Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245 – 253.

Bunke, H. & Shearer, K. (1998). A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3–4):255 – 259.

Chang, C.-C. & Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pp. 380--388, New York, NY, USA. ACM.

Chung, F. R. K. (1997). *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92)*. American Mathematical Society.

Cohen, E. (1997). Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences*, 55(3):441 – 453.

Cohen, E.; Datar, M.; Fujiwara, S.; Gionis, A.; Indyk, P.; Motwani, R.; Ullman, J. & Yang, C. (2001). Finding interesting associations without support pruning. *Knowledge and Data Engineering, IEEE Transactions on*, 13(1):64–78.

Cortes, C. & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

Cristianini, N. & Shawe-Taylor, J. (2000). *An introduction to support Vector Machines: and other kernel-based learning methods*. Cambridge University Press, New York, NY, USA.

Debnath, A. K.; Debnath, G.; Shusterman, A. J. & Hansch, C. (1992). A qsar investigation of the role of hydrophobicity in regulating mutagenicity in the ames test: 1. mutagenicity of aromatic and heteroaromatic amines in salmonella typhimurium ta98 and ta100. *Environmental and Molecular Mutagenesis*, 19(1):37--52.

Dipert, R. R. (1997). The mathematical structure of the world: The world as graph. *Journal of Philosophy*, 94(7):329–358.

Egghe, L. & Rousseau, R. (1990). *Introduction to Informetrics : quantitative methods in library, documentation and information science*. Elsevier Science Publishers.

Elmagarmid, A.; Ipeirotis, P. & Verykios, V. (2007). Duplicate record detection: A survey. *Knowledge and Data Engineering, IEEE Transactions on*, 19(1):1–16.

Faloutsos, M.; Faloutsos, P. & Faloutsos, C. (1999). On power-law relationships of the internet topology. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '99, pp. 251--262, New York, NY, USA. ACM.

Freund, Y. & Schapire, R. (1999). Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296.

Gaertner, T.; Flach, P. & Wrobel, S. (2003). On graph kernels: Hardness results and efficient alternatives. In *Proceedings of the 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop*, pp. 129--143. Springer-Verlag.

Garey, M. R. & Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

Gionis, A.; Indyk, P. & Motwani, R. (1999). Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pp. 518--529, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Gower, J. C. & Legendre, P. (1986). Metric and euclidean properties of dissimilarity coefficients. *Journal of Classification*, 3:5–48. 10.1007/BF01896809.

Gross, J. & Yellen, J. (1999). *Graph theory and its applications*. CRC Press, Inc., Boca Raton, FL, USA.

Haussler, D. (1999). Convolution Kernels on Discrete Structures. Technical report.

Helma, C.; King, R. D.; Kramer, S. & Srinivasan, A. (2001). The predictive toxicology challenge 2000-2001. *Bioinformatics*, 17(1):107–108.

Horváth, T.; Gärtner, T. & Wrobel, S. (2004). Cyclic pattern kernels for predictive graph mining. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pp. 158--167, New York, NY, USA. ACM.

Huberman, B. A. (2001). *The Laws of the Web: Patterns in the Ecology of Information*. The MIT Press.

Indyk, P. & Motwani, R. (1998). Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pp. 604--613, New York, NY, USA. ACM.

Ito, T.; Chiba, T.; Ozawa, R.; Yoshida, M.; Hattori, M. & Sakaki, Y. (2001). A comprehensive two-hybrid analysis to explore the yeast protein interactome. *Proceedings of the National Academy of Sciences of the United States of America*, 98(8):4569--4574.

Jeong, H.; Mason, S. P.; Barabási, A. L. & Oltvai, Z. N. (2001). Lethality and centrality in protein networks. *Nature*, 411(6833):41--42.

Kalapala, V.; Sanwalani, V.; Clauset, A. & Moore, C. (2006). Scale invariance in road networks. *Phys. Rev. E*, 73(2):026130.

Kashima, H.; Tsuda, K. & Inokuchi, A. (2003). Marginalized kernels between labeled graphs. In *International Conference on Machine Learning*, pp. 321--328.

Ke, Y.; Sukthankar, R. & Huston, L. (2004). An efficient parts-based near-duplicate and sub-image retrieval system. In *Proceedings of the 12th Annual ACM International Conference on Multimedia*, MULTIMEDIA '04, pp. 869--876, New York, NY, USA. ACM.

Kimeldorf, G. & Wahba, G. (1971). Some results on tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 33(1):82 – 95.

Kirsch, A. & Mitzenmacher, M. (2006). Distance-sensitive bloom filters. In *Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments and the Third Workshop on Analytic Algorithmics and Combinatorics (Proceedings in Applied Mathematics)*. SIAM. 0898716101.

Kramer, S.; De Raedt, L. & Helma, C. (2001). Molecular feature mining in hiv data. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pp. 136--143, New York, NY, USA. ACM.

Kulis, B. & Grauman, K. (2009). Kernelized locality-sensitive hashing for scalable image search. In *IEEE International Conference on Computer Vision (ICCV)*.

Kuramochi, M. & Karypis, G. (2001). Frequent subgraph discovery. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, ICDM '01, pp. 313--320, Washington, DC, USA. IEEE Computer Society.

Köbler, J. & Verbitsky, O. (2008). From invariants to canonization in parallel. In Hirsch, E.; Razborov, A.; Semenov, A. & Slissenko, A., editores, *Computer Science – Theory and Applications*, volume 5010 of *Lecture Notes in Computer Science*, pp. 216–227. Springer Berlin Heidelberg.

Leskovec, J.; McGlohon, M.; Faloutsos, C.; Glance, N. S. & Hurst, M. (2007). Patterns of cascading behavior in large blog graphs. In *Proceedings of the Seventh SIAM International Conference on Data Mining*, SDM '07, pp. 551--556.

Lo, D.; Cheng, H.; Han, J.; Khoo, S.-C. & Sun, C. (2009). Classification of software behaviors for failure detection: A discriminative pattern mining approach. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pp. 557--566, New York, NY, USA. ACM.

Mizruchi, M. S. (1982). *The American corporate network, 1904-1974 / Mark S. Mizruchi ; foreword by G. William Domhoff*. Sage Publications, Beverly Hills.

Murzin, A. G.; Brenner, S. E.; Hubbard, T. & Chothia, C. (1995). SCOP: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247(4):536--540.

Neuhaus, M. & Bunke, H. (2005). Self-organizing maps for learning the edit costs in graph matching. *IEEE Transaction on System, Man and Cybernetic.*, 35:503--514.

Neuhaus, M. & Bunke, H. (2007). *Bridging the Gap Between Graph Edit Distance and Kernel Machines*. World Scientific Publishing Co., Inc., River Edge, NJ, USA.

Newman, M. E. J. (2003). The structure and function of complex networks. *SIAM Review*, 45:167--256.

Papadimitriou, P.; Dasdan, A. & Garcia-Molina, H. (2008). Web graph similarity for anomaly detection. Technical Report 2008-1, Stanford InfoLab.

Ralaivola, L.; Swamidass, S.; Saigo, H. & Baldi, P. (2005). Graph kernels for chemical informatics. *Neural Networks*, 18(8):1093--1110.

Ramon, J. & Gaertner, T. (2003). Expressivity versus efficiency of graph kernels. In *Proceedings of the First International Workshop on Mining Graphs, Trees and Sequences*, pp. 65–74.

Rapoport, A. & Horvath, W. J. (1961). A study of a large sociogram. *Behavioral Science*, 6(4):279--291.

Samet, H. (1990). *Applications of spatial data structures: Computer graphics, image processing, and GIS*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Sanfeliu, A.; Alquézar, R.; Andrade, J.; Climent, J.; Serratosa, F. & Vergés-Llahí, J. (2002). Graph-based representations and techniques for image processing and image analysis. *Pattern Recognition*, 35(3):639–650.

Schölkopf, B.; Burges, C. J. C. & Smola, A. J., editores (1999). *Advances in kernel methods: support vector learning*. MIT Press, Cambridge, MA, USA.

Scholkopf, B. & Smola, A. J. (2001). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA.

Schölkopf, B.; Tsuda, K. & Vert, J. P., editores (2004). *Kernel Methods in Computational Biology*. MIT Press.

Srinivasan, A.; King, R. D.; Muggleton, S. H. & Sternberg, M. J. E. (1997). The predictive toxicology evaluation challenge. In *Proceedings of the 15th international joint conference on Artifical intelligence - Volume 1*, pp. 4--9, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Stockham, C.; Wang, L.-S. & Warnow, T. (2002). Statistically based postprocessing of phylogenetic analysis by clustering. In *ISMB*, pp. 285–293.

Tatikonda, S. & Parthasarathy, S. (2010). Hashing tree-structured data: Methods and applications. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pp. 429–440.

Todeschini, R.; Consonni, V.; Mannhold, R.; Kubinyi, H. & Timmerman, H. (2000). *Handbook of Molecular Descriptors*. Wiley-VCH.

Tsvetovat, M.; Reminga, J. & Carley, K. M. (2004). Dynetml: Interchange format for rich social network data.

Vishwanathan, S. V. N.; Borgwardt, K. M. & Schraudolph, N. N. (2006). Fast computation of graph kernels. In *In Advances in Neural Information Processing Systems 19 (NIPS 2006*, pp. 1--2. MIT Press.

Weiss, N. A. (2005). *Introductory Statistics*. Addison Wesley, 7 edition.

White, H. D.; Wellman, B. & Nazer, N. (2004). Does citation reflect social structure?: Longitudinal evidence from the "globenet" interdisciplinary research group. *Journal of the American Society for Information Science and Technology*, 55(2):111--126.

Wolfson, H. J. & Rigoutsos, I. (1997). Geometric hashing: An overview.

Yan, X.; Yu, P. S. & Han, J. (2004). Graph indexing: A frequent structure-based approach. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pp. 335--346, New York, NY, USA. ACM.

Yan, X.; Zhu, F.; Yu, P. S. & Han, J. (2006). Feature-based similarity search in graph structures. *ACM Transactions on Database Systems (TODS)*, 31:1418--1453.