LUCAS FARGONI DI IANNI

AUTOMATIZAÇÃO DE PREDIÇÃO DE LINKS EM REDES

São Paulo

2013

LUCAS FARGONI DI IANNI

AUTOMATIZAÇÃO DE PREDIÇÃO DE LINKS EM REDES

Relatório final apresentado à disciplina PMR 2500 – PROJETO DE CONCLUSÃO DE CURSO I PARA OBTENÇÃO DO TÍTULO DE ENGENHEIRO.

Orientador(a): Prof. Dr. Fábio G. Cozman

São Paulo

2013

FICHA CATALOGRÁFICA

Di Ianni, Lucas Fargoni
Automatização de predição de links em redes / L.F. Di Ianni. -- São Paulo, 2013.
80 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.Predição (Automação) 2.Redes de computadores I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II.t.

AGRADECIMENTOS

A meu orientador, Prof. Dr. Fabio G. Cozman que sempre me apoiou na escolha desse projeto, mesmo nos momentos mais difíceis. A minha família, pelo suporte financeiro, o que permitiu que eu terminasse minha graduação com foco apenas nos estudos. A minha namorada, pelo suporte emocional, o qual me trouxe serenidade para que eu conseguisse seguir em frente durante esses cinco anos. A meus colegas de graduação, pelo companheirismo durante os anos de faculdade, me trazendo alegria em fazer parte dessa faculdade. E, finalmente, a meu colega Wilson Leão Neto que foi como um mestre para tanto nas minhas escolhas acadêmicas quanto profissionais.

RESUMO

A predição de links visa prever relacionamentos em uma rede de dados. Uma grande variedade de campos pode se beneficiar dessa tecnologia; por exemplo, a modelagem de ambientes físicos através de grafos visando à movimentação de robôs. Hoje há dificuldade em se obter um programa que aplique de forma automatizada todas as etapas necessárias de avaliação e recomendação de links. Outro ponto negativo no uso de predição de links é a baixa eficácia dos preditores de menor custo computacional. Os algoritmos mais interessantes do ponto de vista de tempo de execução, infelizmente, ainda possuem baixo poder preditivo. Esse relatório descreve a criação de um software automático que realiza todo o processo de predição de links de forma transparente ao usuário, através de uma interface de fácil utilização. Outra contribuição do projeto foi o aumento da capacidade de predição dos algoritmos visando tornar sua aplicação mais confiável.

Palavras-chave: Predição de Links, Software Automático de Predição de Links, Algoritmo Local Supervisionado.

ABSTRACT

Link prediction aims at predicting relationships in a network. A variety of fields can benefit from this tecnology; for instance, physical environments can be modeled by graphs so as to encode robot navigation. It is now difficult to find a computer program that can automatically run all steps needed in link evaluation and recommendation. Another difficult point in link prediction is the low accuracy of predictors with low computational cost. The best algorithms from the point of view of execution time are, unfortunately, of low predictive power. This report describes an automated software package that runs the whole prediction link process in a transparente manner, through an easy-to-use interface. Another contribution of the project was an increase in prediction performance for a predictor with low computational cost.

Keywords: Link Prediction, Automatic Software of Link Prediction, Supervised Local Algorithm.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

O termo “rede” denota um grupo de objetos que pertencem a uma comunidade interagindo entre si. Uma rede pode ser descrita como um composto de nós, ou vértices, que representam os objetos, e uma lista de arestas, ou ligações, que representam uma relação entre nós. Este tipo de estrutura é representada por um grafo. Há uma grande variedade de sistemas que são modelados através desta abordagem na natureza, para diferentes fins. No âmbito mecatrônico, há autores [Dudek et al. 1991] que utilizam de grafos como uma abstração do campo físico de exploração de robôs, onde os vértices são pontos de localização e estes caminham através de arestas utilizando-se de algoritmos de exploração. No campo biológico, há exemplos de estudos das propriedades dinâmicas de redes formadas pela interação entre proteínas [Li et al. 2004] e modelagem de sistemas biológicos para se recuperar informações [Liao et al. 2003]. Além disso, o estudo de redes com evolução temporal tem atraído atenção dos cientistas, especialmente as redes sociais, uma vez que estas são suficientemente complexas a ponto de permitir o estudo da dinâmica dos mecanismos que controlam o sistema [Barabási, Albert-Laszlo et ai. 2002] além das características topológicas e estruturais que regem sua evolução [Kossinets, Gueorgi e Watts 2006].

O problema de predição de links já não é um conceito estranho para o público em geral. Especialmente para os usuários de sites de redes sociais e de comércio eletrônico, onde os usuários se se acostumaram a ver recomendação de amigos [Walter, Frank Edward, Stefano Battiston, e Frank Schweitzer] ou produtos [Sarwar, Badrul, et al. 2000]. Outra aplicação interessante é a previsão de links de sites, onde os autores construíram um previsor de arestas baseado em modelos de cadeia de Markov, criados pelo próprio usuário na sua navegação [Zhu et al. 2002].

Este trabalho descreve um projeto de software na área de predição de links: seus objetivos e a forma de se realizar essa tarefa, tanto em alto nível, através da sua modelagem e em baixo nível. Também se realiza um breve estudo dos algoritmos de predição local e propõe-se uma forma de se aumentar sua capacidade preditiva. Na Seção 2 encontra-se a motivação para o projeto, seus objetivos, e sua metodologia de desenvolvimento. A Seção 3 aborda o problema de predição de links, definindo-o formalmente, revisa os principais índices de avaliação local e propõe uma nova

forma de se analisar a predição baseada nos índices que se mostraram mais efetivos, além de apresentar testes e resultados para esta nova proposição. Na seção 4 expõem-se os detalhes da programação do software, como a utilização do banco de dados e a modelagem do domínio. Por fim relatam-se alguns pontos para trabalhos futuros e finaliza-se com a conclusão do projeto.

2. DESENVOLVIMENTO

2.1 Motivação

O estudo da predição de links está inserido nos ramos da ciência que utilizam grafos para a modelagem de seus fenômenos. Um exemplo real do potencial dessa área de pesquisa pode ser encontrado no estudo das interações celulares da levedura; segundo Haiyuan et al. (2008) apenas vinte por cento destas interações foi mapeada, demonstrando que ainda há uma grande parcela a ser descoberta.

Hoje não há, dentre os principais programas que lidam com redes, funcionalidades que contemplem a predição de links. Além disso, dentre as poucas implementações de algoritmos preditores (contidas em biblioetacas do software R), o que há disponível só é acessível a quem possui conhecimentos de programação, devido à sua interface em baixo nível.

Outro fator que motiva este trabalho é a baixa eficácia dos preditores locais de links. O trabalho de Zhou e Lü (2009), sobre o desempenho de algoritmos que utilizam exclusivamente informações dos vizinhos para a predição, indica que há a possibilidade de se melhorar a precisão para redes genéricas. De maneira geral, sempre os mesmos algoritmos obtêm os melhores resultados, variando entre si de acordo com as propriedades particulares de cada grafo. Quando comparado à taxa de acerto destes, como feito por Liben-Nowell et al. (2007), revela-se que estes preditores preveem diferentes links para as mesmas redes, indicando que há informações que alguns frameworks conseguem adquirir enquanto outros não. Ou seja, apesar de serem sempre os mesmos preditores, as informações utilizadas por cada um não são as mesmas, mostrando que há espaço para algum preditor mais genérico extrair todas essas informações. Outra forma de se verificar a possibilidade de melhoria do poder preditivo encontra-se no estudo da importância do peso das arestas. Demonstrou-se que o desempenho dos algoritmos pode variar de acordo com o modo que estes avaliam o grau de importância das relações, possuindo pontos de maximização da eficácia dos resultados (Lü e Zhou 2011).

Além disso, o crescente aumento de artigos relacionados ao problema da predição de links também serviu como motivacional para o projeto. A ferramenta Ngram Viewer do Google, que realiza buscas de citações em livros, indica que o assunto está em plena ascensão (Figura 1). A sua atualidade também indica que há um grande campo a ser estudado, expressando uma oportunidade de descobertas de interesse acadêmico e comercial.

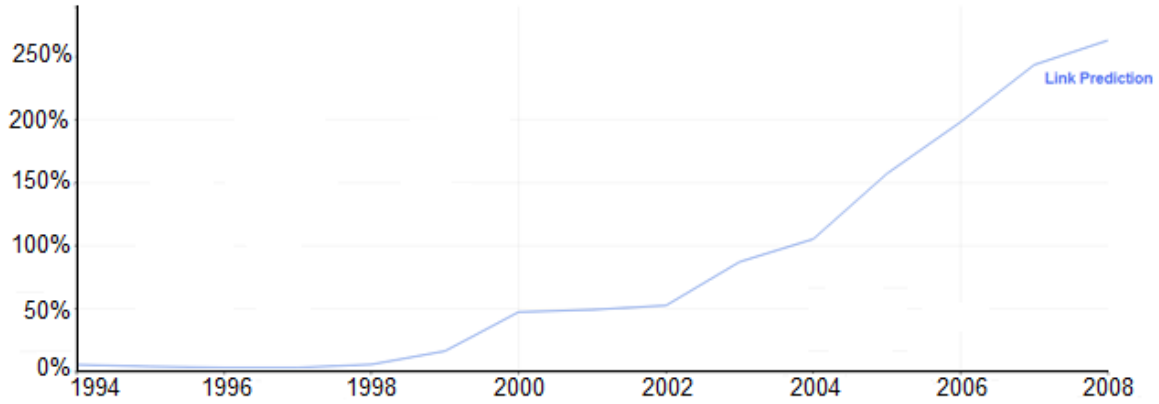

Figura 1 - Número de citações em livros segundo o Google Ngram Viewer.

Portanto, dois fatores são primordiais para a motivação deste trabalho; a inexistência de um software de predição de links e a baixa eficiência dos algoritmos disponíveis. A baixa eficiência dos algoritmos já foi comprovada em trabalhos anteriores e as divergências de resultados de cada framework indicam que de fato há oportunidades de melhoria.

2.2-Objetivos

Como objetivos do trabalho têm-se a elaboração de um software capaz de solucionar o problema da predição de links automatizadamente e o desenvolvimento de um algoritmo que seja capaz de obter um desempenho superior aos atualmente propostos.

2.7 Metodologia

Para a análise do software e sua modelagem utilizaram-se os diagramas UML. O diagrama de atividades exemplifica o fluxo e as decisões dinâmicas do sistema. Os casos de uso foram usa-

dos para levantar as funcionalidades que serão disponibilizadas ao usuário. Utilizou-se um conceito minimalista para esse projeto, visando dar ao usuário à máxima eficiência com o menor número de funcionalidades. Essa abordagem torna o software mais *user-friendly*, ou seja, com maior usabilidade. Em contrapartida tem-se que este acaba apresentando maior nível de complexidade de codificação uma vez que precisa tomar um grande número de decisões próprias.

O diagrama de componentes demonstra como foi elaborada a organização dos pacotes do sistema.

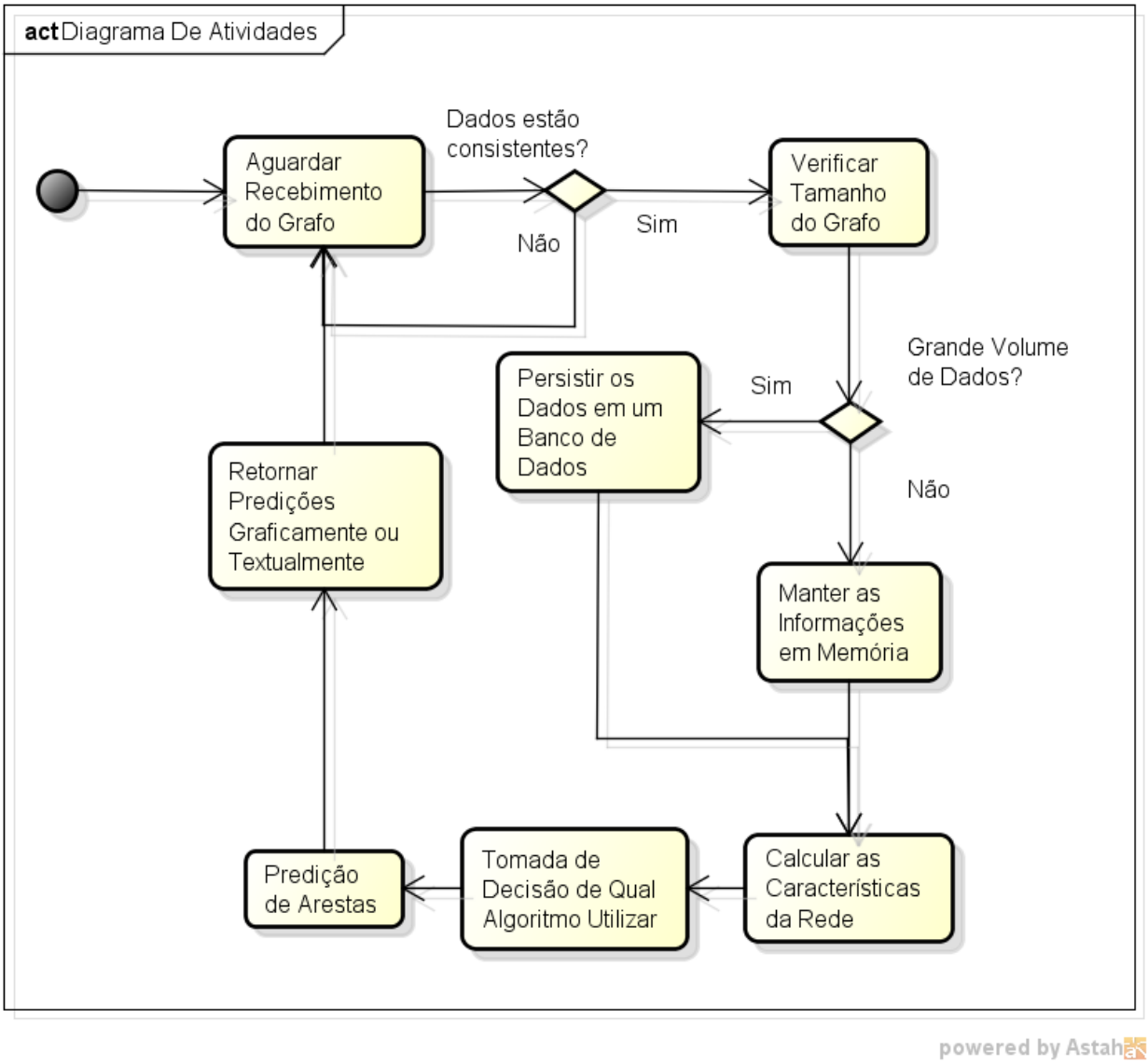

Figura 2- Diagrama de Atividades do Programa de Automatização de Predição de Links.

A Figura 2 indica o diagrama de atividades do programa completo. Ao iniciar, este deve esperar o usuário importar uma rede persistida na memória do computador, em formato específico ou direto do banco de dados. O programa faz uma verificação para garantir que o usuário entrou com dados consistentes evitando travamentos. Em seguida o software realiza a predição. O software calcula o tamanho do grafo, para verificar se há a possibilidade de se utilizar a memória do computador, ou se será necessário persistir os dados em um banco de dados. Depois o software caracteriza a rede de acordo com as suas propriedades e seleciona o melhor algoritmo que se aplica àquela configuração. O programa predizas arestas, criando os grafos de treinamento e teste, processando o algoritmo escolhido e fazendo verificações de validade do método de acordo com as métricas de avaliação. Finalmente o software retorna as informações ao usuário de maneira gráfica, caso haja poucos dados, ou textual, se estiverem em grande quantidade.

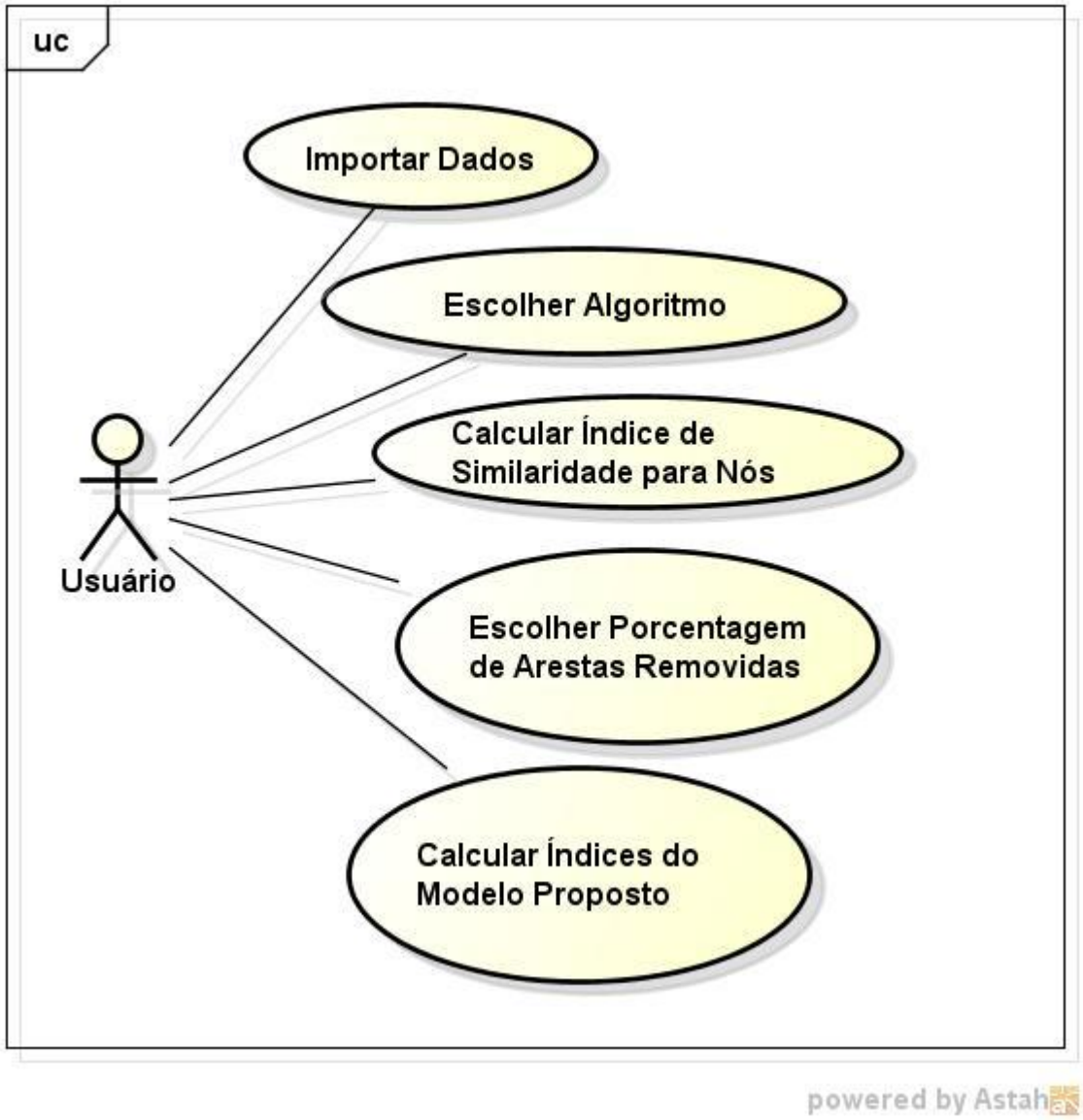

Figura 3 - Diagrama de Casos de Uso do Sistema.

O software desenvolvido apresenta funcionalidades básicas como importar dados, escolher algoritmo, calcular índice de similaridade, escolher a quantidade de arestas removidas para calcular o índice AUC e salvar os dados. Cabe ao programa tomar decisões mais complexas como exibir o grafo graficamente ou em forma de tabela, qual algoritmo escolher para realizar a predição e fazer a computação dos dados em memória virtual ou armazenar em disco rígido. Essa abordagem visa criar uma experiência positiva ao usuário, uma vez que se constitui em algo útil e

de fácil utilização. Espera-se que essas funcionalidades supram as necessidades de quem não possui profundos conhecimentos de tecnologia ou mesmo de predição de links.

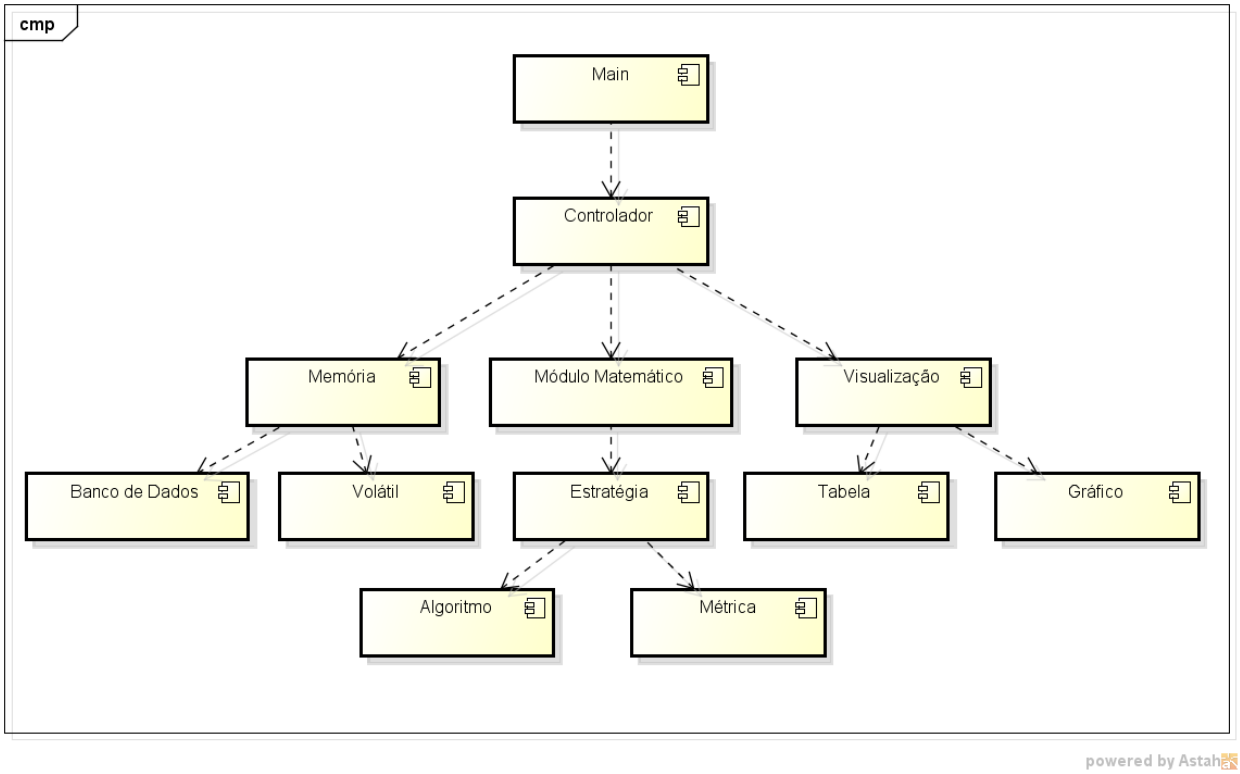

Figura 4 - Diagrama de Componentes do Sistema.

O diagrama de componentes expressa a complexidade exigida ao desenvolvimento do software. O software foi projetado para desacoplar a automatização de cada componente de sua implementação. Assim este apresenta uma camada acima dos algoritmos, da persistência e da visualização, para realizar a tomada de decisão deste.

3. O PROBLEMA DE PREDIÇÃO DE LINKS

3.1 Abordagem Matemática

A abordagem matemática do problema de predição de links foi originalmente formulada por Liben-Nowell e Kleinberg (2007), onde uma rede social é representada por G(V,E), sendo E o conjunto de arestas composto por e=(u,v), que representa uma aresta formada entre os vértices u e v. Para grafos que apresentam evolução temporal atribuiu-se as arestas marcas temporais (timestamp) e separamos a rede em dois períodos de tempo diferentes. Se houver múltiplas interações consideram-se arestas paralelas. O subgrafo G(t,t') representa o grafo G restrito a arestas com marcas temporais entre t e t'. Na tarefa de predição de links nós podemos selecionar um intervalo de tempo de treinamento [t0,t0'] e um intervalo de testes [t1,t1'] (onde t0'<t1) para testar a eficácia do método. A lista de arestas presentes no intervalo de treinamento deve estar presente no intervalo de testes.

Outra abordagem foi desenvolvida por Lü e Zhou (2011) para grafos estáticos. Sem a característica da evolução temporal o intervalo de treinamento deve ser criado removendo os links aleatoriamente do grafo principal. A lista de links observados E deve ser dividido em dois conjuntos, o de treinamento, E^T, que contém os links originais menos os removidos (considerado a informação conhecida), e o de prova, E^P, contendo somente os links removidos que serão utilizados para a predição. É possível observar que $E^T \cup E^P = E$ e $E^T \cap E^P = 0$. Os vizinhos de um determinado nó são denotados por Γ(nó) e o seu grau |Γ(nó)|.

3.2 Métricas de Avaliação

Há duas formas mais utilizadas para a medição de desempenho dos resultados dos algoritmos de predição. Uma é a precisão, que avalia a razão entre os valores verdadeiramente positivos e os verdadeiramente negativos. A outra é a Receiver Operating Characteristic (ROC), que avalia graficamente o desempenho sobre a taxa de links verdadeiramente positivos e falsamente

positivos. Uma métrica mais conveniente que a bidimensional ROC, muito usada na comunidade acadêmica, é a unidimensional Area Under Curve (AUC) [Hanley and McNeil 1982], que é a área calculada sob a curva.

Para a construção do gráfico ROC, cada aresta contida no vetor calculado deve ser comparada com cada aresta do conjunto de prova de arestas. Se a aresta calculada está contida na lista de prova, então é um exemplo verdadeiramente positivo; se não estiver, é um exemplo falsamente positivo. A lista de links calculados deve ser ordenada em ordem decrescente de contagem, uma vez que o limite máximo indica a origem do gráfico, como indicado por Fawcett (2004).

A métrica AUC de um algoritmo ranqueia uma instância positiva aleatoriamente escolhida com valor maior que uma instância negativa aleatoriamente escolhida, de acordo com suas propriedades estatísticas (Fawcett 2004). Então é esperado que um link removido receba um score maior que um link que não existe. Nesse trabalho utilizou-se a métrica AUC proposta por Hand and Till (2001). Essa métrica foi utilizada para determinar a eficácia de cada algoritmo.

$$AUC = (S_0 - n_{0*}(n_0+1) / 2) / (n_{0*}n_1),$$

onde S_0 é a soma de todas as posições de todas as arestas positivas na lista decrescente de resultados de determinado framework. A variável n_0 é o número de arestas positivas, ou seja, o tamanho do conjunto de prova, E^P, e n_1 é o número de links negativos, ou seja, o tamanho da lista de resultados menos o tamanho do conjunto de prova. Quanto mais distante de 0,5 for o valor do AUC, mais confiável a predição será feita em relação à pura chance.

3.3 Estado da Arte

Os algoritmos que utilizam a similaridade topológica como característica para prever links são basicamente separados em dois grupos, os que utilizam o nó e sua vizinhança para a predição e os que utilizam o caminho do grafo (também descritos como índices de similaridade locais e globais (Lü et al. 2011)). A grande vantagem dessa abordagem é que os algoritmos são genéricos podendo ser aplicados a qualquer tipo de dado em qualquer tipo de domínio, não sendo requisitadas informações sobre as características da rede modelada [Al Hasan e Zaki 2011].

Apesar de o tema predição de links ter sido proposto recentemente por Liben-Nowell et al. (2007), o conceito de predizer relações, ou pelo menos propor índices de similaridade em nós, em redes já aparecia na literatura em diversos outros formatos. No estudo da evolução temporal das colaborações científicas, Newman (2011) demonstrou que as probabilidades de dois colaboradores vierem a se relacionar aumenta de acordo com o número de outras colaborações que ambos têm em comum. Na área de redes na web, apresentam-se trabalhos que propõe índices quantitativos para avaliar o quão similares são dois nós a partir de suas relações (Adamic e Adar (2003)) além de estudos sobre como lidar com similaridade entre páginas da web (Sergey e Page 1998).

A seguir são apresentados as métricas que utilizam as características topológicas locais do grafo para realizar a predição. O termo "algoritmo" é usado para se referir a métricas.

A métricas apresentadas a seguir são consideradas os mais simples pela comunidade acadêmica, uma vez que utilizam apenas as informações contidas nos próprios nós e seus vizinhos. Para cada par de nós é atribuído um valor, chamado de s_{xy}, definido como sendo a similaridade entre os nós. Por se tratarem de uma abordagem simplória geralmente obtém resultados inferiores a modelos mais complexos, como por exemplo os que utilizam informações globais; porém apresentam custo computacional muito mais baixo visto que necessitam de poucas informações para a predição. O objetivo principal dos algoritmos é de dar notas mais altas aos nós mais similares (Lü et al. 2011).

3.3.1 – Algoritmos e Códigos

Common Neighbors: Proposto por Newman (2001), a ideia por trás dessa métrica é de que quanto mais vizinhos dois nós tem em comum, mais eles estão propensos a estarem ligados. É notável então que essa predição é aplicada para caminhos de comprimento dois. Esta talvez seja o índice de similaridade mais simples, uma vez que considera apenas a intersecção entre dois nós para o cálculo do score.

$$score(x,y) = |\Gamma(x) \cap \Gamma(y)|$$

Jaccard Coefficient: Esse índice normaliza o score calculado pelo algoritmo Common Neighbors adicionando uma divisão pela união entre os dois nós (Jaccard 1901).

$$score(x,y) = |\Gamma(x) \cap \Gamma(y)| / |\Gamma(x) \cup \Gamma(y)|$$

Leich-Holme-Newman Index: Esse índice calcula sua similaridade baseado na contagem de vizinhos em comum dividido pela quantidade esperada desses (Leicht, Holme and Newman 2006).

$$score(x,y) = |\Gamma(x) \cap \Gamma(y)| / |\Gamma(x)| \times |\Gamma(y)|$$

Salton Index: Índice muito similar ao anterior, porém aplicando ao denominador a raiz quadrada [Salton and McGill 1986].

$$score(x,y) = |\Gamma(x) \cap \Gamma(y)| / \sqrt{|\Gamma(x)| \times |\Gamma(y)|}$$

Sørensen Index: Ao invés de penalizar baseado na quantidade esperada de vizinhos em comum, como o Leich-Holme-Newman, este índice penaliza de acordo com o maior numero possível de vizinhos (Sørensen 1948).

$$score(x,y) = 2*|\Gamma(x) \cap \Gamma(y)| / |\Gamma(x)| + |\Gamma(y)|$$

Hub Promoted Index: Esse índice promove a adjacência a nós centrais, visto que penaliza o menor grau entre o par de nós levados em consideração (Ravasz, Erzsébet et al. 2002).

$$score(x,y) = |\Gamma(x) \cap \Gamma(y)| / \min\{|\Gamma(x)|, |\Gamma(y)|\}$$

Hub Depressed Index: Proposto por Zhou et al (2009) índice igual ao anterior porém com denominador revertido.

$$score(x,y) = |\Gamma(x) \cap \Gamma(y)| / \max\{|\Gamma(x)|, |\Gamma(y)|\}$$

Preferential Attachment: Esse algoritmo usa como métrica a esperada quantidade de vizinhos que dois nós tem em comum. Usado originalmente em redes evolutivas (Barabási et al. 1999), foi adaptado para servir como índice de predição.

$$score(x,y) = |\Gamma(x)| \times |\Gamma(y)|$$

Adamic/Adar: O primeiro índice a utilizar a informação específica do vizinho em consideração ao cálculo da similaridade. Penaliza os índices com alta quantidade de informações pois estes teoricamente são os menos importantes para futuros relacionamentos (Adamic and Adar 2003).

$$score(x,y) = \sum_{z \in \Gamma(x) \cap \Gamma(y)} 1 / \log |\Gamma(z)|$$

Resource Allocation: Análogo ao índice Adamic/Adar, porém penaliza mais severamente aqueles com grau mais alto (Zhou et al 2009).

$$score(x,y) = \sum_{z \in \Gamma(x) \cap \Gamma(y)} |\Gamma(z)|$$

3.4 Dados Utilizados

Para o estudo do comportamento dos preditores e as características que influenciam no seu desempenho foram utilizadas três redes e cinco características topológicas. Escolheram-se grafos advindos da área biológica devido à facilidade em obtê-los. As características topológicas foram escolhidas a fim de se capturar os seus diferentes efeitos na predição de links.

Tabela 1. Redes e suas Características Topológicas.

Rede	Nós	Arestas	Densidade	Modularidade	Clustering
Dolphin	62	159	0,084	0,526	0,303
C. Elegans	297	2148	0,049	0,393	0,308
Disease	1419	2738	0,003	0,874	0,819

A primeira rede Dolphins, representada na Fig. 5, consiste em uma rede social, com arestas sem direcionamento, das associações frequentes entre 62 golfinhos em uma comunidade vi-

vendo em Doubtful Sound, Nova Zelândia [Lusseau et al. 2003]. Este gráfico é o menor de todos, considerando tanto o número de arestas quanto de nós. No entanto, contém a maior densidade e a segunda maior modularidade, demonstrando que este pode apresentar bons resultados para preditores baseados em contagem de vizinhos em comum. O coeficiente de agrupamento (clustering) é quase o mesmo das outras duas redes.

Figura 5 - Rede Dolphins.

A segunda, representada na Fig. 6, é uma rede dirigida, ponderada representando a rede neural de C. elegans. Para fins acadêmicos transformou-a em sem direção e sem peso. O número de arestas é alto, comparável com o último grafo, mas com um número muito menor de nós, assegurando uma elevada densidade. No entanto, há poucas comunidades como mostrado pela modularidade, criando um ambiente interessante para testes, uma vez que todos os nós estão separados em alguns poucos grupos com elevado grau.

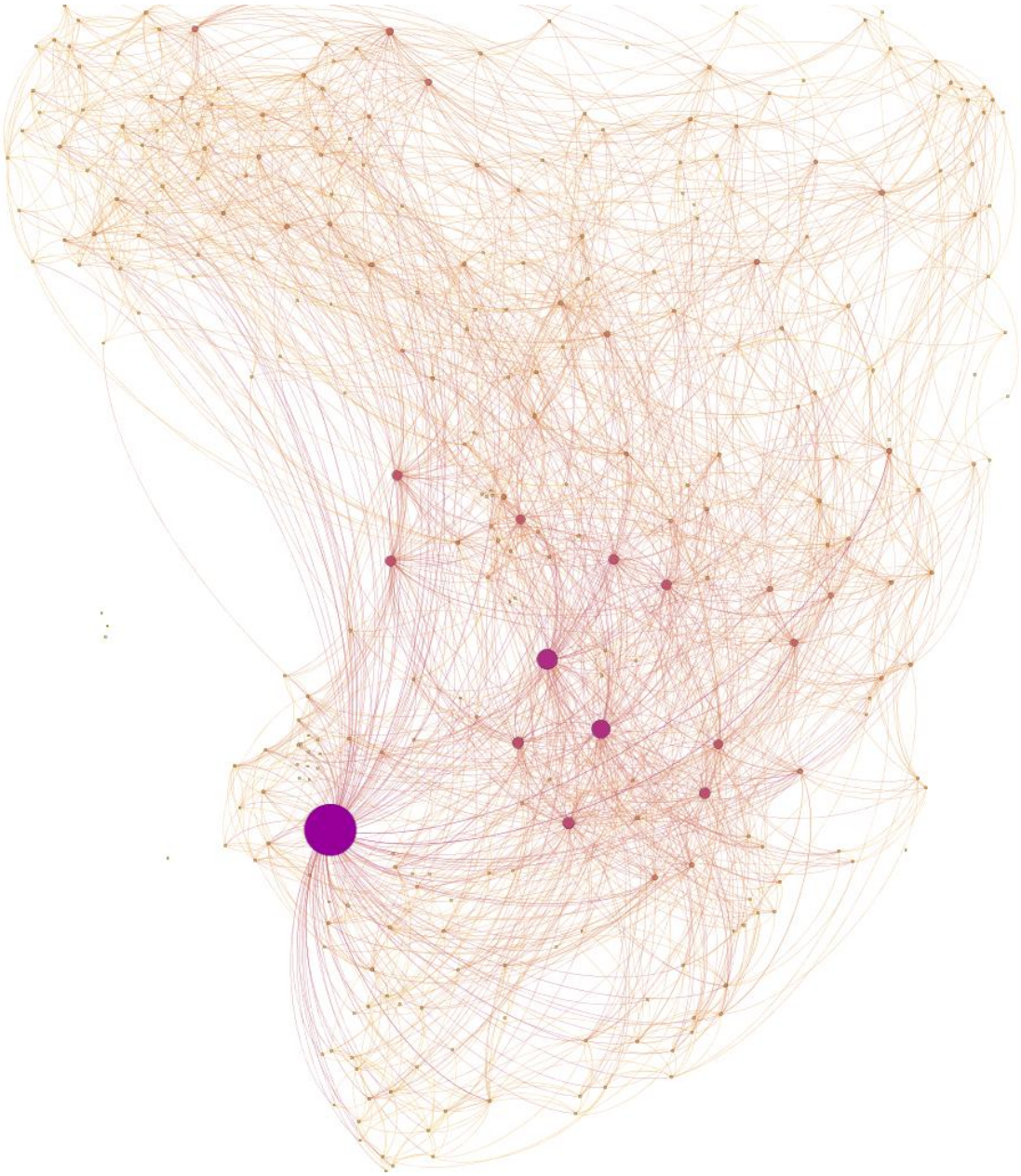

Figura 6 - Rede C. Elegans.

Mostrada na Fig. 7, a terceira rede é uma rede de genes de doenças e distúrbios ligados por conhecidas associações de desordem genética, indicando a origem genética comum de muitas doenças. Genes associados com distúrbios semelhantes mostram maior probabilidade de interações físicas entre si, e maior expressão de similaridade de perfil, apoiando a existência de módulos funcionais específicos de doenças distintas, conforme explicado por Bastian et al. (2009). O ta-

manho do grafo é comparável com aos utilizados em estudos prévios de Liben-Nowell e Kleinberg (2007) e Zhou et al. (2009). A baixa densidade e alta modularidade indicam que os nós estão concentrados em comunidades.

Figura 7 - Rede Disease

3.5 Experimentos

Para os experimentos analisou-se o desempenho de algoritmos para responder a questões como a forma que a eficácia deles varia de acordo com o número de arestas removidas, e dado o melhor conjunto de links removidos, qual algoritmo fornece o maior número de previsões corretas entre os melhores links recomendados. Esta análise é interessante caso procura-se um número reduzido de arestas previstas porque, como não se sabe quais serão as arestas futuras, confia-se nas ligações mais bem avaliadas. Dois tipos de métricas foram utilizados para realizar os testes, ROC e AUC. AUC foi utilizado para comparar a eficiência entre os algoritmos para todos os níveis de ligações retiradas e determinar qual é o melhor algoritmo de predição em determinada percentagem de arestas removidas. Todos os valores da AUC foram obtidos a partir de uma média aritmética de 100 testes. A comparação aparece nas tabelas variando-se o número de arestas removidas dentre metade até 90% das ligações totais. Porém sendo uma medida unidimensional, AUC não é capaz de reportar o comportamento preciso da lista de arestas preditas. Para analisar quais frameworks dão as melhores previsões para os links mais bem avaliados, ou seja, as ligações com valores mais altos score, devemos analisar a curva ROC.

3.5.1 Resultados

Na Tabela 2 é possível ver a comparação de todos os preditores para a rede Dolphins. Para a coluna de cinquenta por cento de links removidos, a informação não é totalmente confiável, uma vez que a melhor previsão é de cerca de quatorze por cento maior do que uma aleatória, tornando-se desaconselhável a predição nesta configuração. Mas é interessante que, mesmo para baixa quantidade de dados na rede, o desempenho geral continua sendo maior do que o puro acaso. Certamente há uma persistência de informações nesta rede, uma vez que o melhor algoritmo é o mesmo para todas as percentagens de ligações removidas. Esse comportamento pode ser responsabilizado à alta densidade, que mantém a característica topológica dos nós mesmo para uma grande retirada dos links. Percebe-se que o desempenho global aumenta significativamente ao aumentar os dados para análise do preditor, isto é, diminuindo a percentagem de ligações removi-

das, conforme esperado. É possível concluir que para esta rede o algoritmo Hub Depressed apresenta o maior poder preditivo, enquanto o Preferential Attachment tem o pior prognóstico. Curiosamente, o último quase não aumenta o seu desempenho ao longo do teste, mantendo a menor eficiência de predição para todas as colunas.

Tabela 2. Comparação do valor AUC para a rede Dolphins para diferentes porcentagens de links retirados.

% of Links Removidos	50%	40%	30%	20%	10%
Common Neighbors	0,637	0,674	0,707	**0,732**	0,746
Adamic/Adar	0,635	0,673	0,705	0,730	0,747
Salton Index	0,635	0,667	0,693	0,714	0,729
Resource Allocation	0,635	0,673	0,705	0,730	0,746
Jaccards Coefficient	0,636	0,671	0,700	0,720	0,734
Sorensen	**0,640**	**0,675**	0,707	0,731	**0,748**
Hub Depressed	**0,640**	**0,675**	**0,708**	**0,732**	**0,748**
Hub Promoted	0,634	0,665	0,690	0,709	0,721
Leich Holme Newman	0,634	0,665	0,689	0,708	0,721
Preferential Attachment	*0,583*	*0,588*	*0,598*	*0,606*	*0,599*
Média	0,632	0,664	0,692	0,714	0,726

Na Figura 8 é possível ver graficamente a superioridade do algoritmo Hub Depressed sobre o Preferential Attachment. Surpreendentemente para os primeiros nós Preferential Attachment tem pior desempenho do que um algoritmo baseado em previsões aleatórias. Enquanto o valor AUC só pode medir a eficiência de todas as previsões, o ROC pode dar-nos qual algoritmo obtém as melhores previsões para os melhores links avaliados. Na Figura 9 nota-se uma inversão da precisão da previsão para as primeiras arestas, por isso se alguém pretende escolher somente as arestas mais bem ranqueadas como as corretas é preferível usar Preferential Attachment para esta tarefa, mesmo que este apresente uma pontuação AUC menor que o framework Hub Depressed.

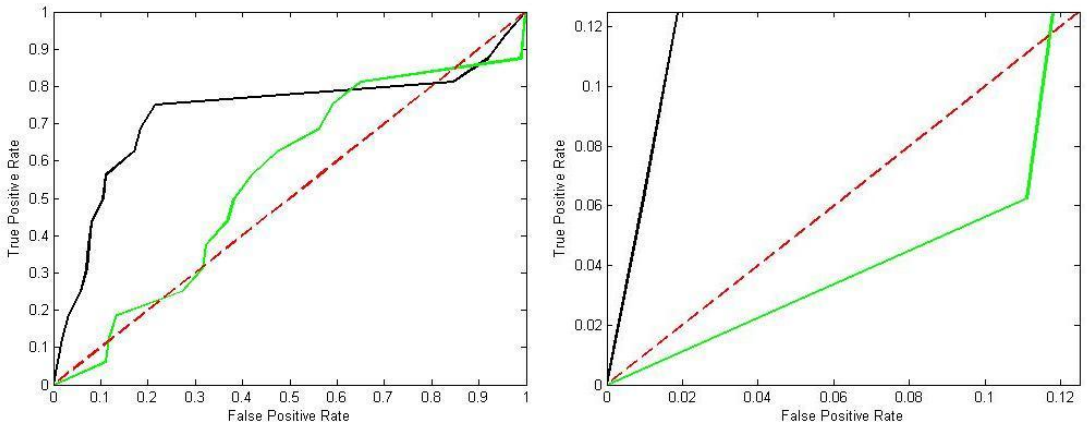

Figura 8 – A curva ROC para a rede Dolphins com 10% dos links removidos. A mesma curva aparece à direita, com a origem ampliada. A linha preta é o algoritmo Hub Depressed predictor enquanto a verde é o Preferential Attachment. O algoritmo aleatório é definido pela linha vermelha.

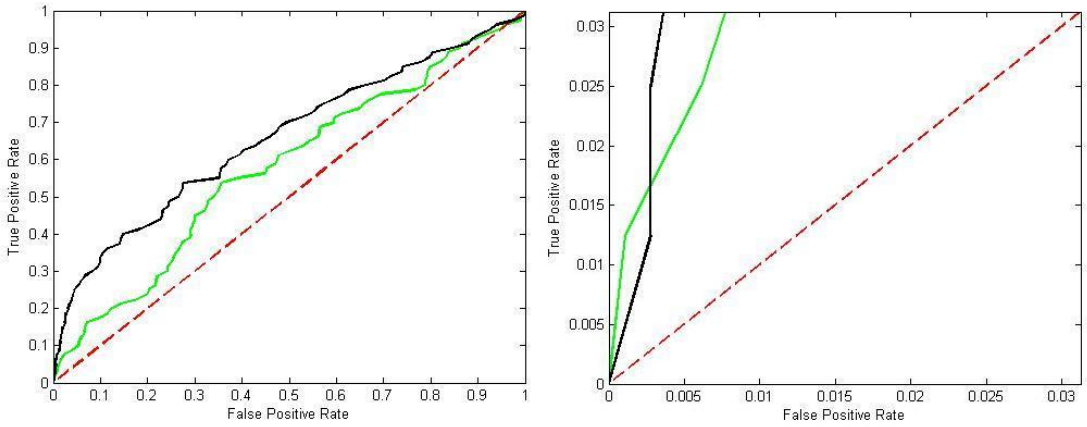

Figura 9- A curva ROC para a rede Dolphins com 50% dos links removidos. A mesma curva aparece à direita, com a origem ampliada. A linha preta é o algoritmo Hub Depressed predictor enquanto a verde é o Preferential Attachment. O algoritmo aleatório é definido pela linha vermelha.

A Tabela 3 consiste na mesma análise, mas para a rede de C. elegans. Há claramente dois algoritmos proeminentes para esta rede, Adamic / Adar e o Resource Alocation. Embora o primeiro não tenha sido o melhor para o grafo anterior (embora tenha chegado perto), no segundo pode-se ver o potencial do framework. Há alguns empates técnicos entre os dois algoritmos para algumas porcentagens específicas de links removidos, mas o maior peso dado aos nós populares pelo algoritmo Resource Allocation acaba por torná-lo mais eficiente. Como na Tabela 2, é possível verificar o fenômeno de aumentar a precisão quando se aumenta o número de arestas no grafo de treinamento. Também é verificável a inferioridade de previsão de algoritmos baseados em

vizinhos comuns para os baseados no conceito Adamic/Adar, graças às propriedades de agrupamento da rede, como discutido por Zhou et al (2009). Mais uma vez Preferential Attachment é o indicador com menor perda de precisão quando removido uma grande quantidade de dados a partir do grafo original, indicando uma propriedade desse indicador.

Tabela 3. Comparação do valor AUC para a rede C. Elegans para diferentes porcentagens de links retirados.

% of Links Removidos	50%	40%	30%	20%	10%
Common Neighbors	0,714	0,756	0,787	0,813	0,833
Adamic/Adar	0,726	0,771	0,804	0,830	0,852
Salton Index	0,701	0,731	0,751	0,770	0,788
Resource Allocation	0,726	**0,772**	**0,806**	**0,832**	**0,853**
Jaccards Coefficient	0,690	0,720	0,737	0,752	0,765
Sorensen	0,691	0,724	0,745	0,765	0,782
Hub Depressed	0,691	0,721	0,740	0,757	0,771
Hub Promoted	0,707	0,740	0,761	0,780	0,796
Leich Holme Newman	*0,690*	*0,708*	*0,713*	*0,717*	*0,718*
Preferential Attachment	**0,727**	0,732	0,735	0,739	0,740
Média	0,708	0,741	0,762	0,781	0,795

As próximas duas figuras mostram como os valores AUC analisados isoladamente podem não ser interessantes. A Figura 10 revela o desempenho abaixo do esperado dos preditores Leich Holme Newman quando analisado somente as arestas mais bem ranqueadas. Mesmo tendo um valor AUC (ou seja, a área sobre a curva) maior que 0,5 este não obtém um desempenho melhor do que um algoritmo puramente aleatório. Essa figura também confirma a superioridade do algoritmo Adamic/ Adar, pois curva deste é acima das outras para quase todo o gráfico. Na Figura 11 há outra revelação interessante. De acordo com AUC, a predição Preferential Attachment apresenta um desempenho melhor do que Adamic / Adar, mas especialmente para as arestas mais bem avaliadas, Adamic/Adar desempenha muito melhor, indicando superioridade da precisão a partir deste ponto de vista.

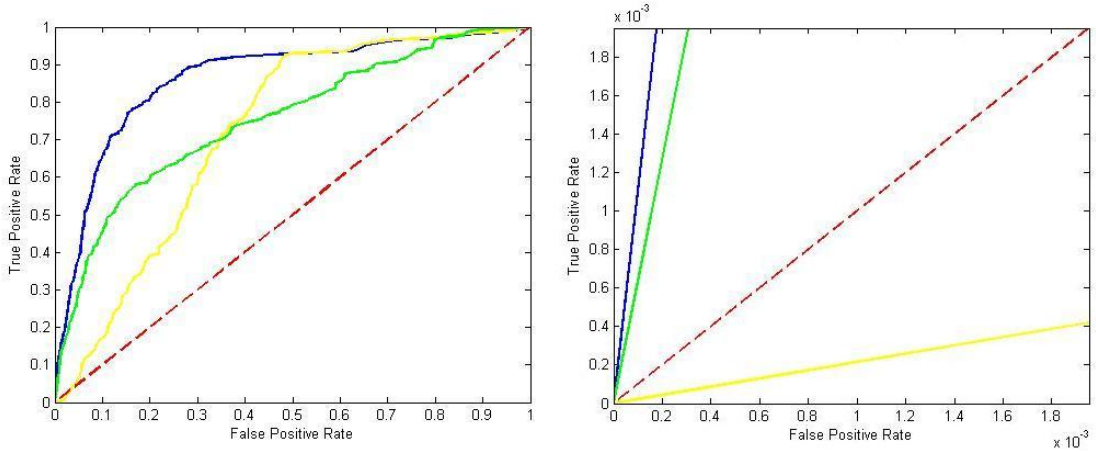

Figura 10 - A curva ROC para a rede C. Elegans com 10% dos links removidos. A mesma curva aparece à direita, com a origem ampliada. A linha azul é o algoritmo Adamic/Adar, a verde é o Preferential Attachment e a amarela é o Leich Holme Newman. O algoritmo aleatório é definido pela linha vermelha.

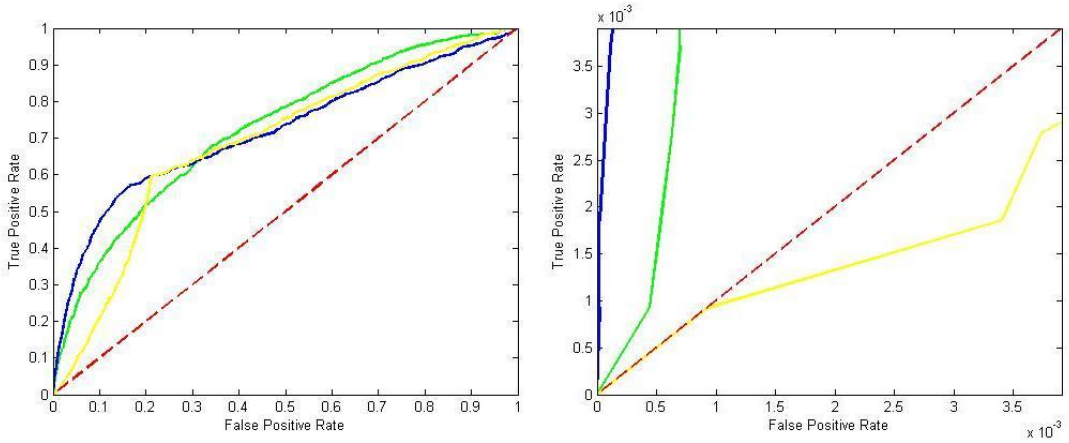

Figura 11 - A curva ROC para a rede C. Elegans com 50% dos links removidos. A mesma curva aparece à direita, com a origem ampliada. A linha azul é o algoritmo Adamic/Adar, a verde é o Preferential Attachment e a amarela é o Leich Holme Newman. O algoritmo aleatório é definido pela linha vermelha.

Na Tabela 4, apresentam-se os resultados para a rede Disease. Para as duas maiores quantidades de dados disponíveis para realizar a predição (ou seja, as duas últimas colunas) há um empate entre dois preditores. Adamic / Adar e Resource Allocation.

Tabela 4 - Comparação do valor AUC para a rede Disease para diferentes porcentagens de links retirados.

% of Links Removidos	50%	40%	30%	20%	10%
Common Neighbors	0,756	0,807	0,845	0,874	0,899
Adamic/Adar	0,757	0,808	0,846	**0,875**	**0,900**
Resource Allocation	0,757	0,808	0,846	**0,875**	**0,900**
Preferential Attachment	0,661	*0,665*	*0,664*	*0,663*	*0,663*
Jaccards Coefficient	**0,788**	**0,824**	**0,852**	0,874	0,896
Sorensen	0,766	0,810	0,844	0,872	0,895
Hub Depressed	0,766	0,810	0,844	0,872	0,895
Hub Promoted	0,656	0,688	0,715	0,741	0,766
Leich Holme Newman	*0,655*	0,687	0,715	0,739	0,764
Salton Index	0,656	0,688	0,715	0,741	0,766
Média	0,725	0,764	0,794	0,818	0,841

Comparando todas as tabelas, é possível verificar algumas características gerais da previsão de link. Uma vez que a coluna com o maior número de arestas apresenta o índice AUC mais elevado, fica evidente que quanto maior o número de informações no gráfico melhor é a predição. Além disso, quanto menor for a remoção de arestas, maior será o desempenho dos algoritmos. Assim a análise do desempenho é fortemente influenciada pelo número de arestas removidas, uma vez que os melhores algoritmos para cada rede variaram para diferentes porcentagens de links removidos, o que pode levar a conclusões erradas de desempenho se não for considerado o ponto correto de porcentagem de links retirados. A surpresa foi a métrica Preferential Attachment, com a menor variação percentual de desempenho, em torno de 2% para todas as redes, o que lhe permitiu ser o indicador mais confiável quando o gráfico contém poucas informações enquanto outros preditores perdem vigorosamente a qualidade de precisão.

3.6 Algoritmo Proposto

O algoritmo proposto deve seguir algumas regras para se encaixar no subgrupo de preditores de links através de características locais. Devido à natureza desse tipo de índice de similaridade, o algoritmo não deve usar qualquer informação estrutural que a rede contenha, como menor distância entre dois nós ou a qualidade dos nós baseado em quantas ligações este recebe através da rede (Sergey e Page 1998). O framework também deve se basear em dados estritamente pertencentes a cada par de nós, como características de vértices em comum, por exemplo.

O fato de se utilizar de dados locais não impede que o algoritmo proposto baseie-se em estender a ideia de se utilizar informações topológicas. Algoritmos probabilísticos, por exemplo, baseiam sua predição em modelos que são construídos a partir da abstração da estrutura do grafo (Lü e Zhou 2011). Outra forma de se extrair a informação de redes e adaptá-la para uma função é através de algoritmos supervisionados. Apesar de terem grande potencial preditivo, pois estes conseguem moldar-se de acordo com as diferentes características do grafo, ainda são pouco estudados. Há relatos de benchmark entre os principais algoritmos supervisionados para a predição de links (Al Hasan et al. 2006), porém há poucas publicações a respeito de frameworks que foram feitos exclusivamente para o problema de predição de links.

3.6.1 Características da Predição Local

Os algoritmos de predição local são basicamente divididos em três grupos menores: os que utilizam a contagem de nós relacionados entre dois elementos, os que utilizam as características do nó relacionado e os que utilizam simplesmente as características dos nós analisados. O primeiro grupo, representado pelo algoritmo Common Neighbor e seus variantes, utiliza-se como índice de similaridade o número de nós que são comuns aos dois vértices analisados, baseando-se na ideia de que quanto mais elementos em comuns há entre dois nós, maior é a probabilidade destes estarem ligados. O segundo grupo, representado por Adamic/Adar, considera como índice de similaridade as características topológicas dos nós em comum. Por fim, o último grupo ignora totalmente a informação de relacionamento em comum entre dois nós e considera apenas o núme-

ro de relacionamentos totais entre estes. Neste grupo há apenas um algoritmo, o Preferential Attachment. Na Figura 12 mostra-se, para os dois principais algoritmos citados, Common Neighbors e Adamic/Adar, a relação de nós em comum e a valorização destes no cálculo do índice de similaridade.

Os dois grupos que demonstraram uma maior capacidade preditiva servirão de base para o algoritmo a ser proposto. Dentro destes, emulara-se aqueles algoritmos que apresentaram boa capacidade preditiva. Segundo os testes propostos, os frameworks Common Neighbors e Adamic/Adar, de modo geral, apresentaram bons resultados. Considerando-se todos os resultados, esses quando não foram os melhores preditores, ficaram próximos a eles. A métrica utilizada por ambos está representada na Figura 12.

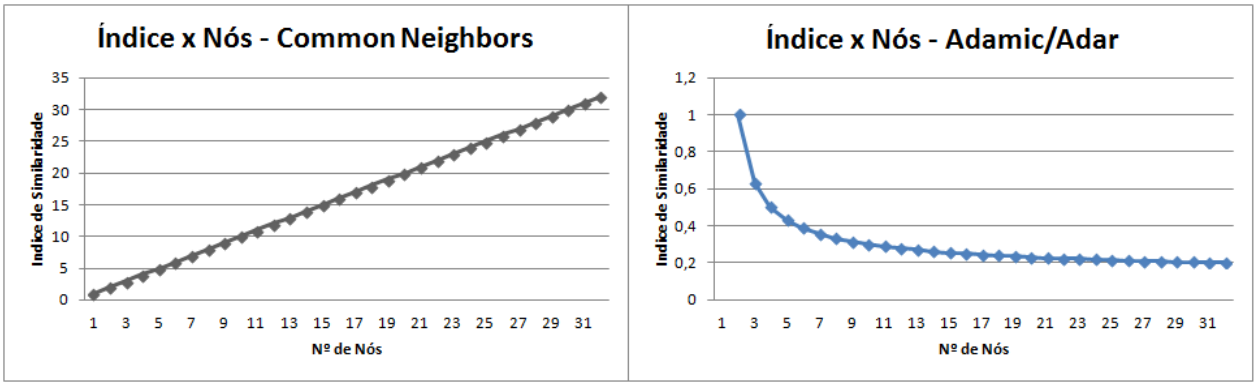

Figura 12 - Comparação entre Índices de Similaridade dos Algoritmos Common Neighbors e Adamic/Adar.

A ideia por trás do novo algoritmo é que ele contém as propriedades dos dois algoritmos citados, podendo adaptar-se às características do grafo convenientemente. Para que isso fosse possível adaptaram-se as propriedades de predição dos dois algoritmos em um novo e adicionaram-se constantes para que este fosse capaz de se modificar até alcançar a máxima eficiência.

$$score(x,y) = \sum_{z \in \Gamma(x) \cap \Gamma(y)} |\Gamma(z)|^{A} / \log |\Gamma(z)|^{B}$$

Com essa fórmula nós temos embutido as características dos algoritmos do Common Neighbors e do Adamic Adar. Os expoentes dos termos servem para criar uma fórmula adaptável ao grafo de modo supervisionado. Percebe-se que o algoritmo proposto simula fielmente o Common

Neighbors quando ambas constantes valem zero e simula o Adamic Adar quando A vale zero e B vale um.

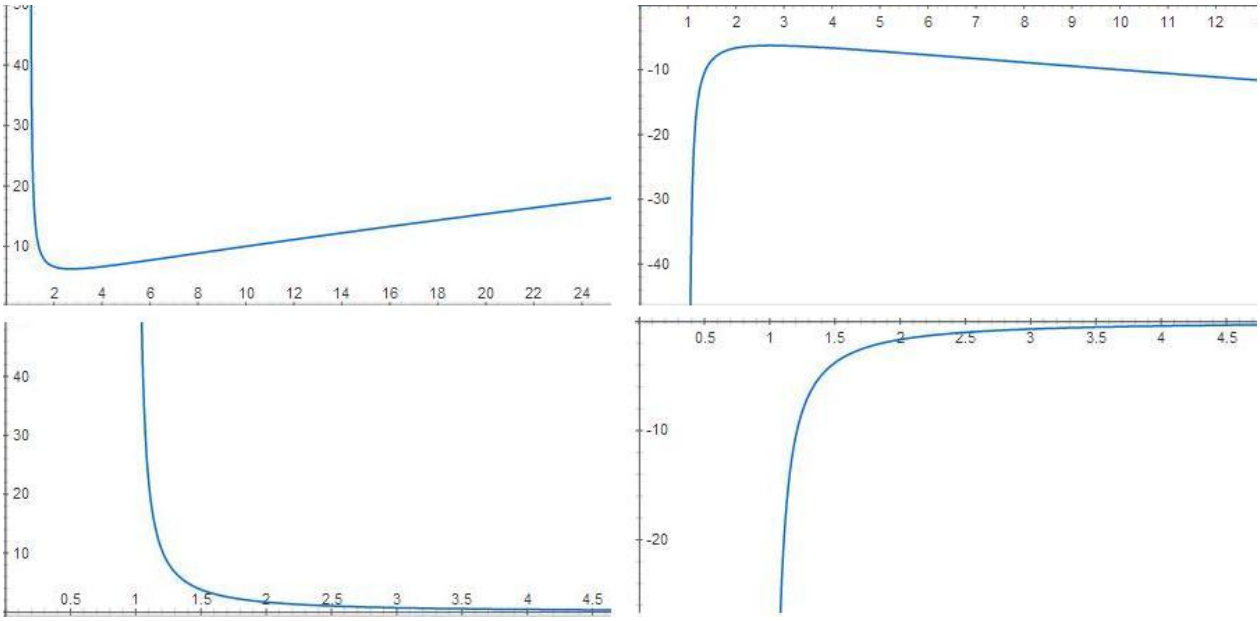

Figura 13- Característica da predição dos quatro algoritmos gerados. No canto superior esquerdo temos o gráfico para A maior que zero e B maior que zero. No canto superior direito temos A maior que zero e B menor. No canto inferior esquerdo temos A menor que zero e B maior que zero. No canto inferior direito temos A e B menores que zero.

Analisando-se as características de predição para o algoritmo proposto percebe-se que há quatro grandes grupos presentes. O primeiro, indicado na Figura 13, no canto superior esquerdo, indica que o algoritmo procura valorizar os nós com maiores números de amigos em detrimento dos com poucos amigos. Nota-se que o algoritmo que surge ao utilizarem-se valores de A maior que zero e B menor que zero é exatamente o oposto. Para a métrica no canto inferior esquerdo, nota-se que há a valorização de nós com poucos relacionamentos, porém sem aumentar a valorização ao longo do aumento de ligações. Por fim, a curva que surge ao definirmos A menor que zero e B menor que zero, apresenta um relacionamento oposto à penúltima curva.

3.6.2 Experimentos

Os testes realizados visaram analisar o comportamento da predição de links para um amplo espectro de valores para o algoritmo proposto. Calcularam-se os valores de AUC percorrendo

um espaço quadrado variando-se tanto a constante A quanto a B de -50 até +50, realizando-se um número de testes em cada ponto o suficiente para reduzir o desvio padrão a valores efetivos para o teste. Os grafos utilizados foram os explicados na seção 3.4, porém, diferentemente dos testes da seção 3.5, foram retirados apenas 10% dos links totais.

Os resultados da rede Dolphins estão expressos na Figura 14. A característica marcante desse resultado é que há dois tipos de algoritmos quando analisado a capacidade preditiva. Na parte superior estão os menos eficientes, em especial os que apresentam A maior que zero e B menor que zero os quais valorizam apenas os nós com poucos relacionamentos e desvaloriza os com elevado número de relacionamentos, enquanto os com melhor poder preditivo são aqueles que valorizam os nós com poucos relacionamentos, porém desvalorizam de forma menos aguda os que possuem muitas arestas. Nota-se que estes estão divididos através de uma linha que contém os algoritmos com maior capacidade preditiva.

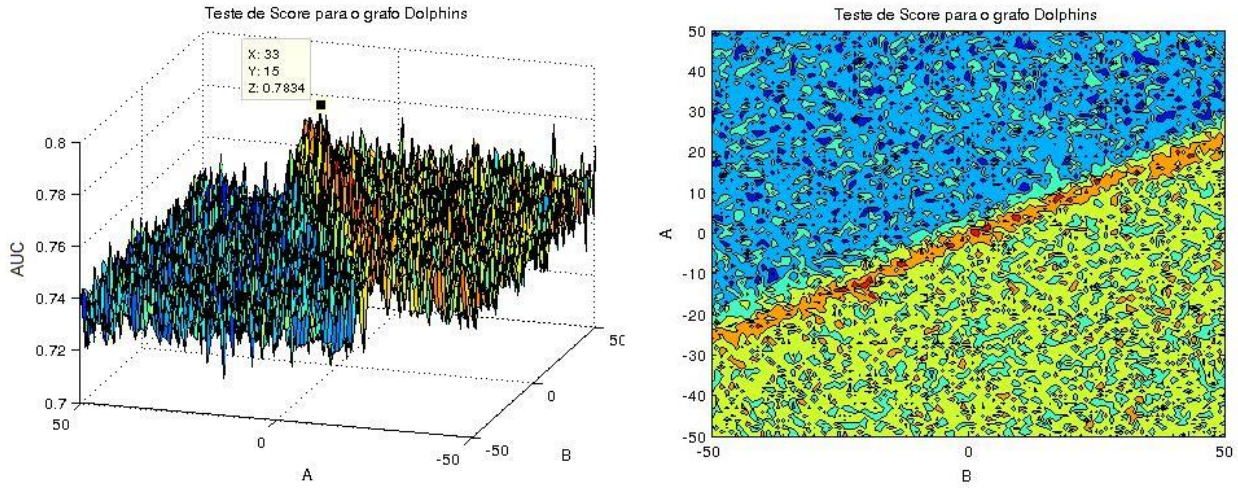

Figura 14 – Gráficos em 3D e 2D contendo os valores AUC para o algoritmo proposto para o grafo Dolphins.

Os quadrantes que apresentam os maiores valores para predição, contidos nos quadrante inferior esquerdo e superior direito, tem-se dois tipos distintos de algoritmos com um desempenho satisfatório. Dentre estes dois, é no primeiro quadrante que se encontra o algoritmo mais favorável à predição. Este algoritmo, representado na Figura 15, apresenta características que não são emuláveis por nenhum outro proposto. Este dá notas altíssimas aos nós com maior número de

relacionamentos, característica dos algorimos que venceram nos testes anteriores, como o Hub Depressed. É notável também que por possuir características tão distintas este apresenta um poder de predição muito superior aos outros algoritmos. Entretanto, o algoritmo que apresenta menor poder de predição está no quadrante superior esquerdo, quando B vale -22 e Y vale 28. Esse algoritmo é basicamente o Adamic Adar distorcido para haver menos variação de índice entre as diferentes quantidades de nós. Nota-se então, que os algoritmos utilizados até aqui nos trabalhos acadêmicos, por sua natureza não supervisionada, não são capazes de predizer com a eficiência total que o grafo é capaz de fornecer.

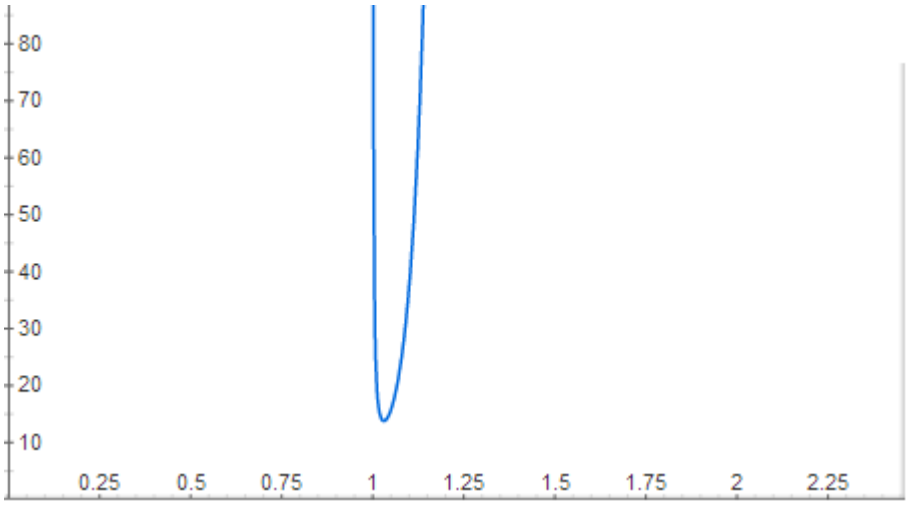

Figura 15 – Gráfico que representa a métrica de predição ideal para o grafo Dolphins.

Na Figura 15 temos o resultado do desvio padrão para o grafo Dolphins. Nota-se que o ponto onde ocorre o desvio padrão está uma casa decimal abaixo da diferença de valores do melhor algoritmo da literatura e do algoritmo proposto, demonstrando-se a validade do teste.

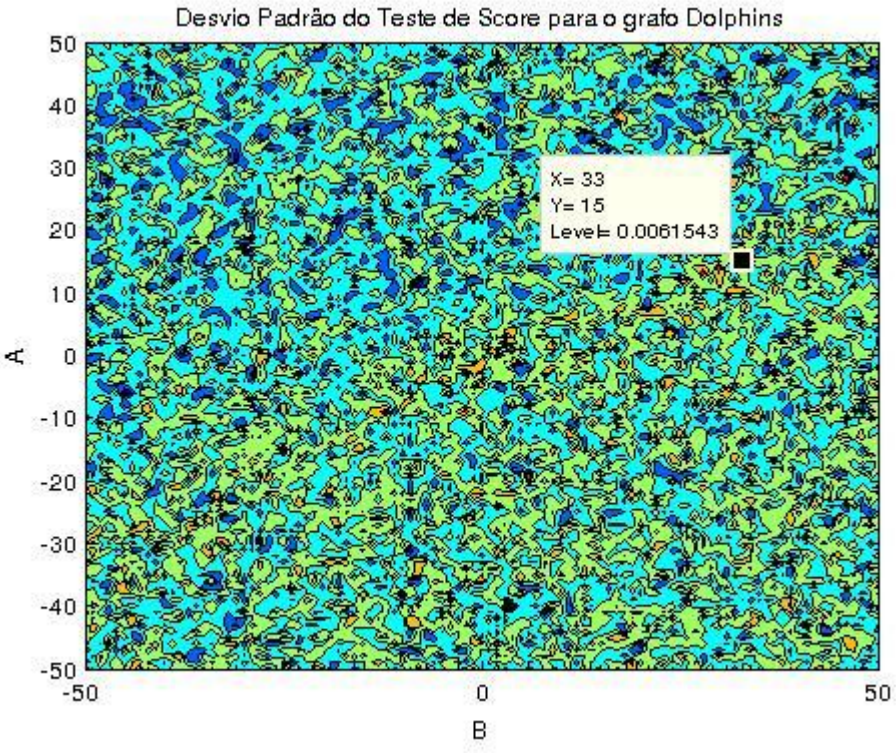

Figura 16– Desvio padrão para cada valor de A e B realizado para o teste para o grafo Dolphins.

Na Tabela 6 demonstrou-se a eficiência do algoritmo proposto. Nesses resultados expostos encontram-se os valores para o algoritmo Adamic/Adar e o vencedor dos testes Hub Depressed. O algoritmo proposto demonstra claramente um poder preditivo muito maior quando comparado aos outros dois.

Tabela 3- Valores AUC para os principais algoritmos para o grafo Dolfins com 10% das arestas removidas.

% of Links Removidos	10%
Adamic/Adar	0,747
Hub Depressed	0,748
Algoritmo Proposto	0,783

Os mesmos testes foram realizados para o grafo C.Elegans. Na Figura 17 vê-se o resultado em 3D, enquanto na Figura 18 o mesmo resultado é expresso em duas dimensões. Identificou-se para este grafo o mesmo comportamento do anterior, ou seja, a existência de duas regiões distin-

tas de predição. E, assim como para o grafo Dolphins, os algoritmos que apresentaram o melhor resultado estão no terceiro e quarto quadrante do grafo, separados desta vez por uma linha menos inclinada, porém qualitativamente de mesmo formato. Outro ponto de convergência entre os testes foi a localização do algoritmo mais eficiente. Apesar de não estarem sobre o mesmo ponto, como era de se esperar, pois cada grafo possui suas próprias características, estão no mesmo local relativo em relação aos gráficos, pois o ponto máximo encontra-se logo após a transição entre o patamar de baixo poder preditivo e o de alto poder preditivo.

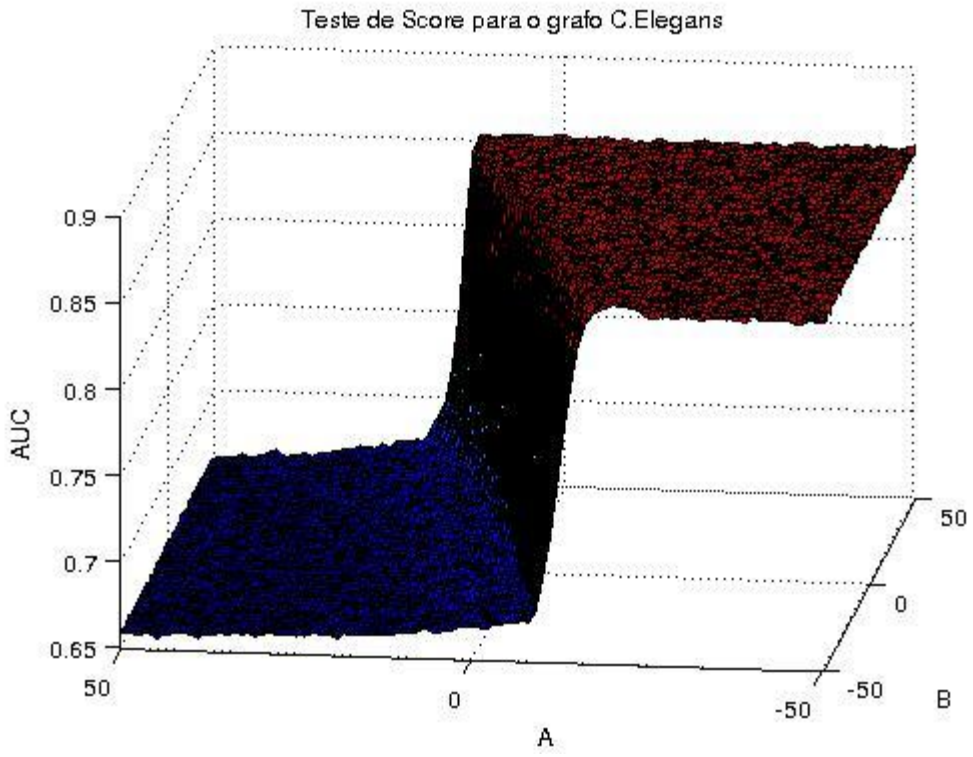

Figura 17 – Gráfico 3D do resultado do teste de score para o grafo C.Elegans.

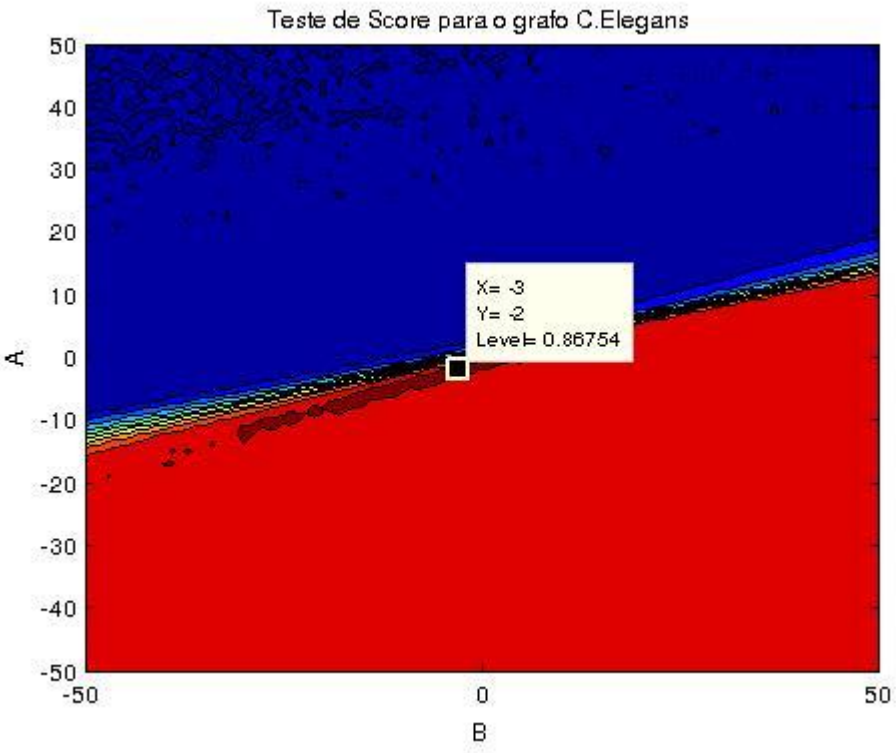

Figura 18 – Gráfico 3D do resultado do teste de score para o grafo C.Elegans.

Na Figura 19 prova-se que o resultado do teste é válido matematicamente, visto que o desvio padrão para o ponto de máximo do gráfico está uma casa decimal abaixo do índice de divergência entre o melhor algoritmo o algoritmo proposto.

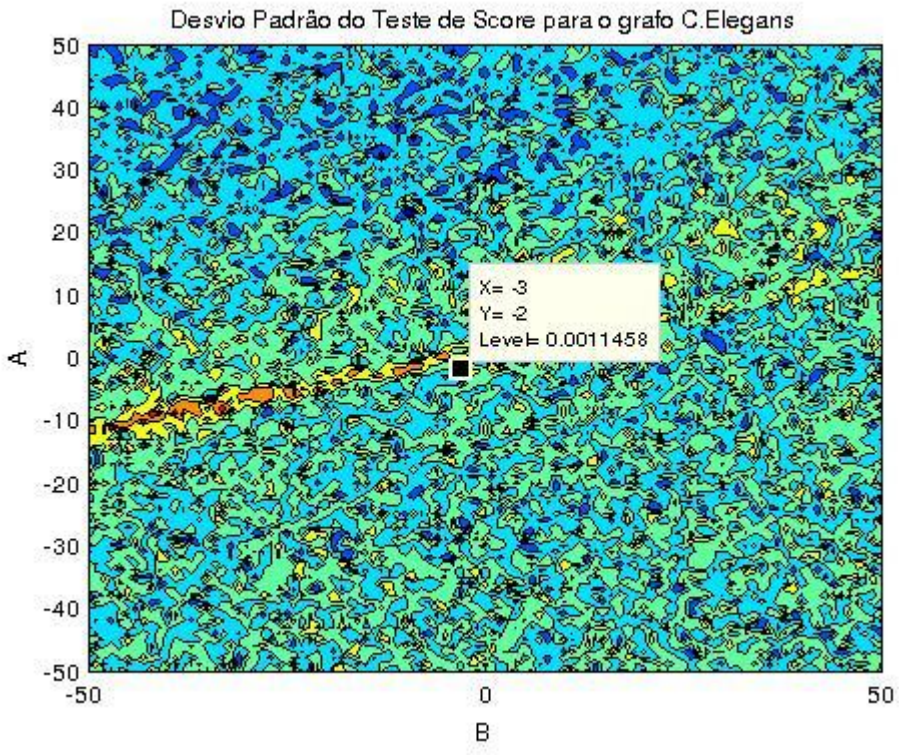

Figura 19 - Desvio padrão para cada valor de A e B realizado para o teste para o grafo Dolphins.

Na Tabela 7 demonstrou-se a eficiência do algoritmo proposto. Nesses resultados expostos encontram-se os valores para o algoritmo Adamic/Adar e o vencedor dos testes Resource Allocation. O algoritmo apresenta um desempenho inferior quando comparado ao grafo Dolphins, porém ainda apresenta um poder preditivo superior aos algoritmos já relatados.

Tabela 4 - Valores AUC para os principais algoritmos para o grafo C.Elegans com 10% das arestas removidas

% of Links Removidos	10%
Adamic/Adar	0,833
Resource Allocation	0,853
Algoritmo Proposto	0,867

4. SOFTWARE

4.1 Componente de Memória

O componente de memória do software foi projetado para ser tanto flexível a grandes quantidades de dados como para alto desempenho em baixa quantidade de dados. Na Figura 20 demonstram-se as diferenças entre as duas abordagens, em termos quantitativos. Enquanto que para um baixo número de nós a abordagem mais indicada é a manutenção dos dados em memória volátil, conforme o número de elementos cresce há a necessidade de uma maneira mais eficiente e mais robusta para o tratamento destes. Além disso, devido à natureza do problema de predição de links, necessita-se haver a persistência de dois grafos simultâneos, ainda que logicamente apenas através de alteração das propriedades dos nós e arestas, um relacionado ao instante anterior ao de predição e um representando o período de testes, aumentando os requisitos de software.

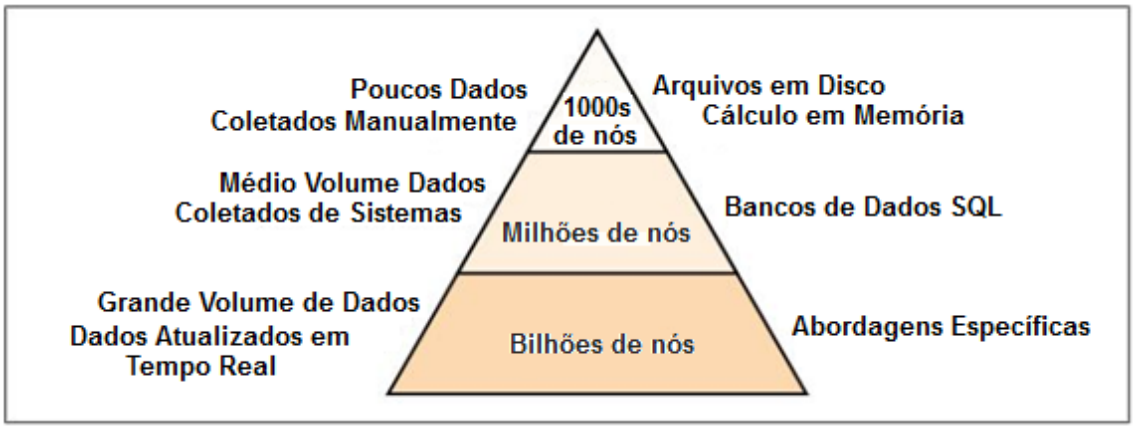

Figura 20 - Quantidade de Dados e sua Abordagem de Persistência.

Para este projeto abordou-se a solução de alocação de memória virtual para grafos pequenos e para grandes quantidades de nós buscou-se soluções que suportam maiores volumes, como a persistência em disco rígido através de banco de dados baseado em grafos porque se espera que o programa tenha a máxima escalabilidade possível.

4.2 Banco de Dados Baseado em Grafos

Os bancos de dados relacionais se tornaram peças fundamentais de sistemas de informação nos últimos anos. Sua força pode ser vista a partir da grande oferta de produtos de diferentes empresas, em particular de gigantes da tecnologia. Essa vitalidade deveu-se a eficiência em se modelar dados através de relações e entidades e também por apresentar uma linguagem unificada e eficiente. Porém, esse tipo de bancos de dados apresenta a inabilidade de se adaptar a mudanças no domínio, pois este é modelado a partir de dados fixos e bem mapeado, ou seja, apresenta baixa escalabilidade. Assim, em áreas onde a topologia das informações e sua interconectividade é mais importante que o domínio por si só, esse tipo de abordagem pode não ser a mais eficiente.

O surgimento de banco de dados baseados em grafos foi visto como uma forma de se representar mais facilmente dados que não apresentam uma estrutura lógica tão rígida. Esse tipo de banco armazena qualquer tipo de estrutura de dados na forma de grafos de maneira completamente genérica, deixando a modelagem a cargo do engenheiro de software. Essa modelagem é facilitada especialmente para a necessidade de se persistir informações que apresentam relacionamentos mutáveis ou adaptáveis durante o tempo. Também se deve considerar a diferença entre os objetos de modelagem para desta abordagem. Diferentemente de banco de dados relacionais, aqui o domínio apresenta características em relação a uma particular entidade e suas ligações, como por exemplo, propriedades específicas de cada elemento ou a presença de vizinhos diretos que visam somente atribuir características mutáveis.

Portanto, devido aos requisitos de memória necessários escolheu-se a utilização de banco de dados. Entretanto, diferentemente de aplicações usuais que utilizam banco de dados relacionais, a elevada diferença de domínios utilizados e as necessidades específicas de se realizar pesquisas em torno de nós e links, escolheu-se o banco de dados baseado em grafos. Além disso, também se considerou a maior eficiência destes quando realizado querys de busca através de seus nós. Segundo Vicknair et al. (2010) o banco Neo4J apresenta uma vantagem de tempo muito superior ao banco relacional MySQL. A Tabela 2 deste trabalho indica o potencial de um banco de dados baseado em grafos para o problema de predição de links. Nas colunas que demonstram os resultados para buscas em profundidade quatro e zero em um conjunto de nós entre mil e um mi-

lhão, a velocidade de busca do banco de dados Neo4J mostrou-se muito mais rápida em quase todos os casos (exceto por um).

4.3 Escolha do Produto

Os critérios para a escolha do banco de dados de grafos dentre os que estão disponíveis no mercado foram tanto de ordem objetiva quanto subjetiva. Objetivamente deseja-se um produto que apresente alta eficiência, que seja grátis (se possível software livre, pois assim há a possibilidade de se ter uma comunidade mais ativa), visto que o projeto não tem financiamento, e, por fim, facilidade de utilização; particularmente suporte a tecnologia Java foi um diferencial nesse quesito por ser a linguagem do projeto e também vista no curso. As métricas subjetivas consideradas foram nível de maturidade da tecnologia, uma vez que softwares mais recentes estão mais suscetíveis a erros e uma rede de utilização ampla, visto que a ideia é disponibilizar o programa para a comunidade.

Foram analisados para os principais produtos encontrados, Neo4J, HyperGraphDB, Oracle NoSQL Database EE e DEX, uma série de critérios considerados objetivos. Um benchmark entre alguns desses bancos demonstrou que DEX e Neo4J são os mais eficientes do mercado (Dominguez-Sal 2010). Todos os testes foram realizados utilizando as interfaces Java do banco, o que torna o teste alinhado com os objetivos deste projeto. Os testes foram realizados dentro da escala de nós que se pretende utilizar nesse trabalho, mil, três mil e um milhão com o número total de objetos (nós e arestas) variando de dez mil até mais de nove milhões. Para grandes quantidades de artefatos apenas os bancos Neo4J, DEX e Jena conseguiram realizar a carga inicial. Mas mesmo assim, para a maior quantidade de dados Jena se mostrou incapaz de obter desempenho satisfatório. Dentre os produtos DEX e Neo4J, vê-se um desempenho mais interessante do primeiro na realização da inserção de dados (Kernel 1), na busca por subgrupos de arestas (Kernel 2) (este teste, porém não é interessante para esse trabalho, visto que aqui realizamos apenas a procura por nós). Para o teste mais significativo para a aplicação de predição de links através de características topológicas locais, o Kernel 3, os resultados foram favoráveis ao Neo4J. Para pequenas quantidades de nós há praticamente um empate técnico entre todos os bancos, porém o Neo4J escala melhor para valores elevados de nós. Além disso, o este também apresenta como vantagem (menos significativa) a criação de dados com menor memória alocada. Assim, para o quesito de de-

sempenho o banco Neo4J foi considerado o produto ideal para este trabalho. A necessidade de se utilizar um software livre gerou grandes restrições ao problema. Considerando-se os quatro bancos listados apenas o HypergraphDB é cem por cento de graça. O fabricante de DEX fornece uma versão ultraleve que serve apenas para testes. O Neo4J apresenta uma versão para a comunidade, porém sempre anterior ao estado da arte do software assim como o banco de dados da Oracle. De acordo com essa restrição, o único banco retirado de hipótese foi o DEX. O último requisito necessário considera a integração do banco com a tecnologia Java (ou seja, facilidade de integração ao projeto). Dos bancos remanescentes, Neo4J e HypergraphDB, nenhum apresentou problemas de conectividade com aplicações em Java.

Para os critérios subjetivos verificou-se que a colocação no mercado do produto acaba sendo resultado de um produto maduro e muito aplicado. Segundo o site http://db-engines.com/en/ranking_trend/graph+dbms hoje o banco líder de mercado é o Neo4J, o qual supre todas as necessidades subjetivas analisadas.

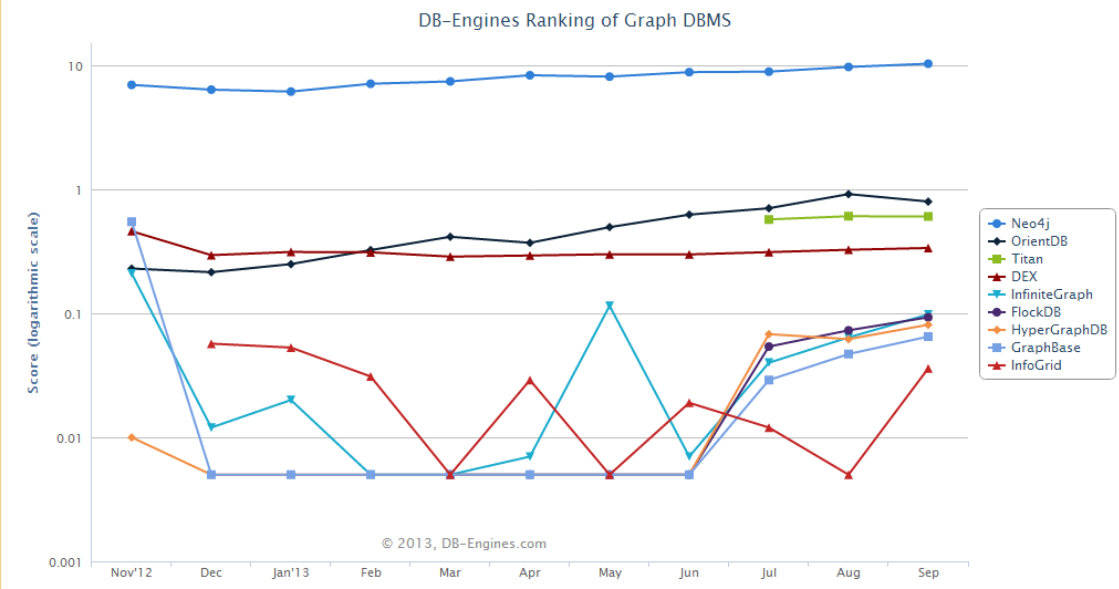

Figura 21 - Comparação de Popularidade dos Bancos de Dados

Portanto, segundo as analises feitas, tanto objetiva, como subjetiva, resultaram na escolha do banco de dados baseado em grafos Neo4J.

4.4 Características do Banco de Dados Neo4J

Neo4j é um dos mais populares bancos de dados baseados em grafo da atualidade em produção desde 2013, com licença livre e comercial, escrito totalmente em Java e open source. Esse produto apresenta algumas características interessantes para esse projeto, como o suporte a grafos genéricos, através da possibilidade de se realizar ligações extras a um determinado nó e a capacidade de atribuir características a qualquer elemento do grafo em qualquer instante, tornando-o uma ferramenta versátil e interessante para a predição de links. Além disso, devido a sua própria implementação, possui uma alta afinidade com a linguagem Java, contendo mais de uma API de acesso aos dados persistidos, sendo que cada uma possui características únicas para a escolha de sua utilização.

4.5-Estrutura de Dados em Memória Volátil

Para pequenos grafos, de até dois mil nós, dependendo da memória RAM disponível, uma alternativa é a utilização de estruturas de dados que mantenham todo o grafo em memória volátil. A utilização dessa abordagem tem a vantagem de se beneficiar da velocidade com que essa memória trabalha, muito superior ao disco rígido, diminuindo o tempo computacional gasto com os cálculos. Para essa abordagem utilizou-se a tecnologia JgraphT. Este framework implementa eficientemente uma estrutura de grafos de forma genérica com a capacidade de se adicionar propriedades aos elementos.

```
public class CacheDataBase implements DataBase {

	private static Graph<String, DefaultEdge> instance = new SimpleGraph<String, DefaultEd-
ge>(DefaultEdge.class);

	public static Graph<String, DefaultEdge> getInstance() {
		return instance;
	}

	@Override
	public void createNode(String label) {
		instance.addVertex(label);
	}

	@Override
	public void createRelationship(String firstNodeId, String secondNodeId) {
		instance.addEdge(firstNodeId, secondNodeId);
	}
```

```
	@Override
	public void createNodesFriendsProperty() {
		Set<String> firstNode = instance.vertexSet();
		Set<String> secondNode = instance.vertexSet();

		for (String first : firstNode) {
			for (String second : secondNode) {
				Set<DefaultEdge> edges = instance.getAllEdges(first, second);
				int size = edges.size();

				instance.addVertex(first + second);
				instance.addEdge(first, String.valueOf(size));
			}
		}

	}

	@Override
	public List getAllNodes() {
		return (List) instance.edgeSet();
	}

	@Override
	public List neighborsOf(String id) {
		List neighbors = Graphs.neighborListOf(instance, id);
		return neighbors;
	}

	@Override
	public void clear() {
		instance = new SimpleGraph<String, DefaultEdge>(DefaultEdge.class);
	}

}
```

4.6-Modelagem

A modelagem para a persistência em forma de grafo considera como elementos para a persistência as entidades participantes no domínio e o relacionamento entre elas. Apesar de o problema de predição de links não apresentar uma alta variedade de tipos de nós e relacionamentos, alguns cuidados em relação à modelagem podem ser tomados a fim de se diminuir o tempo computacional dos cálculos. Basicamente há a existência de um tipo de nó genérico e um tipo relacionamento bidirecional entre esses, uma vez que o banco exige direcionamento das arestas.

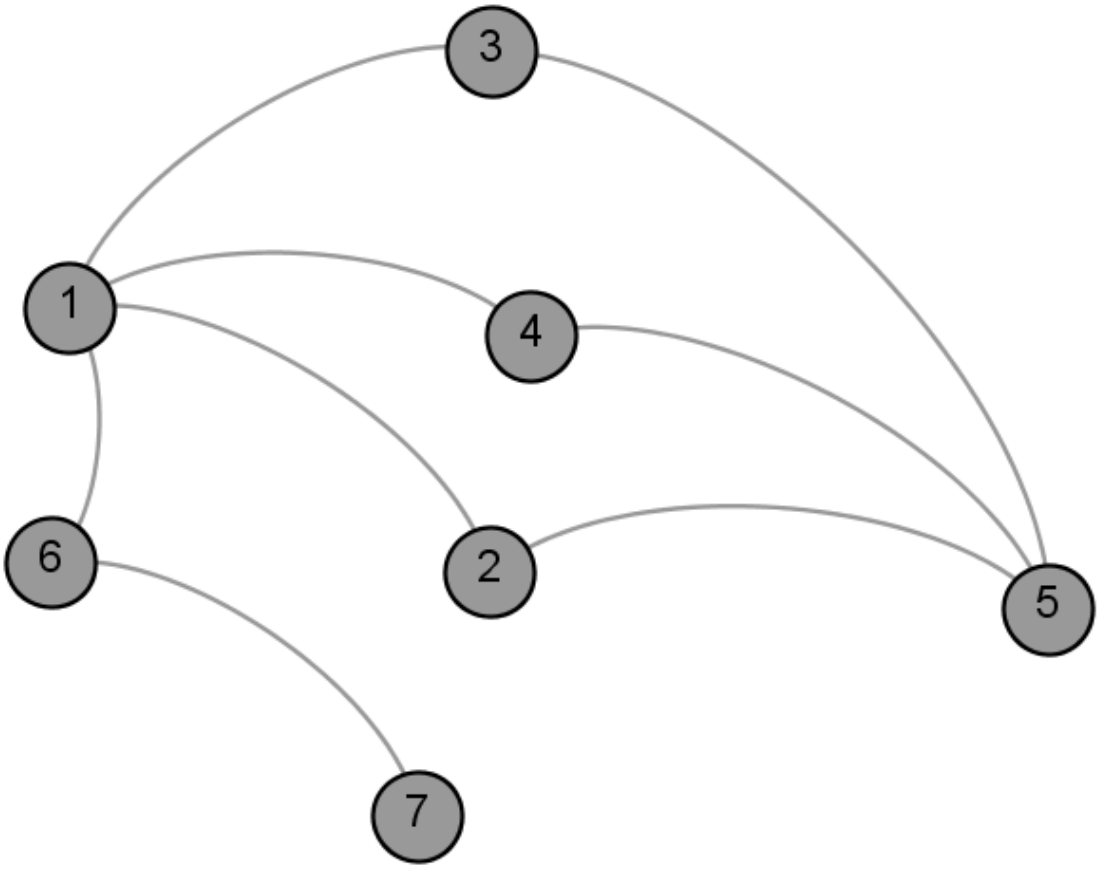

Figura 22 - Rede Exemplo da Modelagem

Porém, esse domínio pode ser estendido visando facilitar o cálculo da predição de relacionamentos. A adição de entidades ou propriedades visando expressar quantitativamente as principais características topológicas necessárias para a predição, como o número de vizinhos de determinado nó utilizado, por exemplo, em algoritmos como Adamic/Adar, pode trazer elevados benefícios computacionais quando considerado o cálculo de grande volume de dados, uma vez que o custo de se buscar por um nó é inferior ao de se buscar por todos vizinhos.

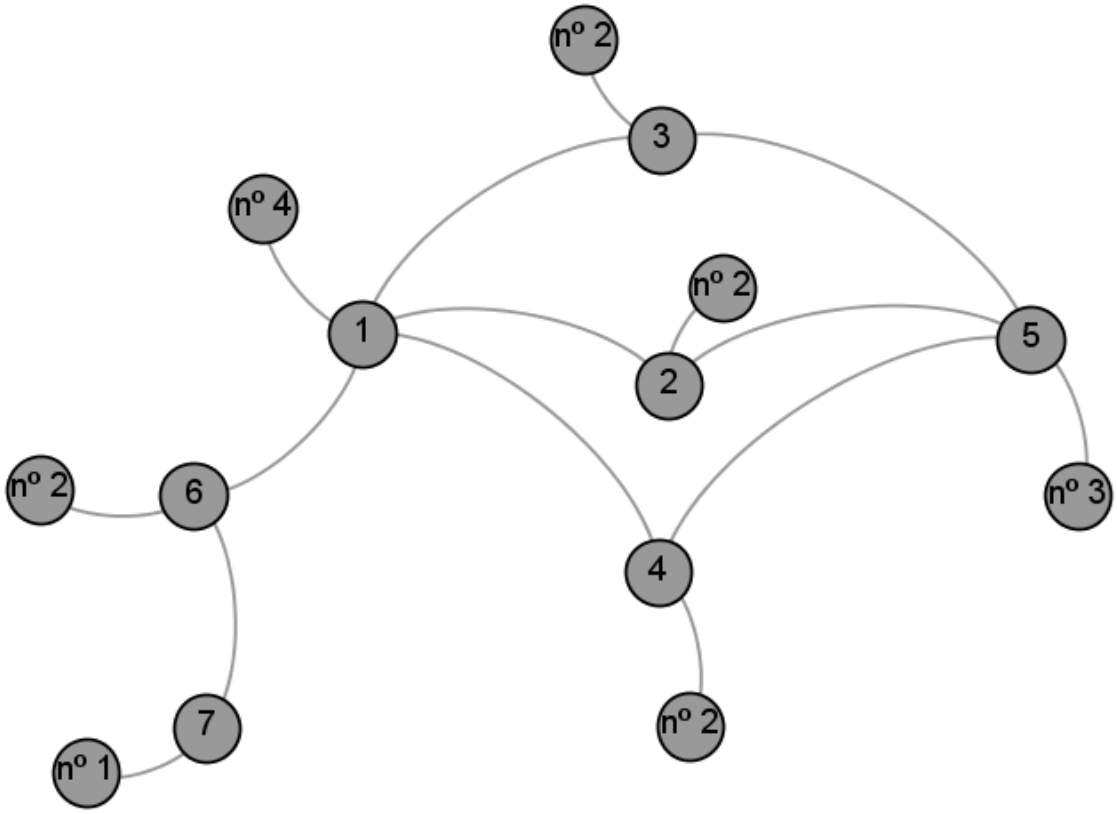

Figura 23 - Rede Exemplo Estendida para Facilitar a Predição de Links

Outro fator necessário para haver a predição, e que é diretamente impactante na modelagem do banco, é a necessidade de haver dois instantes temporais das entidades do domínio. A persistência de mais de um grafo (no caso da predição seria o de aplicação do algoritmo e de validação) torna-se inviável quando tratado grandes volumes de dados. Assim, assumindo-se essa requisição, modelam-se as arestas com propriedades características para representação temporal, como timestamps, ou cria-se uma propriedade booleana associada às arestas visando indicar a qual dos dois períodos aquela entidade está inserida, ou seja, uma propriedade visando identificar se a aresta pertence ao instante temporal de teste ou de validação. No caso desse trabalho adicionou-se a propriedade booleana “removed” para indicar se a aresta está logicamente removida (período de testes) ou não (período de validação).

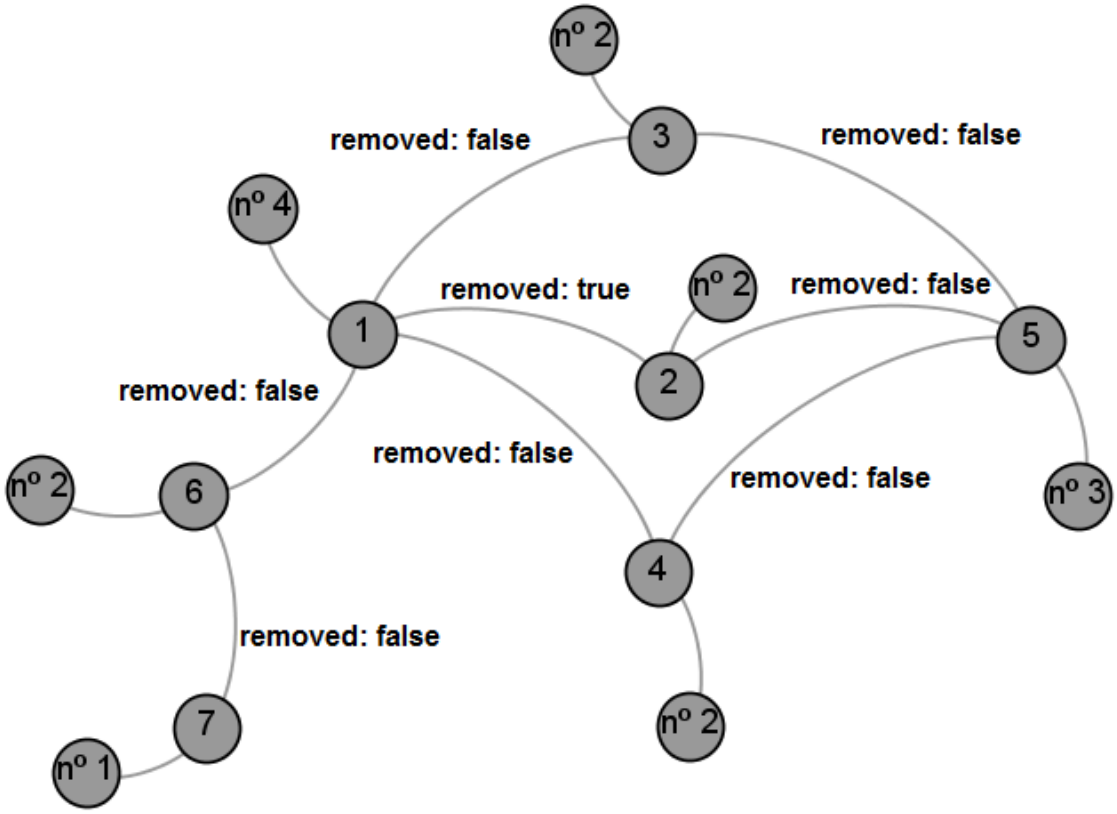

Figura 24 - Rede Exemplo Demonstrando a Modelagem do Período de Testes e Validação

4.7-Core API

O banco Neo4J apresenta uma boa variedade de interfaces para a recuperação de informações. São três formas de se relacionar com o banco sendo que todas possuem uma API em java. Essas interfaces apresentam diferenças em relação à linguagem de acesso ao banco, ao desempenho computacional e a forma de se realizar o acesso. Na Figura 25 representam-se estas APIs como uma pilha de blocos onde o mais acima se preza pela expressividade e o mais abaixo pela precisão.

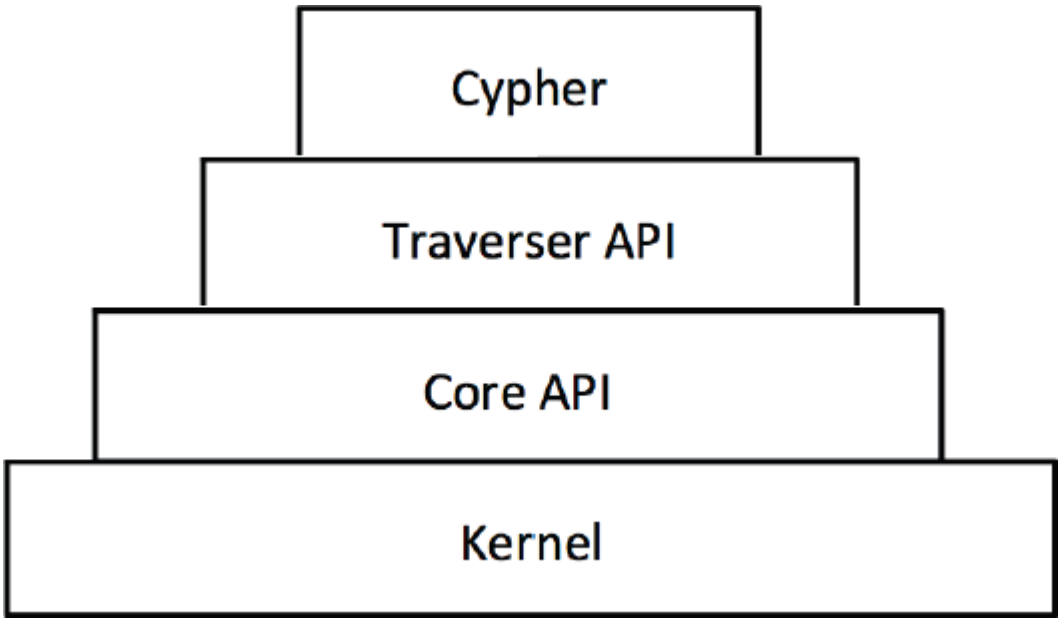

Figura 25 - Pirâmide de Frameworks do Neo4J

A primeira delas, a Cypher Query Language, é a linguagem de mais alto nível para se comunicar com o banco. Essa linguagem é a representação adaptada, para a forma de grafos, do SQL dos bancos de dados relacionais. Por estar em alto nível apresenta características interessantes ao desenvolvedor, como alta legibilidade, por exemplo, o que torna suas querys facilmente entendíveis mesmo por quem não é profundo conhecedor da linguagem. Porém esse nível de abstração vem com um alto custo de desempenho (Liben-Nowell et al. 2007), o retorno de resultados é em média muito mais lento que outras interfaces que acessam diretamente o domínio. Como esse projeto visa a maior eficiência possível para o cálculo de predição de links, escolheu-se a API de mais baixo nível possível. A mais próxima possível do banco, a Kernel, não está completamente disponível aos usuários, visto que apenas o tratamento de eventos nas entidades é passível de utilização. A interface Core contém os tipos primitivos do banco, como nó, relacionamentos e propriedades. Por se tratar de uma API de baixo nível, teoricamente, o banco não precisa avaliar nenhuma condição de pesquisa, assim a linguagem de programação consegue acessar diretamente os dados persistidos. Devido a essa abordagem o desempenho depende diretamente da qualidade da programação de acesso aos dados. A seguir demonstra-se uma simples implementação nesta API para demonstrar sua efetividade em se conectar ao banco.

4.8 Teste da API do Neo4J– Algoritmo Common Neighbors

Para exemplificar a utilização do banco de dados para a predição de links demonstrar-se-á a implementação do algoritmo Common Neighbors utilizando-se a Core API. Este algoritmo parte da simples ideia que dois nós aumentam sua probabilidade de estarem conectados conforme seu número de nós em comum aumenta. Implementou-se este da forma mais simplificada possível visando-se a didática dessa explicação e não a mais eficiente computacionalmente. Nota-se que o cálculo deste algoritmo é feito elevando-se a matriz simétrica de arestas de um determinado nó ao quadrado. Porém, visando evitar o alto custo computacional dessa abordagem, aqui iremos simplesmente percorrer os relacionamentos de dois nós no banco de dados e conferir se há elementos em comum.

```
public class CommonNeighbors implements Strategy {

    @Override
    public Map<String, Double> calculate(DataBase instance) {

        Map<String, Double> map = new HashMap<String, Double>();

        Iterable<Node> nodes = instance.getAllNodes();

        for (Node firstNode : nodes) {

        List<Node> firstNodeFriends = instance.neighborsOf(String.valueOf(firstNode.getId()));

        for (Node secondNode : nodes) {
            if (secondNode.getId() <= firstNode.getId()) {
            continue;
            }

        List<Node> secondNodeFriends = instance.neighborsOf(String.valueOf(secondNode.getId()));

        int commonFriends = CollectionsUtil.compare(firstNodeFriends, secondNodeFriends);

        map.put("(" + firstNode.getId() + " : " + secondNode.getId() + ")", (double) commonFriends);

        }

    }

    return map;
}
```

```
            @Override
       public List<Node> getAllNodes() {
              Transaction tx = getInstance().beginTx();
```

```
		List<Node> nodes = new ArrayList<Node>();

		try {

			Iterable<Node> allNodes = GlobalGraphOperati-
ons.at(getInstance()).getAllNodes();

			for (Node node : allNodes) {
				nodes.add(node);
			}

			tx.success();
		} finally {
			tx.finish();
		}

		return nodes;
	}

	@Override
	public List<Node> neighborsOf(String id) {

		Transaction tx = getInstance().beginTx();

		List<Node> friends = new ArrayList();

		Node node = getInstance().getNodeById(Long.valueOf(id));

		try {

			for (Relationship relationship : node.getRelationships()) {

				Node friend = relationship.getStartNode();

				if (friend.equals(node)) {

					friend = relationship.getEndNode();

				}

				friends.add(friend);

			}

			tx.success();
		} finally {
			tx.finish();
		}

		return friends;

	}
```

A resposta desse programa para o banco de dados inicializado no exemplo anterior daria como saída:

```
Creating Nodes ...

Creating Relationships ...

Creating Friends Property ...

{(3 : 6)=1.0, (3 : 7)=0.0, (1 : 6)=0.0, (3 : 5)=0.0, (4 : 5)=0.0, (2 : 7)=0.0, (6 : 7)=0.0, (4
: 6)=1.0, (2 : 6)=1.0, (2 : 5)=0.0, (4 : 7)=0.0, (1 : 5)=3.0, (1 : 7)=1.0, (1 : 4)=0.0, (3 :
4)=2.0, (5 : 6)=0.0, (1 : 2)=0.0, (1 : 3)=0.0, (2 : 3)=2.0, (5 : 7)=0.0, (2 : 4)=2.0}

Shutting down database ...
```

4.9 Tipos de Grafos

Os formatos de representação de grafos são os mais variados possíveis. O grande número de propriedades que um grafo pode possuir permitiu que, ao longo do tempo, fossem desenvolvidos diferentes tipos de formas de serem armazenados, conforme as necessidades de cada software. Cada tipo visa solucionar uma necessidade específica, desde representar uma matriz de adjacência até grafos que contém tipos complexos contendo informações como, peso de arestas e posicionamento do elemento (nós) na tela. Infelizmente essa situação criou uma impossibilidade de um software conseguir suportar todos os tipos possíveis de extensões, o que levou a escolha de formatos específicos para a utilização do programa.

A escolha dos formatos suportados baseou-se nas características suportadas por este e na disponibilidade de arquivos de grafos. O primeiro fator é de crucial importância ao sistema, pois um arquivo pobre em informações restringe as funcionalidades do sistema. As características mínimas necessárias pelas necessárias para o problema de Predição de Links são os tipos básicos dos dados da rede, nó e arestas, porém é interessante que haja o suporte a alguns atributos de redes como labels e pesos de arestas. O segundo fator poderá ditar o sucesso do programa uma vez que a dificuldade de conversão entre arquivos (basta observar o elevado nível de especificidade dos softwares que lidam com grafos) pode limitar a sua aplicabilidade. Esse critério de “popularidade” teve um alto peso na escolha dos formatos. Na Figura 26, identificou-se que poucos formatos não aceitam as duas requisições.

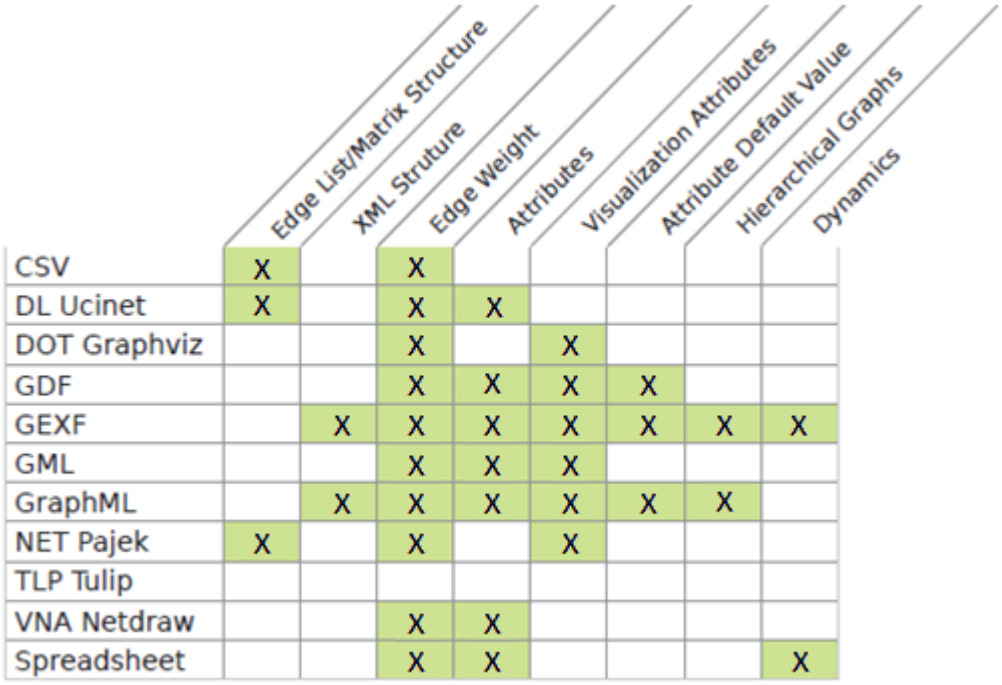

Figura 26 - Comparação dos Tipos de Formato de Grafos

Os tipos escolhidos para o programa suportar foram NET Pajek e GML. O tipo NET Pajek, desenvolvido em 1996, apresenta uma simplicidade e efetividade impar na representação de grafos. Esse formato disponibiliza a possibilidade de persistência das características básicas de um grafo, sendo perfeitamente ajustável ao problema de predição de links. Sua sobrevida histórica acabou resultando em uma alta disponibilidade de grafos, tornando seu suporte crucial para a sobrevida de qualquer programa da área de redes. Hoje, ele é suportado por quase todos os programas, incluindo o próprio Pajek, NodeXL, NetworkX e Grephi. A seguir apresenta-se o grafo modelado na seção *Modelagem* neste formato.

```
*Vertices     7
1 "1"
2 "2"
3 "3"
4 "4"
5 "5"
6 "6"
7 "7"
*Arcs
*Edges
1     2     1
1     3     1
1     4     1
```

```
1    6    1
2    5    1
3    5    1
4    5    1
6    7    1
```

O formato GML, ou Graph Modeling Language, é basicamente um arquivo em formato de texto com uma sintaxe muito amigável ao usuário. Contém também as principais necessidades desse projeto, como propriedades de nós e arestas e possui alta utilização dentre os programas comerciais na área. Todos os arquivos utilizados nesse trabalho foram adquiridos em formato GML. A seguir apresenta-se o mesmo grafo anterior nesse formato.

```
graph
[
 node
 [
          id 1
          label "Node A"
 ]
 node
 [
          id 2
          label "Node B"
 ]
 node
 [
          id 3
          label "Node C"
 ]
 node
 [
          id 4
          label "Node C"
 ]
 node
 [
          id 5
```

```
        label "Node C"
]
node
[
        id 6
        label "Node C"
]
node
[
        id 7
        label "Node C"
]
 edge
[
        source 1
        target 2
]
edge
[
        source 1
        target 3
]
edge
[
        source 1
        target 4
]
edge
[
        source 2
        target 5
]
edge
[
        source 3
```

```
        target 5
 ]
 edge
 [
        source 4
        target 5
 ]
 edge
 [
        source 1
       target 6
 ]
 edge
 [
        source 6
        target 7
 ]
]
```

4.10 Visualização

Para a camada gráfica escolheu-se o JavaFX, pois alem de possuir algumas características interessantes possui total compatibilidade com a linguagem utilizada nas outras camadas. Essa tecnologia basicamente empacota outras já desenvolvidas anteriormente para o próprio Java, como o Swing e a AWT, porém focando na interatividade e na interface gráfica para os usuários. A interface Canvas em especial representa uma vantagem na parte gráfica devido a seu desempenho desenhando elementos na tela, uma vez que os grafos apresentam grande quantidade de nós e ligações a serem exibidos.

A tela baseou-se na separação de camadas de informações ao usuário. A primeira camada de visualização define um BorderPane, Figura 27, artefato que possui as regiões de usabilidade padrão para o usuário, sendo aqui utilizado o topo para o cabeçalho do programa, à esquerda para menus e botões, a central para a visualização do grafo e a direita para a saída de informações.

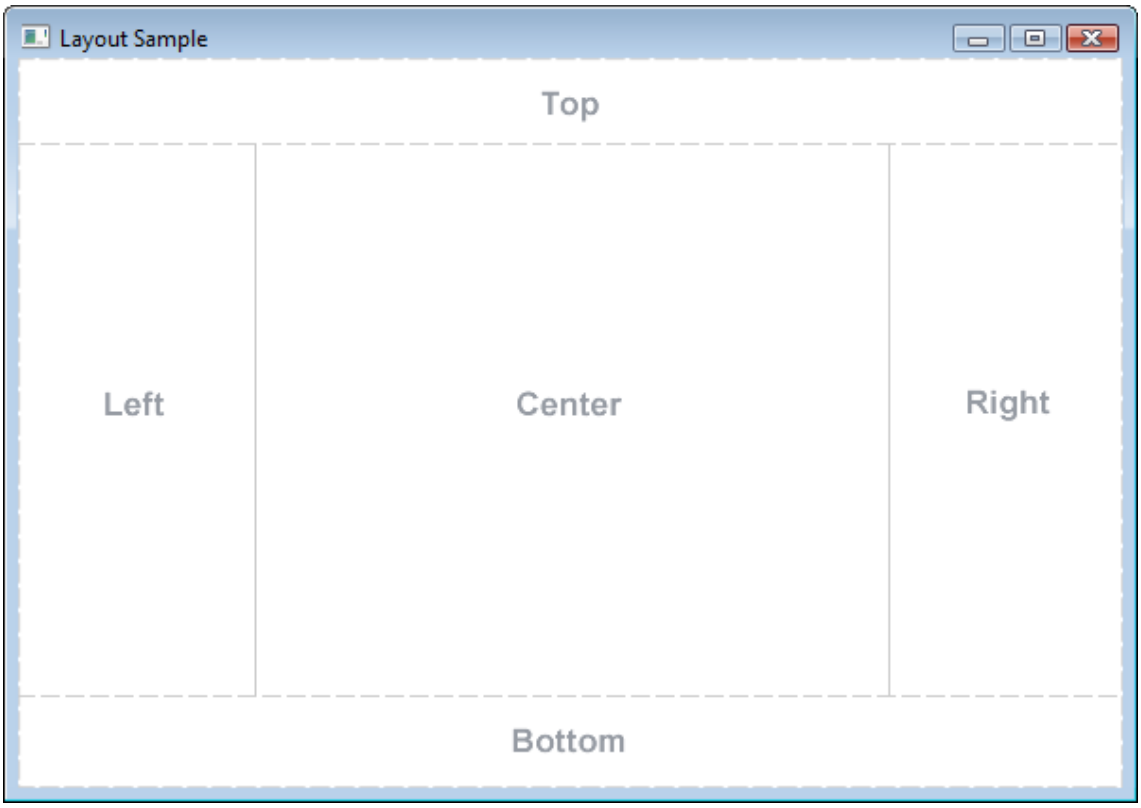

Figura 27 - Layout Padrão do Programa

Para a visualização do grafo, contido na parte central, criou-se um tipo complexo de nós para fazer a representação visual do objeto contido no banco de dados e instanciou-se um objeto linha entre os pontos centrais dos nós. A classe que representa visualmente o vértice foi desenvolvida para transformar as informações de posição, tamanho e texto contidas no domínio em informações visuais para o usuário. Essa classe baseou-se na implementação do StackPane, que permite a sobreposição de artefatos visuais; o círculo que representa um nó e um texto representando seu Label, além de possuir possibilidades de atribuir propriedades que contém informações a respeito do posicionamento espacial do objeto. Para a criação das linhas entre os elementos precisou-se percorrer todos os nós desenhados na tela, seus vizinhos e adicionar uma linha entre os pontos centrais destes. Os códigos dessas implementações estão listados em anexo.

5. PROGRAMA EM FUNCIONAMENTO

A tela em funcionamento pode ser vista na Figura 28.

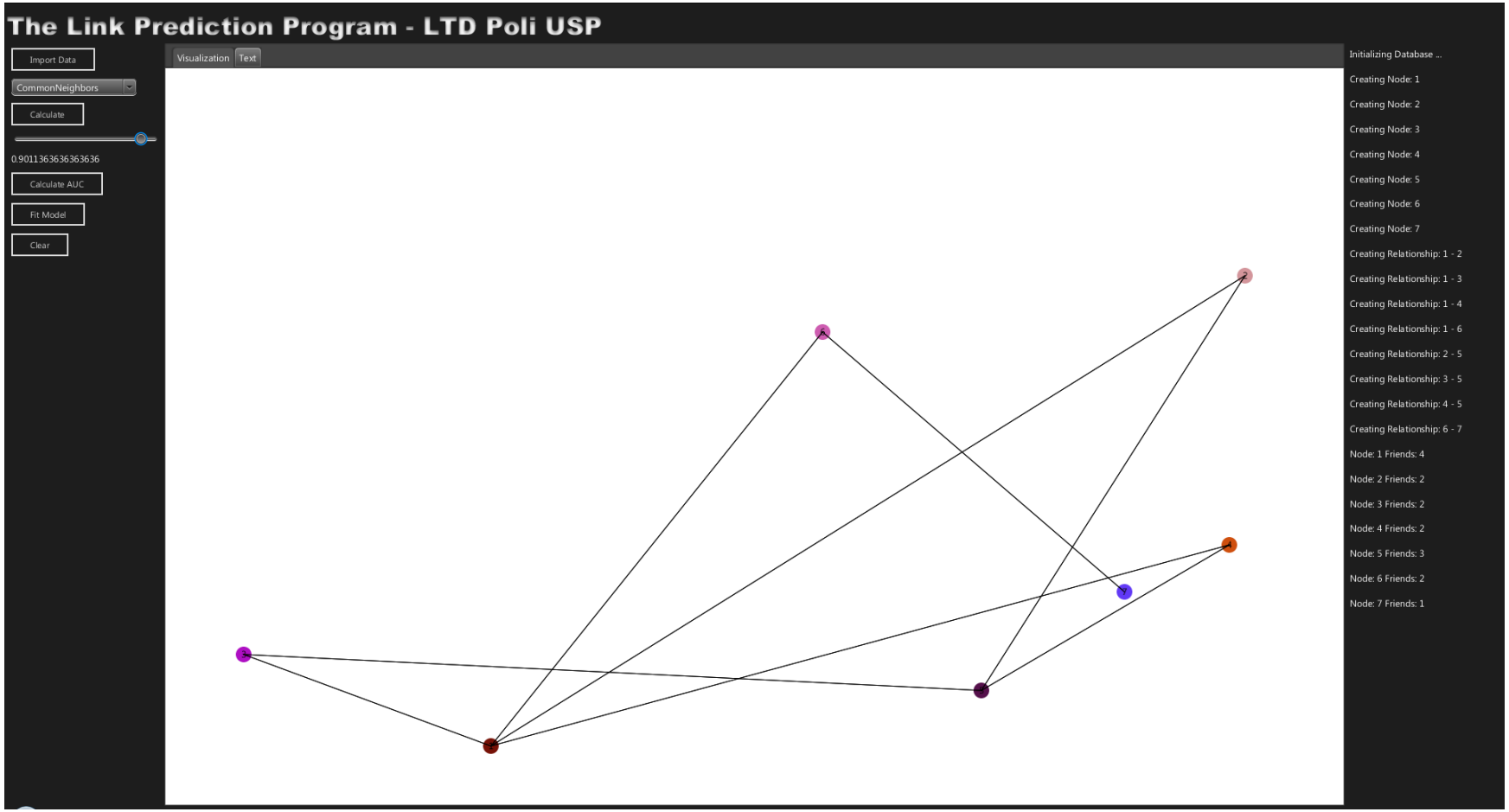
The Link Prediction Program - LTD Poli USP
Import Data
CommonNeighbors
Calculate
0.9011363636363636
Calculate AUC
Fit Model
Clear
Visualization
Text
Initializing Database ...
Creating Node: 1
Creating Node: 2
Creating Node: 3
Creating Node: 4
Creating Node: 5
Creating Node: 6
Creating Node: 7
Creating Relationship: 1 - 2
Creating Relationship: 1 - 3
Creating Relationship: 1 - 4
Creating Relationship: 1 - 6
Creating Relationship: 2 - 5
Creating Relationship: 3 - 5
Creating Relationship: 4 - 5
Creating Relationship: 6 - 7
Node: 1 Friends: 4
Node: 2 Friends: 2
Node: 3 Friends: 2
Node: 4 Friends: 2
Node: 5 Friends: 3
Node: 6 Friends: 2
Node: 7 Friends: 1

Figura 28 - IHM do Programa contendo a Rede Genérica

6. TRABALHOS FUTUROS

O presente projeto focou em alguns pontos do problema de predição de links, que podem ser ampliados futuramente. Primeiramente, utilizaram-se apenas algoritmos de predição local, o que beneficia o software no quesito velocidade, porém penaliza na questão da eficácia. Uma possível extensão seria estender o projeto para algoritmos de predição global, a fim de se tornar o programa mais versátil. Além disso, não foi utilizada nenhuma informação semântica dos grafos, como peso de arestas nos algoritmos preditores, podendo ser outro ponto de extensão. Finalmente, a ampliação e otimização da API desenvolvida para esse software é o caminho mais indicado a ser seguido de imediato.

7. CONCLUSÕES

Neste relatório apresentou-se a criação de um software capaz de automatizar a predição de links. O trabalho também construiu um framework capaz de utilizar poucas informações contidas no grafo e obter uma alta eficácia. Para o desenvolvimento do programa, demonstrou-se como solucionar problemas associados à predição de links, em particular a persistência e a visualização dos dados. Em relação à persistência, demonstrou-se como resolver o problema de armazenagem dos dados, utilizando-se de bancos especiais para grafos e como se fazer o acesso a esses dados de forma mais rápida possível. Em especial para pequenas quantias de dados, demonstrou-se como realizar a computação mantendo os dados em memória, aumentando a velocidade dos cálculos. A camada de visualização, por sua vez, exigiu que fossem utilizadas tecnologias gráficas recentes, pois a sua complexidade está na dificuldade em se exibir diferentes quantidades de dados. Esse problema foi resolvido utilizando duas técnicas: utilizando um componente capaz de imprimir na tela os grafos no seu formato original, e listando-se todos os tipos em tabelas estáticas. Para propor uma nova métrica, utilizou-se de conhecimentos prévios a respeito da capacidade preditiva de algoritmos que utilizam informações locais de maneira não supervisionada. A partir do entendimento de como funcionam os relacionamentos entre nós de uma rede, e de quais algorit-

mos desempenham melhor tentando extrair essas informações, propôs-se um método que aumentou significativamente a eficácia em relação a estes.

Portanto, esse trabalho consistiu em expandir as fronteiras da predição de links, criando um software de fácil utilização para qualquer usuário interessado no problema, e também aumentando a capacidade preditiva dos algoritmos existentes.

8. REFERÊNCIAS BIBLIOGRÁFICAS

Adamic, Lada A., Eytan Adar. "Friends and neighbors on the web." *Social networks* 25.3 (2003): 211-230.

Al Hasan, Mohammad, Mohammed J. Zaki. "A survey of link prediction in social networks." *Social Network Data Analytics*. Springer US, 2011. 243-275.

Barabási, Albert-László, et al. "Evolution of the social network of scientific collaborations." *Physica A: Statistical Mechanics and its Applications* 311.3 (2002): 590-614.

Bastian M., Heymann S., Jacomy M. Gephi: an open source software for exploring and manipulating networks.International AAAI *Conference on Weblogs and Social Media*. (2009).

Barabási, Albert-László, Réka Albert. "Emergence of scaling in random networks." *science* 286.5439 (1999): 509-512.

Brin, Sergey, Lawrence Page. "The anatomy of a large-scale hypertextual Web search engine." *Computer networks and ISDN systems* 30.1 (1998): 107-117.

Bu, Dongbo, et al. "Topological structure analysis of the protein–protein interaction network in budding yeast." *Nucleic Acids Research* 31.9 (2003): 2443-2450.

Cohen, Sheldon, et al. "Social ties and susceptibility to the common cold."*JAMA: the journal of the American Medical Association* 277.24 (1997): 1940-1944.

DeLong, Elizabeth R., David M. DeLong, Daniel L. Clarke-Pearson. "Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach." *Biometrics* (1988): 837-845.

Dominguez-Sal, David, P. Urbón-Bayes, Aleix Giménez-Vañó, Sergio Gómez-Villamor, Norbert Martínez-Bazán, Josep-Lluis Larriba-Pey. *"Survey of graph database performance on the hpc scalable graph analysis benchmark."* In*Web-Age Information Management*, pp. 37-48. Springer Berlin Heidelberg, 2010.

D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, S. M. Dawson, *Behavioral Ecology and Sociobiology* 54,(2003): 396-405

D. J. Watts, S. H. Strogatz, *Nature* 393 (1998): 440-442

Dudek, Gregory, Michael Jenkin, Evangelos Milios, David Wilkes. "Robotic exploration as graph construction." *Robotics and Automation, IEEE Transactions on* 7, no. 6 (1991): 859-865.

Erich, Gamma, Helm Richard, Johnson Ralph, Vlissides John. "Design patterns: elements of reusable object-oriented software." *Reading: Addison Wesley Publishing Company* (1995).

Fawcett, Tom. "ROC graphs: Notes and practical considerations for researchers." *Machine Learning* 31 (2004): 1-38.

Goh, Kwang-Il, et al. "The human disease network." *Proceedings of the National Academy of Sciences* 104.21 (2007): 8685-8690.

Goh K-I, Cusick ME, Valle D, Childs B, Vidal M, Barabási. A-L Proc Natl Acad Sci USA (2007): 104:8685-8690

Hand, David J., Robert J. Till. "A simple generalisation of the area under the ROC curve for multiple class classification problems." *Machine Learning* 45.2 (2001): 171-186.

Huang, Jin, Charles X. Ling. "Using AUC and accuracy in evaluating learning algorithms." *Knowledge and Data Engineering, IEEE Transactions on*17.3 (2005): 299-310.

Kossinets, Gueorgi, Duncan J. Watts. "Empirical analysis of an evolving social network." *Science* 311.5757 (2006): 88-90.

Leicht, E. A., Petter Holme, M. E. J. Newman. "Vertex similarity in networks." *Physical Review E* 73.2 (2006): 026120.

Liao, James C., et al. "Network component analysis: reconstruction of regulatory signals in biological systems." *Proceedings of the National Academy of Sciences* 100.26 (2003): 15522-15527.

Liben-Nowell, David, Jon Kleinberg. "The link-prediction problem for social networks." *Journal of the American society for information science and technology* 58.7 (2007): 1019-1031.

Lü, Linyuan, Tao Zhou. "Link prediction in complex networks: A survey."*Physica A: Statistical Mechanics and its Applications* 390.6 (2011): 1150-1170.

Lü, Linyuan, Tao Zhou. "Link prediction in weighted networks: The role of weak ties." *EPL (Europhysics Letters)* 89.1 (2010): 18001.

Newman, Mark EJ. "Clustering and preferential attachment in growing networks." *Physical Review E* 64.2 (2001): 025102.

Salton, Gerard, Michael J. McGill. "Introduction to modern information retrieval." (1986).

Sørensen, Thorvald. "A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons." *Biol. Skr.* 5 (1948): 1-34.

Schafer, Joseph L., John W. Graham. "Missing data: Our view of the state of the art." *Psychological methods* 7.2 (2002): 147-177.

Société Vaudoise des Sciences Naturelles. *Bulletin de la Société vaudoise des sciences naturelles*. Vol. 5. 1858.

Yu, Haiyuan, et al. "High-quality binary protein interaction map of the yeast interactome network." *Science* 322.5898 (2008): 104-110.

Ravasz, Erzsébet, et al. "Hierarchical organization of modularity in metabolic networks." *science* 297.5586 (2002): 1551-1555.

Sarwar, Badrul, et al. "Analysis of recommendation algorithms for e-commerce." *Proceedings of the 2nd ACM conference on Electronic commerce*. ACM, 2000.

Walter, Frank Edward, Stefano Battiston, Frank Schweitzer. "A model of a trust-based recommendation system on a social network." *Autonomous Agents and Multi-Agent Systems* 16.1 (2008): 57-74.

Vicknair, Chad, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen, Dawn Wilkins. "A comparison of a graph database and a relational database: a data provenance perspective." In *Proceedings of the 48th annual Southeast regional conference*, p. 42. ACM, 2010.

Zeggelink, Evelien. *Strangers into friends: The evolution of friendship networks using an individual oriented modeling approach*. Amsterdam: Thesis publishers, 1993.

Zhou, Tao, Linyuan Lü, Yi-Cheng Zhang. "Predicting missing links via local information." *The European Physical Journal B* 71.4 (2009): 623-630.

Zhu, Jianhan, Jun Hong, John G. Hughes. "Using Markov models for web site link prediction." *Proceedings of the thirteenth ACM conference on Hypertext and hypermedia*. ACM, 2002.

ANEXO A

1. LINK PREDICTION API

O problema de predição de links hoje apresenta apenas poucos algoritmos implementados em softwares diferentes. O código desenvolvido inclui não somente os algoritmos contidos na literatura, mas também um pacote que seja capaz de avaliar a efetividade desses.

Para a implementação do algoritmo foi utilizada o padrão de projeto *Strategy*, do influente livro Design Patterns, de Erich et al. 1995. A ideia básica por trás desse padrão é que o usuário apenas instancie o objeto no momento de sua construção e chame a sua execução de maneira genérica. O padrão aplicado ao projeto pode ser observado no diagrama UML da Figura 29.

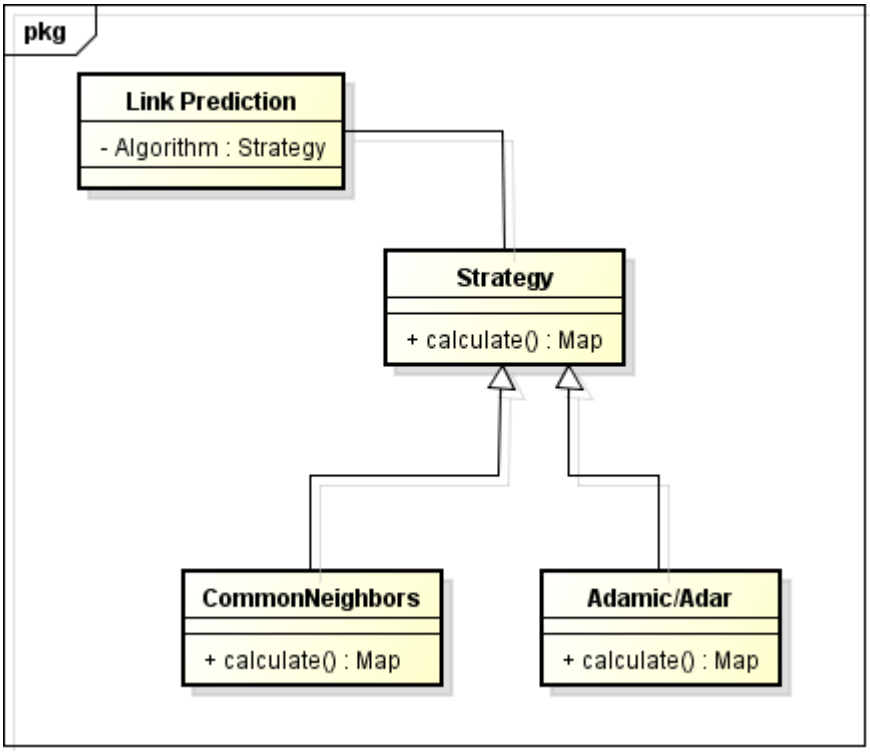

Figura 29 – Padrão Strategy Aplicado ao Software.

Em particular para esse projeto instanciou-se todos os algoritmos previamente em uma enumeração para facilitar ao usuário a chamada da classe.

```
package linkprediction.math.predictors;

/**
 * Enumeration that instantiate all local predictors.
 */
public enum Algorithm {
	CommonNeighbors(new CommonNeighbors()), AdamicAdar(new AdamicAdar()), JaccardsCoefficient(new JaccardsCoefficient()), SaltonIndex(new SaltonIndex()), LeichHolmeNewman(new LeichHolmeNewman()), Sorensen(new Sorensen()), PreferentialAttachment(   new PreferentialAttachment()), HubPromoted(new HubPromoted()), HubDepressed(new HubDepressed()), ResourceAllocation(new ResourceAllocation());

	private Strategy instance;

	private Algorithm(Strategy instance) {
		this.instance = instance;
	}

	public Strategy getInstance() {
		return instance;
	}

}
```

A chamada para a utilização do algoritmo fica facilitada através dessa abordagem, pois cabe ao usuário apenas a escolha do algoritmo. Assim, toda a implementação e inteligência associada à programação ficam transparentes ao usuário da API.

```
LinkPrediction linkPrediction = new LinkPrediction(Algorithm.CommonNeighbors);
Map<String, Double> map = linkPrediction.calculate(dataBase);
```

Porém, o cálculo das arestas preditas exige que os dados estejam armazenados em algum lugar da memória para que este acesse e faça a computação. A dificuldade é tornar a implementação genérica o suficiente para se adaptar a qualquer estrutura de dados. Nesse trabalho, por exemplo, utilizaram-se duas abordagens para manter os dados, uma em memória através da API jgrapht e uma em banco através do framework Neo4J. Para solucionar esse problema criou-se uma interface em que especifica os métodos necessários para a computação da predição de links, e deixa ao usuário a tarefa de implementá-los.

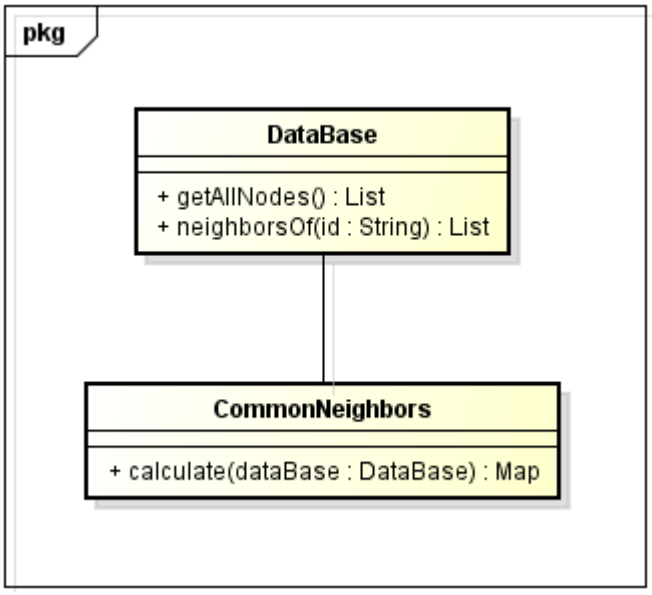

Figura 30 – Abstração das Operações necessárias a Predição de Links.

Verifica-se que para a predição de links através de informações locais basta receber a lista de todos os nós e poder acessar os nós que se relacionam a um determinado. Essa tarefa não será tão simples quando o programa for estendido a algoritmos de predição através de informações globais.

```
package linkprediction.memory;

import java.util.*;

public interface DataBase {

	/**
	 * Retrieve a list with all nodes.
	 */
	public List<Object> getAllNodes();

	/**
	 * Retrieve a list of all neighbors of a specified node.
	 */
	public List<Object> neighborsOf (String id);
}


public class CommonNeighbors implements Strategy {

	@Override
	public Map<String, Double> calculate(DataBase instance) {
	}
```

```
}
```

Outro problema associado à predição de links é comparar os nós preditos com os armazenados em memória. Assim como não se sabe a estrutura de dados utilizada pelo usuário, não se pode prever qual serão os tipos dos elementos do grafo. Assim, uma abordagem específica deve ser utilizada para esse problema, utilizando-se ferramentas avançadas do Java, como reflexão e coleções. A seguir exemplifica-se a utilização de reflexão para adquirir campos de uma classe.

```
	public static Collection<Field> getDeepDeclaredFields(Class c) {
		if (_reflectedFields.containsKey(c)) {
			return _reflectedFields.get(c);
		}
		Collection<Field> fields = new ArrayList<Field>();
		Class curr = c;

		while (curr != null) {
			try {
				Field[] local = curr.getDeclaredFields();

				for (Field field : local) {
					if (!field.isAccessible()) {
						try {
							field.setAccessible(true);
						} catch (Exception ignored) {
						}
					}

					int modifiers = field.getModifiers();
					if (!Modifier.isStatic(modifiers) && !fi-
eld.getName().startsWith("this$")
							&& !Modifier.isTransient(modifiers)) {
						fields.add(field);
					}
				}
			} catch (ThreadDeath t) {
				throw t;
			} catch (Throwable ignored) {
			}

			curr = curr.getSuperclass();
		}
		_reflectedFields.put(c, fields);
		return fields;
	}
```

2. IMPLEMENTAÇÕES

Common Neighbors

```
1.  package linkprediction.math.predictors;
2.
3.  import java.util.HashMap;
4.  import java.util.List;
5.  import java.util.Map;
6.
7.  import linkprediction.memory.DataBase;
8.
9.  import org.neo4j.graphdb.Node;
10.
11. import util.CollectionsUtil;
12.
13. public class CommonNeighbors implements Strategy {
14.
15.     @Override
16.     public Map<String, Double> calculate(DataBase instance) {
17.
18.             Map<String, Double> map = new HashMap<String, Double>();
19.
20.             Iterable<Node> nodes = instance.getAllNodes();
21.
22.             for (Node firstNode : nodes) {
23.                     if (firstNode.getId() == 0) {
24.                             continue;
25.                     }
26.
27.                     List<Node> firstNodeFriends = instan-
    ce.neighborsOf(String.valueOf(firstNode.getId()));
28.
29.                     for (Node secondNode : nodes) {
30.                             if (secondNode.getId() <= firstNode.getId()) {
31.                                     continue;
32.                             }
33.
34.                             List<Node> secondNodeFriends = instan-
    ce.neighborsOf(String.valueOf(secondNode.getId()));
35.
36.                             double commonFriends = CollectionsU-
    til.compare(firstNodeFriends, secondNodeFriends).size();
37.
38.                             map.put("(" + firstNode.getId() + " : " + secondNo-
    de.getId() + ")", commonFriends);
39.
40.                     }
41.
42.             }
43.
44.             return map;
45.     }
46.
```

```
47. }
```

Database

Interface

```
package linkprediction.memory;

import java.util.List;

import org.neo4j.graphdb.Node;


public interface DataBase {

	public List<Node> getAllNodes();

	public List<Node> neighborsOf(String id);

}
```

Banco em Disco Rígido

```
package linkprediction.memory.database;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Set;

import linkprediction.memory.DataBase;
import linkprediction.view.ProgramView;

import org.jgrapht.Graph;
import org.jgrapht.graph.DefaultEdge;
import org.neo4j.graphdb.GraphDatabaseService;
import org.neo4j.graphdb.Node;
import org.neo4j.graphdb.Relationship;
import org.neo4j.graphdb.RelationshipType;
import org.neo4j.graphdb.Transaction;
import org.neo4j.graphdb.factory.GraphDatabaseFactory;
import org.neo4j.kernel.impl.util.FileUtils;
import org.neo4j.tooling.GlobalGraphOperations;

import util.DatabaseUtil;
import util.GraphUtil;

public class GraphDataBase implements DataBase {

	private static GraphDatabaseService database;

	public static int numberOfNodes = -1;
```

```
	public static GraphDatabaseService getInstance() {
		if (database == null) {
			ProgramView.write("Initializing Database ...");
			database = new GraphDatabaseFac-
tory().newEmbeddedDatabase(DatabaseUtil.DB_PATH);
			registerShutdownHook();
		}

		return database;
	}

	public GraphDataBase() {
		clearDb();
	}

	public void createDb(Graph<String, DefaultEdge> graph) {
		clearDb();

		Transaction tx = getInstance().beginTx();

		try {

			defineCorrectionOfIdConstant(graph.vertexSet().iterator().next());

			createNodes(graph.vertexSet());

			createRelationship(graph.edgeSet());

			createRelationshipRemovedProperty(graph.edgeSet());

			createNodesFriendsProperty(graph.vertexSet());

			tx.success();
		} finally {
			tx.finish();
		}

	}

	private void createRelationshipRemovedProperty(Set<DefaultEdge> edgeSet) {
		Iterable<Relationship> relationships = GlobalGraphOperati-
ons.at(getInstance()).getAllRelationships();

		for (Relationship relationship : relationships) {
			relationship.setProperty(DatabaseUtil.REMOVED_PROPERTY, false);
		}

	}

	private void defineCorrectionOfIdConstant(String next) {
		if (next.equals("0")) {
			DatabaseUtil.CONSTANT_CORRECTION = 1;
		} else {
			DatabaseUtil.CONSTANT_CORRECTION = 0;
		}
	}

	private void createNodesFriendsProperty(Set<String> vertexSet) {
		System.out.println();
```

```
                System.out.println("Creating Friends Property");
                for (String vertex : vertexSet) {
                        Node node = getNodeById(vertex);

                        ArrayList<Node> relationshipNodes = DatabaseU-
til.getRelationshipNodes(node);

                        node.setProperty(DatabaseUtil.NUMBER_FRIENDS_PROPERTY, relationshipNo-
des.size());

                }
        }

        @SuppressWarnings("unused")
        private void createRelationship(Set<DefaultEdge> edgeSet) {
                System.out.println();
                System.out.println("Creating Relationships");

                ArrayList<ArrayList<String>> edges = GraphUtil.getEdgeArray(edgeSet);

                for (ArrayList<String> edge : edges) {
                        Node firstNode = getNodeById(edge.get(0));
                        Node secondNode = getNodeById(edge.get(1));

                        Relationship relationship = firstNode.createRelationshipTo(secondNode,
RelTypes.KNOWS);

                }
        }

        /**
         * @param nodeId
         * @return
         */
        private Node getNodeById(String nodeId) {
                return getInstance().getNodeById(Integer.parseInt(nodeId) + DatabaseU-
til.CONSTANT_CORRECTION);
        }

        @SuppressWarnings("unused")
        private void createNodes(Set<String> vertexSet) {
                System.out.println();
                System.out.println("Creating Nodes ...");

                for (String vertex : vertexSet) {
                        Node node = getInstance().createNode();
                }
        }

        public void shutDown() {
                System.out.println();
                System.out.println("Shutting down database ...");

                getInstance().shutdown();
        }

        private void clearDb() {
                try {
                        FileUtils.deleteRecursively(new File(DatabaseUtil.DB_PATH));
```

```
		} catch (IOException e) {
			throw new RuntimeException(e);
		}
	}

	private static void registerShutdownHook() {
		Runtime.getRuntime().addShutdownHook(new Thread() {
			@Override
			public void run() {
				database.shutdown();
			}
		});
	}

	private static enum RelTypes implements RelationshipType {
		KNOWS
	}

	@Override
	public void createNode(String label) {

		Transaction tx = getInstance().beginTx();

		try {
			Node node = getInstance().createNode();
			node.setProperty(DatabaseUtil.NODE_LABEL, label);
			ProgramView.write("Creating Node: " + node.getId());

			tx.success();
		} finally {
			tx.finish();
		}

	}

	@Override
	public void createRelationship(String firstNodeId, String secondNodeId) {

		Transaction tx = getInstance().beginTx();

		try {
			Node firstNode = getNodeById(firstNodeId);
			Node secondNode = getNodeById(secondNodeId);

			Relationship relationship = firstNode.createRelationshipTo(secondNode,
RelTypes.KNOWS);
			relationship.setProperty(DatabaseUtil.REMOVED_PROPERTY, false);

			ProgramView.write("Creating Relationship: " + firstNode.getId() + " - " +
secondNode.getId());

			tx.success();
		} finally {
			tx.finish();
		}

	}

	@Override
```

```
	public void createNodesFriendsProperty() {

		Transaction tx = getInstance().beginTx();

		try {
			Iterable<Node> nodes = GlobalGraphOperati-
ons.at(getInstance()).getAllNodes();

			for (Node node : nodes) {
				if (node.getId() == 0) {
					continue;
				}

				ArrayList<Node> relationshipNodes = DatabaseU-
til.getRelationshipNodes(node);

				int size = relationshipNodes.size();
				node.setProperty(DatabaseUtil.NUMBER_FRIENDS_PROPERTY, size);

				ProgramView.write("Node: " + node.getId() + " Friends: " + size);
			}

			tx.success();
		} finally {
			tx.finish();
		}

	}

	@Override
	public List<Node> getAllNodes() {
		Transaction tx = getInstance().beginTx();

		List<Node> nodes = new ArrayList<Node>();

		try {

			Iterable<Node> allNodes = GlobalGraphOperati-
ons.at(getInstance()).getAllNodes();

			for (Node node : allNodes) {
				if (node.getId() == 0) {
					continue;
				}

				nodes.add(node);

			}

			tx.success();
		} finally {
			tx.finish();
		}

		return nodes;
	}

	@Override
	public List<Node> neighborsOf(String id) {
```

```
		Transaction tx = getInstance().beginTx();

		List<Node> friends = new ArrayList<Node>();

		Node node = getInstance().getNodeById(Long.valueOf(id));

		try {

			for (Relationship relationship : node.getRelationships()) {

				Node friend = relationship.getStartNode();

				if (friend.equals(node)) {

					friend = relationship.getEndNode();

				}

				if (friend.getId() == 0) {

					continue;
				}

				friends.add(friend);

			}

			tx.success();
		} finally {
			tx.finish();
		}

		return friends;

	}

	@Override
	public void clear() {

		Transaction tx = getInstance().beginTx();

		try {
			// Deleting All Relationships
			Iterable<Relationship> allRelationships = GlobalGraphOperati-
ons.at(getInstance()).getAllRelationships();
			for (Relationship relationship : allRelationships) {
				ProgramView.write("Deleting Relationship: " + relation-
ship.getEndNode() + " "
						+ relationship.getStartNode());
				relationship.delete();
			}

			// Deleting All Nodes
			Iterable<Node> allNodes = GlobalGraphOperati-
ons.at(getInstance()).getAllNodes();
			for (Node node : allNodes) {
				numberOfNodes++;
				ProgramView.write("Deleting Nodes: " + node.getId());
```

```
                        node.delete();
                }

                tx.success();
        } finally {
                tx.finish();
        }

    }
}
```

Banco de Dados em Memória

```
package linkprediction.memory.cache;

import java.util.List;
import java.util.Set;

import linkprediction.memory.DataBase;

import org.jgrapht.Graph;
import org.jgrapht.Graphs;
import org.jgrapht.graph.DefaultEdge;
import org.jgrapht.graph.SimpleGraph;

public class CacheDataBase implements DataBase {

    private static Graph<String, DefaultEdge> instance = new SimpleGraph<String, DefaultEd-
ge>(DefaultEdge.class);

    public static Graph<String, DefaultEdge> getInstance() {
        return instance;
    }

    @Override
    public void createNode(String label) {
        instance.addVertex(label);
    }

    @Override
    public void createRelationship(String firstNodeId, String secondNodeId) {
        instance.addEdge(firstNodeId, secondNodeId);
    }

    @Override
    public void createNodesFriendsProperty() {
        Set<String> firstNode = instance.vertexSet();
        Set<String> secondNode = instance.vertexSet();

        for (String first : firstNode) {
            for (String second : secondNode) {
                Set<DefaultEdge> edges = instance.getAllEdges(first, second);
                int size = edges.size();
```

```
                        instance.addVertex(first + second);
                        instance.addEdge(first, String.valueOf(size));
                    }
                }

        }

        @Override
        public List getAllNodes() {
                return (List) instance.edgeSet();
        }

        @Override
        public List neighborsOf(String id) {
                List neighbors = Graphs.neighborListOf(instance, id);
                return neighbors;
        }

        @Override
        public void clear() {
                instance = new SimpleGraph<String, DefaultEdge>(DefaultEdge.class);
        }

}
```