MONIQUE VAZ VIEIRA

BUSCA EFICIENTE EM REDES SOCIAIS

Belo Horizonte
30 de março de 2007

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

BUSCA EFICIENTE EM REDES SOCIAIS

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

MONIQUE VAZ VIEIRA

Belo Horizonte
30 de março de 2007

Resumo

Redes sociais são utilizadas para a interação entre as pessoas. Uma das principais operações de uma rede social é a busca por um usuário. Enquanto máquinas de busca utilizam sinais baseados em sua estrutura de enlaces (*links*), como *Pagerank* e autoridade, para melhorar a qualidade de seus resultados, em uma rede social, esses sinais não são apropriados. Nesse caso, um sinal alternativo para melhorar a busca por uma pessoa são os relacionamentos de amizade definidos na rede social. Para ilustrar, se João procura por Maria, uma boa função de ordenação daria maior peso para as Marias mais próximas do círculo de relacionamento de João. Entretanto, se o grafo de relacionamentos é grande, a computação eficiente dessas distâncias deixa de ser um problema trivial. Diante desse problema, propomos um algoritmo baseado em sementes que aproxima as distâncias de maneira eficiente, e pode oferecer ganhos nos tempos de execução no Orkut de até três ordens de grandeza com relação à solução força bruta, mantendo precisões acima de 70%. Reduzindo o ganho nos tempos de execução para duas ordens de grandeza, a precisão dos resultados ultrapassa 90%. Esses resultados mostram que é possível obter excelentes ganhos de desempenho na computação de distâncias de relacionamento em redes sociais - um sinal crucial para a busca - dentro de margens de erro aceitáveis, o que viabiliza a utilização desse importante sinal em redes sociais de grande porte.

Abstract

Search-engines use link-based signals, like pagerank ou authority, to improve the quality of the results. However, in other scenarios where these signals are either not appropriate or impossible to calculate, some other form of trust signal must be used. For instance, in the case of social networks, one alternative signal to improve a search for a given person are friends relationships. To illustrate, if John is looking for Maria, a good ranking function would favor the Maria's that are closer to John. However, if the relationships graph is large, computing these distance efficiently is non-trivial. To overcome this, we propose a seeds-based approximation algortihm that can speed up execution times on the Orkut social network by three orders of magnitude with respect to the brute force solution, while keeping the approximation error on the ranking smaller than 30%. By reducing the speedup to two orders of magnitude, we are able to attain approximation errors smaller than 12%. These results show that great speed up can be attained for computing friendship distances in social networks - a crucial signal for search ranking - within acceptable error margins.

Aos meus Pais

Sumário

1 Introdução **1**

1.1 Caracterização do Problema 3

1.2 A Solução 4

1.3 Estrutura da Dissertação 5

2 Trabalhos Relacionados **6**

3 Evolução da *Web* **8**

3.1 História da *Web* 8

3.2 Estado atual da *Web* 9

4 Recuperação de Informação na *Web* **15**

4.1 Ordenação de Respostas 17

4.1.1 Modelo Vetorial 17

4.1.2 Ordenação de Respostas na *Web* 19

4.2 Avaliação 22

5 A solução e resultados **24**

5.1 Possíveis soluções 24

5.1.1 Solução 1: Pré-computação de todas as distâncias 24

5.1.2 Solução 2: Cálculo de distâncias em tempo de consulta 26

5.1.3 Solução 3: Pré-computação de listas de amigos 26

5.2 Solução Baseada em Sementes Aleatórias 27

5.2.1 Índices que Representam a Estrutura do Grafo 27

5.2.2 Anotações Baseadas em Sementes Aleatórias 28

5.2.3 Computação dos Vetores de Distâncias 29

5.2.4 *MapReduce* 30

5.2.5 Algoritmo para Computação dos Vetores de Distâncias 32

5.2.6 Função de Similaridade 34

5.2.7 Análise de Complexidade de Nossa Solução 36

5.3 Experimentação 38
5.3.1 Ambiente de Experimentação 38
5.3.2 Algoritmos avaliados 39
5.3.3 Requisitos de Espaço 39
5.3.4 Resultados 41
5.3.5 Análise dos Resultados 45

6 Conclusões e trabalhos futuros **46**

Referências Bibliográficas **48**

Lista de Figuras

1.1 Grafo de relacionamentos: John está a uma distância 1 de 'Maria A.',a uma distância 2 de 'Maria B.', e a uma distância 3 de 'Maria C.' 3

4.1 Sistema de recuperação de informação *Web*. 16

5.1 Grafo com 3 sementes pré-selecionadas: S1, S2 e S3. 28
5.2 *Cluster* de 128 máquinas: processadores de 2GHz x86-64, rede ethernet 1 Gbps, 40M/128 perfis de usuários por máquina. 39

Lista de Tabelas

5.1 Espaço necessário para armazenar as distâncias entre todos os pares de usuários do grafo. Os cálculos levam em conta que cada distância pode ser armazenada em 4 bits. 25
5.2 Espaço necessário para armazenar distâncias menores ou iguais a 3 entre todos os pares de usuários do grafo. Os cálculos levam em conta que cada distância pode ser armazenada em apenas 2 bits. 25
5.3 Requisito de espaço para diferentes números de sementes. 40
5.4 Exemplos de listas de resultados ordenados. 42
5.5 Precisão em 10 média. 42
5.6 Lista de resultados ordenados para solução de teste modificada. 43
5.7 Precisão generalizada média em 1, 5 e 10 resultados. 44
5.8 Tempo médio de execução por consulta para todas as soluções. 44
5.9 Tempo de execução para o cálculo dos vetores de distâncias. 45

Capítulo 1

Introdução

Redes sociais ou *sites de relacionamento* são serviços *Web* que buscam conectar indivíduos e mediar a comunicação entre eles, permitindo assim a interação social de seus membros. Em uma rede social, cada usuário cria seu próprio perfil, onde declara suas informações pessoais, preferências e interesses. Um usuário pode ainda compartilhar fotos, vídeos e músicas com outro usuários, e participar de comunidades sobre assuntos de seu interesse. Além disso, cada usuário estabelece relacionamentos de amizade com outros usuários do sistema. Tais relacionamentos podem ser representados como arestas em um grafo cujos vértices são os próprios usuários.

A flexibilidade e a conveniência providas por essa forma de comunicação, intensificadas pelo baixo custo de acesso e pela liberdade de publicação de idéias e opiniões, levaram a uma rápida proliferação de plataformas de redes sociais de grande porte. Exemplos ilustrativos são MySpace[1] com cerca de 155 milhões de usuários, Orkut[2] com 41 milhões de usuários e Facebook[3] com 17 milhões de usuários[4].

Em uma rede social com um grande número de usuários, uma operação comum é a busca por um usuário. Afinal, as pessoas utilizam redes sociais para interagir e compartilhar experiências umas com as outras. Na verdade, um dos casos de uso mais comuns em redes sociais é reencontrar amigos antigos e fazer novos amigos, o que torna a busca por usuários de importância crítica para uma rede social. Contudo, busca por usuários é um problema com desafios particulares, uma vez que os sinais disponíveis em uma rede social são completamente distintos daqueles disponíveis na *Web*. Ao invés de apontadores (*links*) e texto, temos agora relacionamentos de amizade e preferências declaradas pelos usuários. Isto é, busca por usuários em uma rede social é um problema

[1] http://www.myspace.com

[2] http://www.orkut.com

[3] http://www.facebook.com

[4] Fonte: http://en.wikipedia.org/wiki/List_of_social_networking_websites - acessado em 17 fevereiro de 2007

distinto e importante cuja solução discutimos neste trabalho.

Para ilustrar as dificuldades inerentes ao problema, consideramos uma solução baseada em um simples casamento de texto que, dado um nome de pessoa encontra usuários com aquele nome no *site* de relacionamentos Orkut ([32]). Para isso, selecionamos 750 consultas dentre as milhões de consultas submetidas ao Orkut e contamos o número de perfis de usuários cujos nomes declarados correspondiam exatamente à consulta. Por exemplo, para a consulta 'Maria', contamos quantos usuários declararam seus nomes como 'Maria'. A média de resultados é 48, o que significa que utilizando uma técnica que faça apenas casamento de texto, os 48 primeiros resultados, tipicamente distribuídos ao longo de 5 páginas, seriam retornados sem qualquer critério de ordenação. Se formos além e considerarmos também os casamentos parciais entre a consulta e o nome declarado no perfil, por exemplo, perfis com o nome 'Maria Antônia' para a mesma consulta 'Maria', a média de resultados sobe para 6.034 por consulta. Quais perfis dentre tantos devem então ser mostrados para o usuário?

Um dos desafios para o desenvolvimento de bons sistemas de busca em redes sociais se deve ao fato da noção de relevância ser completamente distinta daquela em busca *Web* e não estar claramente definida. Um estudo recente sobre o Facebook [21] sugere que interações em uma rede social tendem a seguir os mesmos padrões das interações na vida real, o que significa que as pessoas tendem a interagir com pessoas que já conhecem. Assim, assumiremos que, quando um usuário procura por um nome, é mais provável que fique satisfeito ao ver pessoas que estão mais próximas dele em seu círculo de relacionamentos do que ao ver pessoas completamente desconhecidas. Ou seja, consideraremos que a estrutura de relacionamentos de uma rede social pode ser utilizada pelo sistema de busca para melhorar a precisão dos resultados de consultas que procuram por pessoas.

Para ilustrar, considere o grafo de relacionamentos na Figura 1.1, onde os vértices representam usuários de uma rede social e as arestas representam relacionamentos de amizade entre os usuários. Assuma que 'John' é um usuário conectado em nossa rede social que especificou a consulta 'Maria'. Existem três usuários distintos cujos nomes contêm o termo 'Maria'. Destes, 'Maria A.' é amiga de Jonh, 'Maria B.' é amiga de um dos amigos de John, e 'Maria C.' é um usuário mais distante, separada de John por três níveis. Argumentamos que qualquer função de ordenação de repostas (*ranking*) deveria dar um peso maior para 'Maria A.' e depois para 'Maria B.'. A função pode levar em conta também outros sinais tais como interesses em comum entre John e cada uma das Marias, mas a distância do relacionamento entre John e cada Maria é um sinal crítico para a função de ordenação de respostas e não deve ser ignorado.

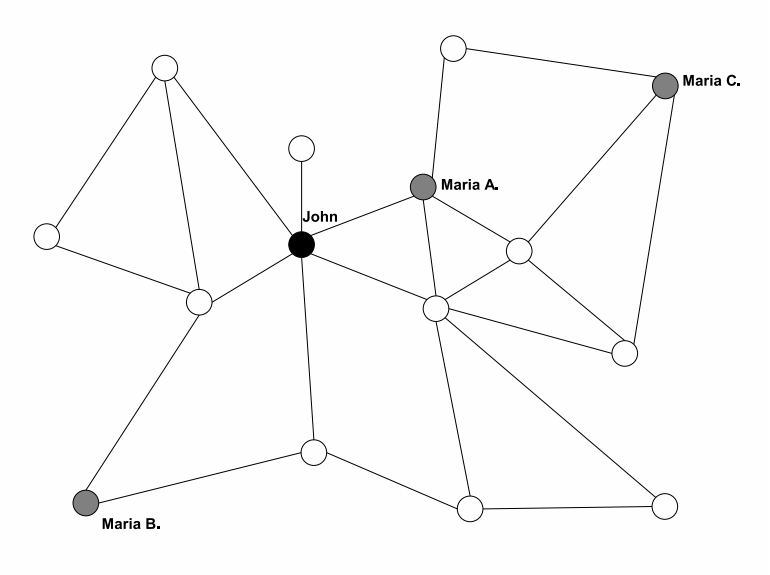

Figura 1.1: Grafo de relacionamentos: John está a uma distância 1 de 'Maria A.',a uma distância 2 de 'Maria B.', e a uma distância 3 de 'Maria C.'

1.1 Caracterização do Problema

Mostramos que as distâncias de relacionamento são essenciais em uma função de ordenação de respostas para busca de usuários em redes sociais. Entretanto, quando o número de usuários de um rede social é muito grande, computar as distâncias de relacionamento se torna um problema não trivial e é exatamente este problema que abordaremos neste trabalho. Dessa forma, definimos nosso problema como se segue.

Definição 1 *Seja u_i um usuário de uma rede social que inclui um grafo de relacionamentos entre usuários. Seja q uma consulta feita pelo usuário u_i especificando um nome de usuário. Considere que o número de usuários recuperados pela consulta é k. Nosso problema consiste em ordenar os k usuários de acordo com sua distância social até o usuário u_i, onde a distância social entre dois usuários é definida como o tamanho do caminho mínimo entre eles, no respectivo grafo de relacionamentos.*

O requisito básico para uma eventual solução é que as distâncias entre do is usuários quaisquer estejam disponíveis ou possam ser eficientemente calculadas durante o processamento da consulta, já que desejamos utilizá-las na função de ordenação de respostas de uma rede social.

Quando lidamos com redes sociais de pequeno porte (com alguns milhares de usuários), podemos facilmente pré-calcular as distâncias sociais (ou de relacionamento) entre todos os possíveis pares de usuários. Podemos ainda fazer um caminhamento em largura no grafo ([33]), a partir do usuário que especifica a consulta, em busca dos

usuários que satisfazem a consulta, encontrando assim as distâncias à medida que o caminhamento progride. Entretanto, para redes sociais de proporções maiores, isso é inviável, já que o número médio de amigos por usuário tende a ser grande. Para ilustrar, suponha um número médio de 100 amigos por usuário em nossa rede social, e considere que o usuário John está procurando por Maria. Suponha que adotássemos a abordagem do caminhamento em largura no grafo a partir de John, computando as distâncias até cada uma das Marias à medida que são encontradas. Com essa abordagem, visitaríamos 100 vértices no primeiro passo, 10.000 no segundo e, no terceiro, 1 milhão de vértices. Se desejarmos encontrar apenas as Marias até uma distância 2 de John, esta abordagem é perfeitamente viável, mas, nesse caso, podemos estar deixando de lado Marias que possam ser de interesse para John. Dessa forma, o número de caminhos a serem considerados para encontrarmos todos os usuários com o nome Maria cresce muito rápido, mostrando que essa abordagem não é uma alternativa viável para a solução do problema.

1.2 A Solução

Neste trabalho, propomos uma abordagem alternativa para o cálculo de distância social na busca por usuários em redes sociais. Nosso método apresenta bons resultados e oferece bom desempenho para redes sociais com um grande número de usuários.

Na Seção anterior citamos duas possíveis soluções para resolver o problema de cálculo de distâncias entre usuários de uma rede social. A primeira consiste em pré-computar e armazenar todas as distâncias, o que demanda muito espaço e, a segunda, em calcular as distâncias em tempo de consulta, o que demanda muito tempo. Nossa solução se situa entre as duas abordagens, numa tentativa de equilibrar os requisitos de espaço e tempo.

Em nosso método, pré-computamos as distâncias de todos usuários até um conjunto fixo de usuários escolhidos *a priori*, que serão utilizadas em tempo de consulta para inferir as distâncias aproximadas entre dois usuários. Os usuários pré-selecionados são chamados de *sementes* e são utilizados como pontos intermediários nos caminhos entre dois usuários quaisquer. O caminho que passa por uma semente não é necessariamente o caminho mínimo entre os dois usuários em questão, nos fornecendo apenas uma estimativa do limite superior do tamanho do caminho mínimo. Como não precisamos de valores exatos para nossa aplicação, apenas de uma ordenação relativa entre os usuários, as distâncias aproximadas são suficientes para a obtenção de resultados satisfatórios, como mostram os experimentos apresentados na Seção 5.3.

1.3 Estrutura da Dissertação

Este trabalho está organizado da seguinte forma. No Capítulo2 fazemos uma revisão da literatura, mostrando alguns trabalhos relacionados. Em seguida, no Capítulo 3, descrevemos de maneira breve o surgimento da *Web*, mostrando em seguida o seu impacto na sociedade e o seu estado atual. No Capítulo 4 descrevemos brevemente as principais técnicas de Recuperação de Informação utilizadas na busca de documentos *Web*, bem como as métricas de avaliação mais difundidas. Nossa solução para a computação eficiente de distâncias de relacionamento e sua introdução na busca por usuários em redes sociais é descrita no Capítulo 5, assim como outras possíveis soluções. Também neste capítulo são mostrados os experimentos de desempenho da nossa solução. Por fim, no Capítulo 5 fazemos uma conclusão do trabalho, descrevendo futuras extensões do trabalho que podem vir a ser desenvolvidas

Capítulo 2

Trabalhos Relacionados

O problema de encontrar caminhos mínimos para uma única fonte em grafos, conhecido como SSSP (*single-source shortest paths*), tem sido intensamente estudado. A solução clássica para o problema, o algoritmo de Dijkstra, foi apresentado em 1959 em [10]. Entretanto, o problema ainda representa um grande desafio para os pesquisadores diante dos crescentes grafos e da necessidade de encontrar soluções exatas ou aproximadas extremamente eficientes.

Existem diversas técnicas para calcular caminhos mínimos entre pares de vértices em grandes grafos esparsos. Uma descrição de algumas técnicas heurísticas de otimização do algoritmo de Dijkstra, bem como propostas de como melhor combiná-las pode ser encontradas em [16].

A idéia da utilização de *marcos* para melhorar algoritmos de caminho mínimo também vem sendo apresentada em alguns trabalhos. Recentemente, a técnica foi combinada com o algoritmo de busca heurística A^* [15], criando um método eficiente e elegante para calcular caminhos mínimos. A abordagem é descrita em [14].

Algoritmos para o cálculo de caminhos mínimos entre todos os pares de um grafo, um problema conhecido como APSP (*all pairs shortest path*), também vêm sendo constantemente melhorados. Os primeiros avanços ocorreram com a utilização de técnicas eficientes de multiplicação de matrizes para grafos não direcionais [31]. Recentemente, os limites inferiores de tempo conhecidos para o problema APSP, no contexto de grafos esparsos, foram melhorados em [7]. Algoritmos para o cálculo aproximado de caminhos mínimos também foram propostos em [11, 8].

Um levantamento sobre os algoritmos para os problemas SSSP e APSP em diferentes cenários, tanto para cálculos exatos quanto para aproximações dos caminhos mínimos foi conduzido em [37].

Todas as melhorias propostas recentemente para o problema SSSP poderiam ser aplicadas ao cálculo dos caminhos mínimos em tempo de consulta em nossa aplicação.

No entanto, existem alguns desafios como o armazenamento do grafo completo em memória ou a utilização de versões dos algoritmos que utilizem memória seundária. Além disso, para fins de busca na *Web*, latências superiores a algumas dezenas de milisegundos já são proibitivas.

Se utilizássemos os algoritmos para o problema APSP, não teríamos problemas de tempos de resposta elevados, já que a computação de todos os caminhos mínimos seria executada previamente. Entretanto, dadas as dimensões de nosso grafo, não é possível armazenar todas as distâncias calculadas. Dessa forma, nenhum dos algoritmos propostos na literatura pode ser diretamente aplicados a nosso problema.

Um outro problema semelhante aos descritos acima é a criação de *oráculos de distâncias aproximadas*, definido em [34]. Em seu estudo sobre cálculo de distâncias em grafos [37], Zwick define o problema como "dado um grafo $G = (V, E)$, queremos pré-processá-lo de forma tal que consultas subseqüentes por caminhos mínimos possam ser respondidas eficientemente". A definição ilustra bem os requisitos da nossa aplicação. Dessa forma, qualquer oráculo poderia ser utilizado diretamente em nossa arquitetura. No entanto, a maioria dos oráculos na literatura oferecem garantias de erros máximos, o que, em nossa aplicação, é desejável mas não um pré-requisito. Assim, decidimos basear nossa solução na idéia de *índices da estrutura do grafo*, conforme proposto em [25], onde o foco é obter soluções aproximadas sem quaisquer garantias.

Embora este trabalho compartilhe várias das idéias apresentadas em [25], uma diferença notável de nosso trabalho, além da aplicação em busca em redes sociais, é que as distâncias pré-calculadas são armazenadas de maneira esparsa, descartando distâncias acima de um dado limite, a partir do qual não oferecem informação útil para nossa aplicação. Para descartar as distâncias maiores, partimos do pressuposto que os usuários de uma rede social estão principalmente interessados em se relacionar com pessoas próximas. Devido a essa característica, podemos utilizar milhões de marcos, denominados *sementes* em nossa solução, sem comprometer nossas restrições de espaço.

O argumento que usuários possuem menor interesse em outros usuários mais distantes no grafo é corroborado por um estudo recente realizado com os usuários da rede social Facebook [21], que sugere que os usuários utilizam a rede social principalmente para se conectar a pessoas com quem já tiveram algum contato pessoalmente.

Acreditamos ser este o primeiro trabalho que ataca o problema de personalização da ordenação de respostas de máquinas de busca em redes sociais através da introdução de distâncias no grafo como uma fonte de evidência.

Capítulo 3

Evolução da *Web*

O surgimento da *World Wide Web* desencadeou uma revolução no mundo moderno. A *Web* mudou radicalmente a maneira de nos comunicarmos e nos relacionarmos. Neste capítulo, vamos traçar um breve histórico da *Web*, descrever seu impacto no mundo moderno e mostrar o seu estado atual.

3.1 História da *Web*

O termo *Internet* é utilizado hoje como sinônimo de *Web*, mas a *Internet* precede a *Web* em várias décadas. Em meados dos anos 80, a maioria dos computadores das universidades e laboratórios de pesquisa nos Estados Unidos e Europa Ocidental já estavam conectados à *Internet*. Até então era possível se conectar a um computador através de ferramentas como *telnet* e recuperar arquivos de servidores públicos através de clientes *FTP*. Nessa época, houve um grande desenvolvimento na infra-estrutura de *software* para facilitar o acesso à *Internet* e foi esse movimento que originou a *World Wide Web*.

O principal responsável pela criação da *Web* foi Tim Berners-Lee, que trabalhava então como consultor do *CERN*. Sua visão era construir um espaço global onde documentos pudessem ser identificados unicamente e pudessem apontar para outros documentos. O objetivo da *Web* era facilitar o compartilhamento de documentos no meio científico. Em 6 agosto de 1991, Berners-Lee tornou a *Web* disponível publicamente na *Internet*, ao anunciat uma primeira implementação do protocolo HTTP e do analisador de documentos HTML. Rapidamente, outras instituições científicas tais como departamentos de universidades e laboratórios de pesquisa passaram a utilizar o novo serviço.

A *Web* causou um grande impacto na comunidade científica e na maneira como a pesquisa era feita e publicada. Pesquisadores passaram a ter acesso em tempo real a

uma grande quantidade de dados e informações, e ao mesmo tempo ganharam a facilidade de publicar suas pesquisas rapidamente, fazendo com que outros pesquisadores pudessem acessá-las. Dessa forma, a colaboração entre diferentes grupos de pesquisa e até mesmo entre diferentes comunidades foi muito facilitada.

Em 1994, iniciou-se a abertura da *Web* em larga-escala, que passou a atingir os demais setores da sociedade, trazendo profundas mudanças na maneira como o mundo se comunicava. Na próxima Seção discutiremos o impacto causado na sociedade pela introdução da *Web*.

O surgimento da *Web* proporcionou um salto na comunicação moderna. O sucesso da *Web* é inegável. Em janeiro de 2007, estatísticas apontavam cerca de 1 bilhão de usuários no mundo e 22 milhões no Brasil[1].

O grande sucesso da *Web* como novo meio de comunicação de alcance global se deve tanto à facilidade de acesso quanto a de publicação de informações. Qualquer indivíduo com acesso à *Web* passou a ter a seu alcance uma grande quantidade de dados, publicados nas mais diversas partes do mundo, sem ter que se deslocar até bibliotecas ou comprar jornais e revistas, passando assim a ter acesso às mais diversas opiniões e idéias de maneira muito mais barata e eficiente. Ao mesmo tempo, esse mesmo indivíduo passou a ter a oportunidade de divulgar suas idéias globalmente, sem sofrer nenhum tipo de controle ou censura, a um custo próximo de zero. Pela primeira vez, o poder sobre a informação deixou de ser exclusividade das editoras e grandes grupos de mídia, passando também para as mãos da sociedade.

3.2 Estado atual da *Web*

Desde sua criação em 1994, a *Web* sofreu grandes mudanças. O aprimoramento e o surgimento de novas tecnologias utilizadas no desenvolvimento das aplicações *Web* permitiram que páginas de conteúdo predominantemente estático, tais como páginas pessoais (*homepages*) e portais, dessem lugar a páginas com interação cada vez maior com os usuários. É inegável que presenciamos hoje o surgimento de serviços *Web* com algumas características marcantes principalmente no sentido de um aumento da participação dos usuários. Seja isso um resultado natural da evolução da *Web* ou não, o fato é que tais serviços vêm se transformando em um fenômeno, promovendo a intensa comunicação das massas e gerando uma enorme quantidade de conteúdo.

Uma das tecnologias com um papel muito importante na *Web* hoje é o chamado *Ajax* (*Asynchronous JavaScript and XML*). Ajax é na verdade uma técnica de desenvolvimento de aplicações *Web* que utiliza tecnologias já conhecidas (*Javascript* e *XML*)

[1] http://www.internetworldstats.com/stats.htm

para criar aplicações altamente interativas. A característica básica de serviços que utilizam Ajax é que apenas pequenas porções do conteúdo das páginas são carregadas do servidor após cada interação do usuário. Dessa forma, não é necessário carregar toda a página do servidor diversas vezes, o que resulta em aplicações mais ágeis e, assim, uma melhor experiência para o usuário. Na prática, a utilização de Ajax fazem com que aplicações *Web* se comportem de maneira muito semelhante a aplicações *desktop*.

Um dos primeiros fenômenos que pôde ser observado em direção a um aumento da comunicação foi o surgimento dos *weblogs* ou *blogs*. De maneira simplificada, os *blogs* nada mais são que páginas pessoais com uma organização cronólogica de suas entradas, ou seja, um diário na *Web*. A inovação dos *blogs* com relação às páginas pessoais foi introduzir a comunicação bilateral. Enquanto nas páginas pessoais a participação dos outros usuários se limita à leitura do conteúdo, nos *blogs* é possível adicionar comentários e ainda apontar para cada uma das entradas individualmente ao invés da página como um todo. Isso fez com que os *blogs* se tornassem um eficiente meio de comunicação, permitindo a discussão de idéias de maneira pública. Pode-se argumentar que os grupos de discussão, muito mais antigos que os *blogs*, exerçam o mesmo papel, o que, em parte é verdade. No entanto, os *blogs* ofereceram uma identidade a seus donos que não era possível nos grupos de discussão coletiva.

Uma tecnologia de grande impacto na *Web* foi a *RSS*, acrônimo para *Really Simple Syndication*. RSS é uma família de formatos *XML*[2] utilizados para agregar conteúdo de páginas atualizadas com freqüência. Para receber as atualizações, são utilizados agregadores de conteúdo, programas que se mantém sincronizados com os respectivos provedores de conteúdo, criando um índice das atualizações ocorridas em cada *site* escolhido. Embora sua primeira versão tenha sido criada em 1999 pela *Netscape*[3], a tecnologia só começou a ser difundida a partir do ano 2000, quando alguns importantes grupos de notícias como *BBC*, *Reuters* e *CNN* passaram a utilizá-la. A tecnologia RSS exerceu um papel importante para o sucesso dos *blogs*, já que permitiu aos usuários receber notificações com atualizações de seus *blogs* favoritos, bastando para isso se inscrever em tais *blogs*.

Um outro exemplo que ilustra o aumento da participação dos usuários na *Web* é a *Wikipédia*[4], uma enciclopédia baseada no princípio de que uma entrada pode ser adicionada ou editada por qualquer usuário. A Wikipédia é, mais que um serviço, um experimento novo, que põe em prova a confiança nos usuários, marcando uma profunda mudança na criação de conteúdo na *Web*. O projeto conta hoje com mais de 6 milhões de entradas em 250 línguas e está entre os 12 *sites* mais visitados em todo o mundo,

[2] http://www.w3.org/XML/
[3] http://www.netscape.com/
[4] http://www.wikipedia.com

o que mostra seu tremendo sucesso. A Wikipédia é um ótimo exemplo de como a participação dos usuários pode contribuir para a geração de conteúdo de qualidade na *Web* sem a necessidade de controle editorial por parte de instituições([12, 26]).

Um fenômeno bastante interessante são os serviços de compartilhamento de conteúdo, pessoal ou não, na *Web*. Alguns exemplos de *sites* em que usuários compartilham material pessoal são *Flickr*[5] e *Webshots*[6], entre outros *sites* de organização e compartilhamento de fotos, e *del.icio.us*[7], um serviço que permite a criação de apontadores (*links*) para as páginas favoritas dos usuários. Um fato interessante é a mudança na maneira de organizar os dados que alguns serviços vem oferecendo. Enquanto serviços como Webshots permitem que os usuários classifiquem seus dados de acordo com categorias pré-definidas, de maneira similar aos diretórios *Web*, *sites* como Flickr e del.icio.us permitem que os usuários atribuam aos objetos uma ou mais etiquetas (*tags*) - palavras-chave escolhidas livremente, gerando assim associações naturais que flexibilizam a recuperação desses objetos. Nessa mesma categoria de serviços, não podemos deixar de citar o *YouTube*[8]. De maneira similar ao Flickr, ele permite o compartilhamento de vídeos e sua categorização através de *tags*, além de permitir a organização tradicional por categorias pré-definidas. As classificações naturais geradas por tais serviços vem sendo chamadas de taxonomias populares.

A colaboração entre os usuários vem sendo uma característica marcante na *Web*. Exemplo disso são a Wikipédia, descrita anteriormente, e o próprio del.icio.us, onde os usuários contribuem para facilitar a organização e busca de páginas *Web*. Um outro exemplo de utilização da *Web* para facilitar a colaboração entre os usuários é o *Google Textos e Planilhas*[9]. Os processadores de textos e planilhas são tradicionalmente produtos para plataformas específicas, o que dificulta muito a colaboração e até mesmo o simples compartilhamento entre os usuários, devido a questões de compatibilidade de versões de *software* e à inconveniência das constantes transferências e sincronização de documentos para o desenvolvimento simultâneo de documentos por mais de um usuário. Uma vez que a plataforma utilizada passa a ser a *Web*, eliminam-se problemas de incompatibilidade e a necessidade de transferências de arquivos, já que eles estão agora em um repositório central, a *Web*.

Pelos exemplos que utilizamos ao longo desta Seção podemos notar que as aplicações *Web* presentes hoje se baseam em arquiteturas de participação, explorando fortemente a interação entre seus usuários. Vimos também que há uma tendência de oferecer personalização aos usuários, o que se traduz em geral na customização de

[5] http://www.flickr.com
[6] http://www.webshots.com
[7] http://del.icio.us
[8] http://www.youtube.com
[9] http://docs.google.com

espaços individualizados nos serviços, como é o caso dos *blogs*, Flickr e serviços de compartilhamento de maneira geral. O sucesso das chamadas *redes sociais*, *sites* com um grande foco em relacionamentos interpessoais, apenas reforça a existência desse padrão de serviços.

Redes sociais são *sites* que, essencialmente, permitem aos usuários criar uma identidade na *Web* e interagir com outros usuários por meio desta identidade. Em geral, essa identidade é definida por meio de um perfil, no qual o usuário declara suas informações e preferências pessoais. Um usuário pode ainda associar ao seu perfil objetos para compartilhar com outros usuários em formatos tais como áudio, vídeo e imagem. Alguns desses *sites* impõem aos usuários estruturas com certa rigidez na criação de seus perfis, como por exemplo Orkut, enquanto outros, como MySpace, permitem um alto grau de customização. As redes sociais integram vários serviços encontrados isoladamente na *Web* como *blogs*, compartilhamento de vídeos e fotos, grupos de discussão e gerenciamento de eventos, centralizando vários serviços de potencial interesse em um único lugar. Contudo, a funcionalidade central de uma rede social é permitir que os usuários estabeleçam relacionamentos de amizade uns com os outros, através dos quais ocorrem as interações.

Embora a base da redes sociais seja a mesma, existem algumas características que as diferenciam e ,muitas vezes, determinam seu sucesso. O site de relacionamento Facebook, criado em 2004, foi inicialmente desenvolvido para estudantes universitários. Seu tremendo sucesso fez com que fosse expadido também para estudantes do ensino médio e em seguida para qualquer pessoa. Para se registrar no Facebook, era preciso estar associado a uma comunidade real, como uma escola ou universidade. Desde meados de 2006, qualquer usuário pode se registrar no *site*. No entanto, todos são encorajados a se afiliar a uma ou mais comunidades, comprovados através de *e-mail*, já que os controles de privacidade são feitos a nível de comunidades. Um dos grandes diferenciais do Facebook é o espaço ilimitado para o armazenamento de fotos e a possibilidade de enviar fotos diretamente dos telefones celulares, o que faz com que o *site* seja hoje o mais utilizado para o compartilhamento de fotos, com mais de 6 milhões de novas fotos por dia.

O *site* de relacionamento Orkut é a rede social de maior sucesso no Brasil. Apesar de não oferecer grandes possibilidades de personalização e possuir basicamente as mesmas funcionalidades dos concorrentes, o serviço conta hoje com mais de 40 milhões de usuários. Até o fim de 2006, o registro no *site* era restrito a usuários convidados por outros usuários do sistema, o que aumentava a idéia de exclusividade do serviço, incentivando muitos usuários a participarem. O fato do serviço pertencer à já conhecida empresa Google também foi um fator importante para os primeiros registros, que foram principalmente de usuários da comunidade acadêmica e técnica na área de Computação.

Existem fortes indícios de que o serviço explodiu no Brasil a partir de uma competição criada pelos próprios brasileiros para ultrapassar o número de usuários americanos. Dessa forma, os brasileiros passaram a enviar convites para seus amigos se registrarem no serviço, processo repetido pelos novos usuários. A partir daí, criou-se um *efeito viral*, que fez com que houvesse um crescimento contínuo do número de brasileiros na rede. Embora a maioria dos usuários sejam brasileiros, essa tendência vem mudando com o crescimento do número de usuários de países como a Índia e os Estados Unidos, principalmente após a abertura do *site* para qualquer usuário.

A rede social de maior sucesso nos Estados Unidos é hoje o MySpace. A principal característica do MySpace e, provavelmente, seu grande fator de sucesso, é o alto grau de personalização permitido a seus usuários. Ao contrário de *sites* como Facebook e Orkut, o MySpace permite a utilização de código HTML criado pelos próprios usuários em seus perfis, o que abre espaço para a inserção de vídeos e animações *Flash*[10]. Além disso, os usuários podem inserir músicas em seus perfis, o que criou um ótimo meio de divulgação para o trabalho de artistas independentes e até mesmo de artistas consagrados. O MySpace é hoje um dos *sites* com o maior número de acessos em todo o mundo e conta com mais de 150 milhões de usuários.

Ao longo desta Seção, discutimos algumas das características de diversas aplicações bem sucedidas na *Web* de hoje. Faremos agora um sumário das principais mudanças que deram origem a nessa nova geração de serviços, caracterizada pela intensa participação dos usuários. O primeiro ponto que devemos ressaltar e que de certa forma permitiu que os serviços *Web* aflorassem foi uma mudança na visão dos desenvolvedores e provedores de serviços, que passaram a enxergar a *Web* como uma plataforma. Essa nova visão possibilitou a transferência de serviços *desktop* para a *Web*, o que não só reduziu os custos de desenvolvimento de *software* como também eliminou problemas de atualização e compatibilidade, além de facilitar a implementação de funções de colaboração entre os usuários.

Outro fator que muito contribuiu para o aumento da participação dos usuários foi o desenvolvimento tecnólogico dos últimos anos, que propiciou a queda de preço e conseqüente disseminação de dispositivos de captura como câmeras e gravadores digitais e celulares com câmera. Esses dispositivos facilitaram imensamente a geração e transferência de formatos digitais para a *Web*. Vídeos, imagens e arquivos de áudio, que antes precisavam ser digitalizados, passaram a ser transferidos diretamente para a *Web* ou, no máximo, utilizando o computador como uma ponte para a transferência, o que está se tornando prática cade vez menos comum, já que, cada vez mais, os computadores se tornam dispensáveis para a utilização da *Web*. Diversos serviços já oferecem integração

[10] http://www.flash.com

com dispositivos móveis através de mensagens de texto (SMS). Além disso, boa parte dos celulares já são capazes de acessar a *Web* e efetuar, por exemplo, transferências de arquivos. Para o usuário, isso se traduz numa grande facilidade na geração e compartilhamento de conteúdo, o que reflete diretamente na maior utilização dos serviços que oferecem esse tipo de integração.

Capítulo 4

Recuperação de Informação na *Web*

Recuperação de Informação (RI) é uma área de pesquisa que estuda a representação, armazenamento, organização e acesso à informação. Até o início dos anos 90, RI foi uma área de interesse restrito principalmente a bibliotecários e especialistas em informação, quando a chegada da *Web* mudou essa realidade.

Como vimos no capítulo anterior, a *Web* se transformou em um repositório de conhecimento e cultura que permitiu o compartilhamento de idéias informações de uma maneira nunca vista. Apesar de seu sucesso, a *Web* trouxe alguns problemas. Em meio a esse mar de informações, nem sempre é uma tarefa fácil encontrar documentos relativos a um assunto de interesse. Para satisfazer sua necessidade, um usuário pode navegar através dos apontadores procurando por informação de interesse. Entretanto, o universo da *Web* é muito vasto, o que torna essa tarefa geralmente ineficiente e frustante para o usuário. Essas dificuldades atraíram grande interesse em RI, dando impulso à pesquisa por novas soluções.

A tarefa de recuperação de informação pode ocorrer basicamente de duas maneiras. A primeira é chamada de *navegação* e a segunda de *recuperação*. No primeiro caso, o usuário tem um interesse em um assunto muito vasto ou não tem um interesse muito bem definido. Dessa forma, sua atenção pode ser facilmente desviada durante a busca por informação, ao contrário do segundo caso, no qual o usuário tem um interesse bem definido e só ficará satisfeito após obter a informação que atenda a seu interesse.

O processo de recuperação ocorre tipicamente da seguinte forma: o primeiro passo é a definição da necessidade do usuário, que deve estar em uma linguagem que possa ser interpretada pelo sistema, em geral um conjunto de palavras-chave que sumarizem o assunto de interesse denominado *consulta*. Em seguida, essa consulta é processada para obter os *documentos recuperados*. O rápido processamento da consulta é possível devido a um *índice* construído previamente, que, em geral, consiste de uma estrutura de listas invertidas que associam termos aos documentos onde ocorrem([1, 35]). O

último passo é a ordenação dos documentos recuperados pela suas relevâncias para que sejam retornados para o usuário. A recuperação de documentos *Web* ocorre de maneira semelhante ao processo que acabamos de descrever.

De maneira simplificada, um sistema de recuperação de informação *Web* pode ser ilustrado como a seguir:

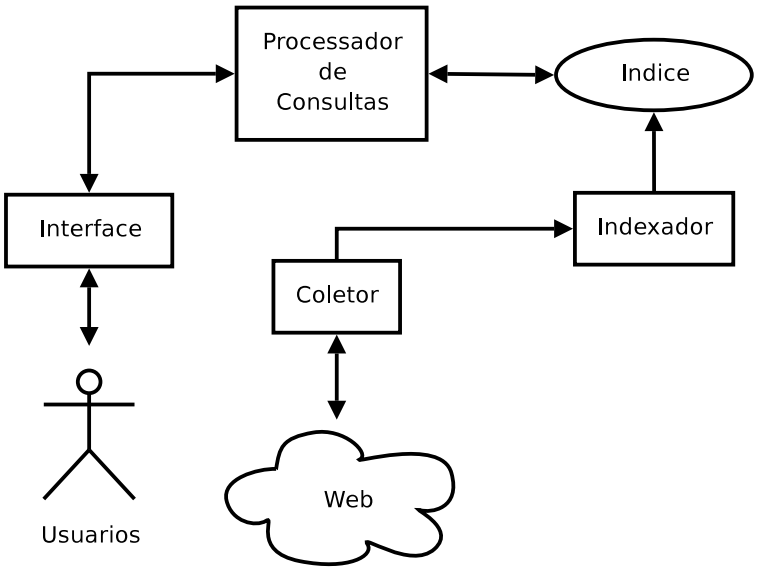

Figura 4.1: Sistema de recuperação de informação *Web*.

O coletor é um componente responsável por reunir as páginas que serão indexadas pelo sistema. A coleta é feita através do envio de requisições a servidores *Web* remotos, quer retornam o conteúdo das páginas. Em teoria, um coletor pode utilizar uma arquitetura centralizada ou distribuída, mas o volume de páginas que as grandes máquinas de buscam indexam hoje (na ordem de bilhões), torna inviável a utilização de arquiteturas centralizadas. Para responder as consultas dos usuários, as páginas coletadas devem ser indexadas. Um índice consiste, de maneira geral, de um *arquivo invertido* contendo, para cada termo do repositório, uma lista dos documentos nos quais ocorre bem como a freqüência de ocorrência. Informações adicionais tais como a posição de cada ocorrência nos documentos podem ser opcionalmente armazenadas no índice.

Com o índice construído, o sistema está pronto para receber e responder consultas, o que é tarefa do processador de consultas. O primeiro passo é identificar e recuperar os documentos que contêm os termos da consulta, o que é feito com o auxílio do índice, bastando para isso fazer a interseção das listas invertidas correspondentes aos termos da consulta para recuperar os documentos com todos os termos da consulta. Se desejarmos recuperar documentos com qualquer um dos termos da consulta, basta fazer a união das mesmas listas. O segundo passo consiste em ordenar os documentos recuperados de acordo com a probabilidade de relevância com relação à consulta, dada por uma

função de ordenação de respostas. Uma função de ordenação de respostas pode levar em conta diversos fatores, alguns desses dependentes da consulta, por exemplo freqüência de ocorrência dos termos da consulta no documento, e outros independentes, como a popularidade do documento. Na próxima Seção descrevemos algumas das técnicas mais comumente utilizadas para para ordenar os documentos *Web*.

4.1 Ordenação de Respostas

4.1.1 Modelo Vetorial

A maioria das máquinas de busca utiliza alguma variação do modelo vetorial[30] em suas funções de ordenação de respostas. O nome do modelo deriva do fato dos documentos e consultas serem representados como vetores em um espaço multi-dimensional. A relevância de um documento para uma dada consulta é dada pela distância entre os dois vetores. Os vetores da consulta e dos documentos são definidos a seguir:

Definição 2 *Seja t o número de termos (indexados pelo sistema) e $K = \{k_1, ...k_t\}$ o conjunto de todos os termos. Considere ainda que n é o número total de documentos e $D = \{d_1, ...d_n\}$ o conjunto de todos os documentos indexados pelo sistema. Por fim, sejam os pesos $w_{i,j} \geq 0$ e $w_{i,q} \geq 0$ associados aos pares (k_i, d_j) e (k_i, q) respectivamente, onde q é a consulta. Dessa forma, o vetor da consulta $\overrightarrow{q}$ é definido como $\overrightarrow{q} = (w_{1,q}, w_{2,q}, ..., w_{t,q})$. O vetor para um documento d_j é definido como $\overrightarrow{d_j} = (w_{1,j}, w_{2,j}, ..., w_{t,j})$*

Pela definição acima, a consulta e os documentos são representados como vetores t-dimensionais. A proposta do modelo vetorial é avaliar o grau de similaridade entre uma consulta q e um documento d_j como uma correlação entre os vetores $\overrightarrow{d_j}$ e $\overrightarrow{q}$. Essa correlação pode ser representada pelo cosseno do ângulo entre esses vetores, o qual é maximizado quando os dois vetores são coincidentes (mesmo ângulo) e minimizada quando os vetores são ortogonais. O cosseno do ângulo entre o vetor documento e o vetor consulta é calculado como se segue.

$$sim(d_j, q) = \frac{\overrightarrow{d_j} \bullet \overrightarrow{q}}{|\overrightarrow{d_j}| \times |\overrightarrow{q}|} \quad (4.1)$$

$$= \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,q}^2}} \quad (4.2)$$

onde $|\overrightarrow{d_j}|$ e $|\overrightarrow{q}|$ são as normas dos vetores do documento e da consulta respectivamente. A norma da consulta não afeta a ordenação dos documentos por ser igual para

todos os documentos. A norma do documento, por sua vez, propicia uma normalização no espaço dos documentos, penalizando documentos mais longos. Como $w_{i,j}$ e $w_{i,q}$ são números não-negativos, sabemos que $sim(d_j, q)$ varia entre 0 e 1. O que resta fazer é definir os pesos dos termos no documento e na consulta.

Os pesos dos termos podem ser calculados através de diversas técnicas, várias delas avaliadas em [29]. Nas técnicas mais efetivas, procura-se balancear dois fatores. O primeiro deles é a frequência de ocorrência de um termo em um dado documento, que indica o quão representativo o termo é para o documento. Esse fator é usualmente denominado fator *tf* (do inglês *term frequency*). O segundo fator procura representar o quanto um termo distingue um documento dos outros e é dado pelo inverso da frequência do termo entre todos os documentos da coleção, o que, na prática, quantifica a raridade do termo na coleção. Esse fator é conhecido como *idf* (*inverse document frequency*). A motivação para a utilização do *idf* é que termos que estão presentes em muitos documentos não são muito úteis para distinguir um documento relevante de um não-relevante.

Definição 3 *Seja N o número de documentos indexados pelo sistema e n_i o número de documentos nos quais o termo k_i está presente. Seja ainda $freq_{i,j}$ a freqüência absoluta do termo k_i no documento d_j, ou seja, o número de ocorrências do termo k_i no documento d_j. Temos que a freqüência normalizada $f_{i,j}$ do termo k_i no documento d_j é dada por:*

$$f_{i,j} = \frac{freq_{i,j}}{max(freq_{l,j})} \tag{4.3}$$

onde max é computado sobre todos os termos presentes no texto do documento d_j. Seja ainda o idf do termo k_i definido por

$$idf_i = log\frac{N}{n_i} \tag{4.4}$$

Os melhores esquemas de pesos utilizam pesos dados por variações da seguinte fórmula.

$$w_{i,j} = f_{i,j} \times idf_i \tag{4.5}$$

Tais estratégias são chamadas de esquemas de tf-idf.

Algumas variações da expressão acima são descritas em [28]. Entretanto, de maneira geral, a fórmula acima se mostra como um bom esquema para diversas coleções de documentos.

Uma observação que devemos fazer é que, pelo modelo vetorial, um documento pode ser recuperado mesmo que não contenha todos os termos da consulta. Entretanto, na prática, a maioria das máquinas de busca comerciais retorna apenas os documentos que casam com todos os termos da consulta a não ser que o usuário especifique o contrário explicitamente.

4.1.2 Ordenação de Respostas na *Web*

Embora os primeiros estudos mostrassem que o modelo vetorial apresentava bons resultados quando utilizado na busca *Web*[36], o aumento do número de páginas fez que com isso deixasse de ser verdade. O modelo, desenvolvido inicialmente para busca em bases textuais, não aborda os problemas trazidos pela *Web*, tampouco explora suas características para melhorar os resultados da busca. A utilização de novas evidências se tornou vital para a melhoria da qualidade das máquinas de busca, que antes lidavam com alguns milhares de documentos e passaram a lidar com milhões e posteriormente bilhões de documentos.

Uma característica importante da *Web* e que a diferencia de bases textuais é sua estrutura de apontadores. De maneira análoga ao mundo acadêmico, onde em geral o número de citações para um trabalho indica sua importância, o número de apontadores para uma página nos dá uma medida da sua popularidade e qualidade. Utilizando a mesma analogia, se citações em comum entre dois trabalhos ou trabalhos citados por um mesmo trabalho indicam uma relação entre esses dois trabalhos, apontadores comuns a duas páginas ou duas páginas apontadas pela mesma página indicam uma relação entre essas duas páginas. Apresentaremos algumas técnicas de ordenação de respostas que exploram essas características.

Uma técnica simples para explorar a estrutura de apontadores da *Web* é descrita em[5]. A técnica apresentada nesse trabalho consiste em ordenar os resultados pela sua conectividade, ou seja, pelo número de apontadores presentes nas páginas, pelo número de apontadres para as páginas ou ainda pela soma de ambos. Uma abordagem parecida é utilizada em [22], que também utiliza a descrição dos apontadores que fazem referência a um documento, conhecidos como *textos de âncora*, como parte deste documento para melhorar os resultados.

Uma abordagem mais avançada, conhecida algoritmo de *HITS* (*Hypertext Induces Topic Search*), foi apresentada em[20]. O método de ordenação de respostas proposto neste trabalho é dependente da consulta e considera o conjunto de páginas S que apontam ou são apontadas pelas páginas no conjunto de respostas. Páginas referenciadas por muitos apontadores são chamadas *autoridades*, páginas com conteúdo pontencialmente relevante. Páginas com muitos apontadores para outras páginas são chamadas

hubs e são tidas como boas referências para outras páginas com conteúdo relevante. A qualidade das autoridades e dos hubs estão relacionados da seguinte forma: boas autoridades são apontadas por bons *hubs* e bons *hubs* apontam para boas autoridades. Essa relação é definida a seguir.

Definição 4 *Sejam $H(p)$ e $A(p)$ os valores de hub e autoridade de uma página p, esses valores devem ser calculados de forma que as seguintes equações sejam satisfeitas para todas as páginas p:*

$$H(p) = \sum_{u \in S | p \to u} A(u) \tag{4.6}$$

$$A(p) = \sum_{v \in S | v \to p} H(v) \tag{4.7}$$

Os valores de *hub* e autoridade são determinados através de um algoritmo iterativo, que mantém e atualiza tais valores para cada página. Os valores de cada tipo são normalizados de forma que a soma de seus quadrados seja igual a 1. As páginas com valores mais altos de $H(p)$ e $A(p)$ são consideradas melhores *hubs* e autoridades respectivamente. Na proposta original, os valores de autoridades são utilizados para a ordenação dos resultados. No entanto, isso pode fazer com que páginas de tópicos muito mais amplos que o da consulta original apareçam, já que o conjunto original de resultados é expandido. Uma solução para esse problema é analisar o conteúdo das páginas com uma técnica tradicional de RI (por exemplo *tf-idf*), atribuindo um peso a cada página, que pode ser incluído nas equações 4.6 e 4.7 como multiplicador de cada termo do somatório [2, 6]. Experimentos conduzidos em [2] mostram que, utilizando-se a técnica descrita acima, a revocação e precisão melhoram consideravelmente para os 10 primeiros resultados.

Uma outra abordagem para utilizar a estrutura de apontadores da *Web* para melhorar a busca é a estratégia proposta pelos criadores da máquina de busca *Google*[1], chamada *PageRank* [24]. PageRank pode ser entendido como um modelo de comportamento do usuário, que navega aleatoriamente a partir de uma página inicial qualquer, seguindo apontadores indefinidamente. O usuário do modelo nunca retorna a uma página anterior, mas eventualmente se cansa da navegação e recomeça de uma outra página qualquer. A probabilidade desse usuário visitar uma página qualquer corresponde ao *PageRank* da página, enquanto a probabilidade do usuário interromper a navegação e recomeçar de uma nova página aleatória é chamada de fator de amortecimento d(*damping factor*). Pode-se alternativamente adicionar o fator d a uma única página ou grupo de páginas, abrindo assim espaço para personalização.

[1] http://www.google.com

Uma propriedade da estratégia é que uma página pode ter um alto valor de *PageRank* se muitas outras páginas apontam para ela ou se algumas das páginas que apontam para esta página possuem um alto valor de *PageRank*. De maneira intuitiva, páginas muito referenciadas na *Web* são páginas de conteúdo possivelmente interessante, além de páginas referenciadas por fontes reconhecidas, como por exemplo *Yahoo!*[2], que dificilmente apresentariam referências para páginas de baixa qualidade ou inexistentes. Assumindo que p é apontada pelas páginas p_1 a p_n, o valor do *PageRank* de uma página p é dado por:

$$PR(p) = d + (1 - d) \sum_{i=1}^{n} \frac{PR(p_i)}{C(p_i)} \tag{4.8}$$

onde $C(p_i)$ corresponde ao número de apontadores presentes na página p_i. De maneira semelhante ao algoritmo *HITS*, o cálculo do *Pagerank* é feito através de um algoritmo iterativo.

Outra característica importante na *Web* é que a descrição dos apontadores introduz uma nova fonte de informação a respeito das páginas. O texto de âncora, texto que descreve um apontador para uma outra página, freqüentemente descreve a página para a qual aponta melhor que o próprio conteúdo da página. Essa característica é explorada em [23], que descreve a extinta máquina de busca *World Wide Web Worm*, e posteriormente em [3], que descreve a arquitetura da busca Google em 1998. Em ambos os trabalhos, os textos de âncora são associados às páginas para as quais apontam. Uma grande vantagem da estratégia é que ela permite que páginas que não foram coletadas ou que não possuam informação textual possam ser retornadas. Além disso, a qualidade dos resultados pode melhorar significativamente com a utilização de texto de âncora. Um estudo apresentado em [13] mostra que os textos de âncora são bastante semelhantes às consultas reais feitas por usuários de máquinas de busca, mostrando-se assim com uma boa fonte de informação para a ordenação de respostas. As principais máquinas de busca utilizam hoje o texto de âncora em suas funções de ordenação de respostas. Outras informações referentes à estrutura dos documentos são utilizadas pelas máquinas de busca atuais. Por exemplo, a *Google* declara atribuir pesos maiores a palavras em destaque no texto, por exemplo, palavras em negrito ou com fonte maior que o restante do documento, bem como palavras no título do documento [3].

[2] http://www.yahoo.com

4.2 Avaliação

No desenvolvimento de um sistema de recuperação de dados qualquer como por exemplo um banco de dados, o tempo de resposta e o espaço de armazenamento necessário são as métricas geralmente adotadas para a avaliação do sistema Entretanto, em um sistema de recuperação de informação, outras métricas devem ser levadas em conta. Ao contrário de um sistema de recuperação de dados, em um sistema de RI, a consulta do usuário representa um interesse vago para o qual não existe um conjunto de respostas exatas, já que o sistema simplesmente retorna documentos ordenados de acordo com sua relevância para a consulta especificada. Dessa forma, é necessário avaliar a qualidade de um conjunto de documentos retornados para uma dada consulta.

As métricas mais utilizadas para medir a qualidade de um sistema de busca são as medidas de *precisão* e *revocação*. Considere uma consulta q, que expressa uma necessidade de informação de um dado usuário. Seja R o conjunto de documentos relevantes para essa consulta e $|R|$ o número de documentos nesse conjunto. Suponha que o sistema de busca avaliado retorna um conjunto de respostas A. Seja $|A|$ o número de documentos no conjunto de respostas e $|R_a|$ o número de documentos na interseção dos conjuntos de relevantes e de respostas R e A. Dessa forma, as medidas de revocação e precisão são definidas como:

Definição 5 *A revocação para uma consulta é a fração dos documentos relevantes que estão entre os documentos retornados pelo sistema.*

$$Revocação = \frac{|R_a|}{|R|} \tag{4.9}$$

Definição 6 *A precisão para uma consulta é a fração dos documentos retornados pelo sistema que estão entre os documentos relevantes para a consulta.*

$$Precisão = \frac{|R_a|}{|A|} \tag{4.10}$$

Tanto a precisão quanto a revocação assumem que todos os documentos no conjunto de documentos retornados (A) foram examinados. No entanto, de maneira geral, o usuário vê apenas um pequeno subconjunto de repostas. Experimentos inconclusivos sugerem que na média os usuários seguem os resultados na ordem em que aparecem, o que faz com que as medidas de precisão e revocação variem à medida que o usuário avança no conjunto de resultados apresentados. Essa variação é freqüentemente avaliada através de um gráfico da precisão em função da revocação.

Para traçar um gráfico de precisão por revocação, calcula-se a precisão a cada ponto de mudança da revocação no conjunto de resultados, ou seja, cada vez que um novo

resultado relevante é alcançado. Para o cálculo da precisão, assume-se que o conjunto dos resultados retornados A compreende os documentos que vão do primeiro resultado até o último resultado relevante alcançado. Como o cálculo é feito individualmente para cada consulta, para avaliar o sistema como um todo é calculada a média das precisões de um conjunto de consultas para cada nível de revocação e, em seguida, é feita a interpolação das precisões. De modo geral, as curvas de precisão versus revocação são traçadas para 11 níveis, correspondentes aos 10 primeiros resultados retornados por uma máquina de busca, o que equivale à primeira página de respostas. Curvas de precisão média versus revocação são bastante úteis para comparar o desempenho de diferentes funções de ordenação de respostas para um conjunto de consultas de teste.

Um dos problemas da utilização das curvas de precisão em função da revocação nos sistemas de busca *Web* é que suas coleções são muito grandes, o que torna virtualmente impossível determinar o conjunto completo de documentos relevantes. Uma alternativa é utilizar o método de *pooling*, descrito em [18], que consiste em selecionar um conjunto de possíveis documentos relevantes, por exemplo os 20 primeiros resultados da mesma consulta em diferentes máquinas de busca, os quais são examinados por especialistas, que julgam suas relevâncias. Os documentos determinados como relevantes são então considerados como o conjunto final para os cálculo de precisão e revocação.

Outra maneira de avaliar o desempenho de um sistema de RI é computar a precisão em determinados pontos de corte no conjunto de resultados. A precisão para uma consulta pode ser calculada avaliando-se apenas os n primeiros resultados do conjunto de documentos retornados. Nesse caso, a métrica é denominada *precisão em n* ou *P@n*. No caso especial em que n corresponde ao número de documentos relevantes para a consulta, a métrica recebe o nome de *R-precisão*. Essas métricas são muito utilizadas para a avaliação individual de cada uma das consultas em um experimento. As métricas R-precisão e P@n também podem ser utilizadas para comparar o desempenho de duas funções consulta a consulta, bastando para isso traçar um histograma com a diferença das precisões apresentadas por cada algoritmo para cada uma das consultas. Pode-se também calcular a média da R-precisão ou P@10 para várias consultas e utilizar o resultado para avaliar o desempenho de uma função de ordenação de respostas.

Capítulo 5

A solução e resultados

Neste capítulo, apresentamos em detalhes nossa solução para introduzir distância social na busca por pessoas em redes sociais. No entanto, para entender a solução proposta neste trabalho, deve-se compreender primeiramente os pré-requisitos impostos pelo problema. Como discutido na Seção 1.1, o requisito básico para uma solução é que as distâncias de relacionamento estejam disponíveis ou possam ser eficientemente calculadas durante o processamento da consulta, já que desejamos utilizá-las na função de ordenação de respostas. Tempos de execução extremamente baixos são necessários para que o desempenho das consultas não seja muito afetado pela introdução das distâncias de relacionamento como nova evidência na função de ordenação de respostas.

Inicialmente, o problema pode parecer simples. No entanto, devido às proporções dos grafos de usuários encontrados nas redes sociais, atuais soluções encontradas na literatura não se mostram eficazes na prática. O principal desafio é encontrar uma solução eficiente em termos de tempo de execução e espaço de armazenamento.

Na Seção 5.1, apresentamos possíveis soluções para o problema, expondo as razões pelas quais são inviáveis quando utilizadas em redes sociais de grande porte. Em seguida, na Seção 5.2, descrevemos a solução baseada em sementes proposta neste trabalho e, por fim, na Seção 5.3, fazemos uma comparação em termos de desempenho e qualidade das respostas entre a solução proposta e as outras possíveis soluções apresentadas.

5.1 Possíveis soluções

5.1.1 Solução 1: Pré-computação de todas as distâncias

Uma possível solução é pré-computar as distâncias entre todos os pares de usuários do grafo, de tal maneira que nenhuma computação seja feita em tempo de consulta. O principal problema desta solução é o espaço necessário para o armazenamento das

distâncias. Para um grafo de 10 milhões de usuários precisaríamos de alguns *exabytes* para armazenar todas as distâncias, o que torna proibitiva a utilização desta abordagem em sistemas reais. O espaço necessário para o armazenamento das distâncias entre todos os pares de usuários pode ser visto na Tabela 5.1.

Número de usuários	Espaço(GB)
10.000	48
100.000	4.768
1.000.000	476.837
10.000.000	47.683.715
100.000.000	4.768.371.582

Tabela 5.1: Espaço necessário para armazenar as distâncias entre todos os pares de usuários do grafo. Os cálculos levam em conta que cada distância pode ser armazenada em 4 bits.

É possível argumentar que não é necessário armazenar as distâncias entre todos os pares de usuários. Ainda assim, se considerarmos uma média de 100 amigos por usuário, então o número médio de usuários até uma distância 3 de um dado usuário é de um milhão (10^6), resultando em uma demanda de espaço de armazenamento ainda na ordem de *exabytes*, como mostra a Tabela 5.2.

Número de usuários	Espaço(GB)
10.000	24
100.000	2.384
1.000.000	238.418
10.000.000	2.384.185
100.000.000	23.841.857

Tabela 5.2: Espaço necessário para armazenar distâncias menores ou iguais a 3 entre todos os pares de usuários do grafo. Os cálculos levam em conta que cada distância pode ser armazenada em apenas 2 bits.

Uma outra alternativa seria ordenar apenas os usuários até uma distância 2. No entanto, essa não seria uma boa opção por alguns motivos. Em primeiro lugar, usuários a uma distância maior que 2 podem ainda assim ser de interesse para o usuário que executa uma busca, em especial nos casos em que outros sinais indicam similaridades entre esses usuários. Em segundo lugar, um usuário pode buscar informações a respeito de um outro usuário que não seja amigo ou amigo de amigo. Mesmo neste caso, a ordenação por distância social ainda se mostra interessante como um sinal para a ordenação, dado que, na ausência de outros sinais, a ordenação final pode não ter

critério algum de ordenação, o que certamente não é uma boa característica de uma função de ordenação de respostas.

5.1.2 Solução 2: Cálculo de distâncias em tempo de consulta

Uma segunda possível solução seria calcular todas as distâncias em tempo de consulta. Considere que o usuário John está conectado ao sistema e executou a consulta 'Maria'. Uma possível abordagem para responder à consulta seria simplesmente executar uma busca em largura por todos os usuários que satisfaçam a consulta partindo de John. Em um grafo no qual a média de amigos por usuários é 100, se limitássemos a busca a usuários a uma distância máxima 3 de John, seria necessário expandir em média 1 milhão de usuários por consulta. Uma alternativa mais sofisticada seria executar uma busca em largura bidirecional como descrito em [27], expandindo o grafo não apenas a partir de John como também a partir de cada Maria, e procurando por interseções nas listas de usuários alcançadas a cada nova expansão. Esta última solução, denominada *solução força bruta*, é avaliada na Seção 5.3.

5.1.3 Solução 3: Pré-computação de listas de amigos

Uma terceira solução é uma combinação das duas primeiras soluções apresentadas anteriormente e consiste em, para cada usuário, criar um índice contendo a lista de seus amigos e, em tempo de consulta, fazer a interseção das listas do usuário de origem e dos usuários presentes nos resultados. Esta solução é eficiente e requer pouco espaço de armazenamento, mas possui a grande desvantagem de identificar apenas os relacionamentos entre amigos e amigos de amigos.

Uma abordagem mais promissora seria armazenar também as listas de amigos de amigos e fazer a interseção destas com a lista de amigos do usuário de origem, o que nos permitiria capturar relacionamentos de distância até 3. Apesar de essa parecer uma solução viável, os experimentos demonstrados na seção 5.3 nos mostram que a precisão da ordenação é bastante afetada pela limitação da distância sociais até o nível 3 imposto por essa abordagem. Além disso, os requisitos de tempo e espaço são razoalvemente altos, o que torna essa alternativa pouco atraente para ser usada em redes sociais de grande porte como o Orkut. Em nossos experimentos nos referimos a esta abordagem como *interseção de listas*.

5.2 Solução Baseada em Sementes Aleatórias

Nesta seção apresentamos uma solução baseada em sementes aleatórias para o problema de computar distâncias entre usuários de uma rede social. Nossa solução captura relacionamentos de distâncias menores ou iguais a 4, apresentando um elevado nível de precisão, além de requisitos espaço e tempo bastante satisfatórios. Nossa abordagem é dividida em duas fases, uma executada durante a indexação e outra em tempo de consulta. Na fase de indexação nós calculamos os vetores de distâncias de todos os usuários até as sementes. Tais vetores serão utilizados durante a consulta para aproximar a distância entre o usuário executando a consulta e cada um dos usuários do conjunto de resultados, Detalhamos a seguir cada uma das fases da solução proposta.

5.2.1 Índices que Representam a Estrutura do Grafo

Antes de explicarmos nossa abordagem, introduziremos a fundamentação matemática na qual baseamos nossa solução. Nossa discussão é inteiramente baseada em[25]. Assim como diversos outros problemas em Ciência da Computação, encontrar distâncias entre usuários de uma rede social equivale a encontrar caminhos mínimos em um grafo. Com dezenas de milhões de vértices no grafo, computar os caminhos mínimos em tempo de consulta é inviável. Nesse caso, a alternativa é pré-computar um índice que representa a *estrutura do grafo* ou suas características fundamentais.

O índice da estrutura do grafo é formado por dois componentes. O primeiro é um conjunto de anotações dos vértices, que provê informação de localização relativa. Formalmente, dado um grafo bidirecional $G(V, E)$ composto de um conjunto de vértices V e um conjunto de arestas E, as anotações definem uma função $A : V \rightarrow T$, onde T é um conjunto de anotações. Uma anotação $t \in T$, que pode ser um vetor multidimensional de valores como proposto aqui, é associado a cada vértice do grafo. O segundo componente do índice é uma função $similaridade : T \times T \rightarrow R$ que associa um número real a qualquer par de anotações dos vértices. Este número pode ser, por exemplo, o caminho mínimo entre os respectivos vértices. Entretanto, nestre trabalho, a função *similaridade* nos dá apenas uma estimativa da *proximidade* entre dois vértices. Os valores retornados pela função *similaridade* não precisam fornecer uma idéia precisa da distância entre dois usuários, bastando que, para diferentes pares, os valores sejam consistentes e assim possam prover um ordenação relativa confiável para um conjunto de pares de usuários.

5.2.2 Anotações Baseadas em Sementes Aleatórias

No caso específico da busca por usuários em uma rede social, nosso principal problema é ordenar os usuários com relação a uma dada consulta. Sendo assim, não precisamos de uma estimativa precisa do caminho mínimo entre dois usuários, precisamos apenas ordenar os usuários de acordo com uma métrica que seja inversamente proporcional à distância do caminho mínimo.

Optamos por um sistema de marcos (*network landmarks*) que consiste em pré-selecionar um conjunto de vértices denominados *sementes* que servirão como pontos de referência navegacionais. A partir daí, tomando como ponto de partida cada uma das sementes, executamos uma busca em largura de modo a alcançar todos os vértices do grafo. Cada vértice alcançado recebe uma anotação contendo a distância até a semente da qual partimos, de tal maneira que o vetor T_i de anotações para um vértice V_i contém as distâncias de V_i para cada uma das sementes. Essas anotações são geradas durante a fase de indexação.

As sementes devem ser escolhidas de acordo com algum critério. Neste trabalho, utilizamos um conjunto de vértices escolhidos aleatoriamente como sementes. Também exploramos a variação da precisão de nossa função de similaridade em função do número de sementes utilizadas. Como discutido adiante, para obter boas precisões, precisamos utilizar um grande número de sementes.

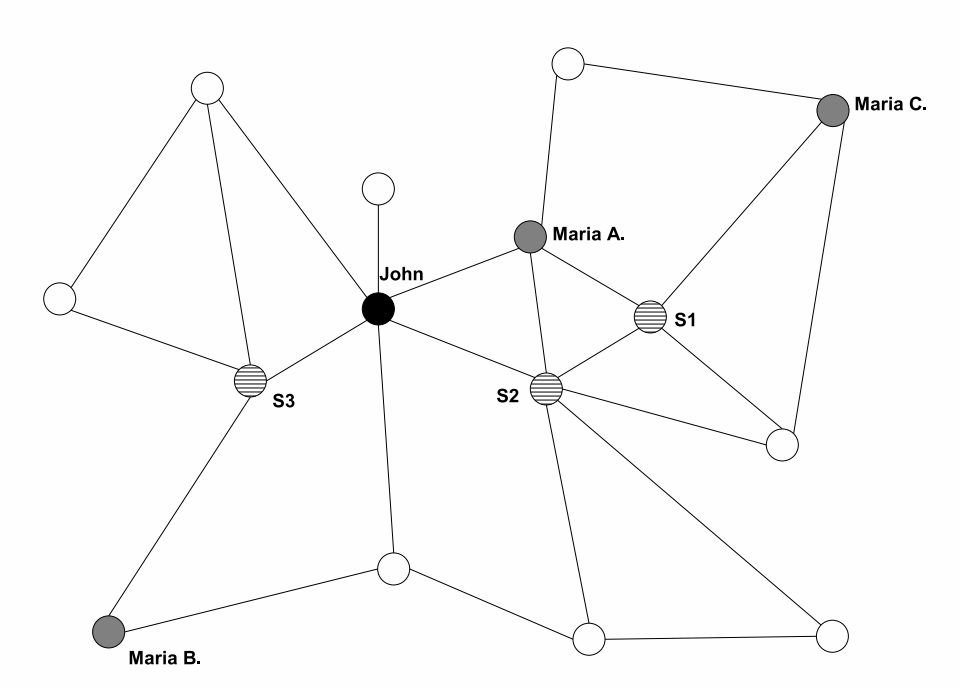

Figura 5.1: Grafo com 3 sementes pré-selecionadas: S1, S2 e S3.

A Figura 5.1 ilustra uma versão modificada no nosso grafo de relacionamentos da Figura 1.1. A diferença é que agora selecionamos 3 sementes para serem nossos pontos de referência de navegação, indicadas por S1, S2 e S3. Para esse caso, os vetores de distâncias até as sementes são mostrados a seguir:

Exemplo 1 *Vetores de distâncias até as sementes para os usuários da Figura 5.1.*

$$\begin{aligned} D_{John} &= [2,1,1] \\ D_{MariaA} &= [1,1,2] \\ D_{MariaB} &= [4,3,1] \\ D_{MariaC} &= [1,2,4] \end{aligned}$$

John está a distâncias 2, 1 e 1 das sementes S1, S2 e S3 respectivamente. Para 'Maria A', o vetor de distâncias é dado por $[1,1,2]$ e assim por diante.

É importante observar que, enquanto as sementes provêem informações quanto à posição *relativa* dos vértices (capturando assim parte da estrutura do grafo), elas não nos permitem computar diretamente todos os caminhos mínimos. De fato, usando os vetores de distâncias acima podemos inferir que a distância mínima entre John e 'Maria A' é 2 ao invés de 1. Assim, os vetores de distâncias nos permitem apenas aproximar o tamanho do caminho mínimo. Quanto maior o número de sementes, melhor essa aproximação tende a ser para um grande número de vértices.

5.2.3 Computação dos Vetores de Distâncias

Em nossos experimentos, observamos que distâncias maiores que 4 entre usuários não são de muita serventia em uma rede como o Orkut. Isso se deve ao fato de cada usuário possuir um grande número de amigos na média, o que faz com que , após caminharmos 4 níveis no grafo, alcancemos um número muito alto de usuários. Na prática, isso significa que não é necessário computar distâncias maiores que 4. Considere agora que as distâncias entre os usuários são estimadas através dos vetores de distâncias até as sementes. Seja $d(S_i, u_j)$ a distância entre a semente S_i e o usuário u_j e $d(u_j, u_k)$ a distância entre os usuários u_j e u_k. Se restringirmos nosso raciocínio aos casos em que a distância estimada está correta, temos as seguintes possibilidades:

- $d(u_j, u_k) = 1 \rightarrow$ existe uma semente S_i cujas distâncias até u_j e u_k são (0,1) ou (1,0);
- $d(u_j, u_k) = 2 \rightarrow$ existe uma semente S_i cujas distâncias até u_j e u_k são (0,2), (1,1), or (2,0);

- $d(u_j, u_k) = 3 \rightarrow$ existe uma semente S_i cujas distâncias até u_j e u_k são (0,3), (1,2), (2,1), or (3,0);
- $d(u_j, u_k) = 4 \rightarrow$ existe uma semente S_i cujas distâncias até u_j e u_k são (0,4), (1,3), (2,2), (3,1), or (4,0).

É possível ver que, se considerarmos somente as distâncias até sementes menores ou iguais a 2, capturaremos a maior parte desses casos. Os casos deixados de fora são apenas aqueles em que há uma distância 3 ou 4 na tupla, os quais são casos menos significativos em nossa função de similaridade por refletirem relacionamentos mais distantes. Além disso, é importante ressaltar que existem diversas sementes oferecendo informações possivelmente equivalentes a respeito da distância entre dois vértices, de tal forma que o fato de descartarmos as distâncias provenientes de uma semente não significa que uma outra semente não contenha informação semelhante.

Como resultado desse raciocínio, consideramos apenas as distâncias até as sementes menores ou iguais a 2. Se uma distância é maior que 3, nós a consideramos uma distância infinita. Com essa aproximação, nossos vetores de distâncias parciais ilustrados no exemplo 1 se transformam nos vetores a seguir.

Exemplo 2 *Vetores aproximados de distâncias parciais para os usuários da Figura 5.1.*

$$\begin{aligned} D_{John} &= [2, 1, 1] \\ D_{MariaA} &= [1, 1, 2] \\ D_{MariaB} &= [\infty, \infty, 1] \\ D_{MariaC} &= [1, 2, \infty] \end{aligned}$$

Observe que, nesse caso, as estimativas para os caminhos mínimos de John para cada Maria permanecem inalterados como 2, 2 e 3.

5.2.4 *MapReduce*

Antes de prosseguirmos com o funcionamento de nosso algoritmo, faremos uma breve descrição do arcabouço utilizado, *MapReduce*. *MapReduce* é um modelo de programação utilizado para o processamento de grandes quantidades de dados descrito em [9]. Nesse modelo, os usuários especificam uma função *Map*, que processa pares chave-valor e gera um conjunto de pares chave-valor intermediários, e uma função *Reduce*, que agrupa todos os valores intermediários associados à mesma chave intermediária. Programas escritos no arcabouço do *MapReduce* são automaticamente distribuídos entre um conjunto de máquinas nas quais são executados.

Uma computação que utiliza o modelo *MapReduce* recebe como *entrada* e produz como *saída* um conjunto de pares chave-valor. Para cada par chave-valor recebido pela função *Map*, é gerado um conjunto de pares intermediários. Todos os valores associados a uma mesma chave intermediária são agrupados e repassados para a função *Reduce*, que recebe como entrada uma chave intermediária e um conjunto de valores associados à chave. Tipicamente, os valores são processados de forma a gerar um valor único ou um conjunto menor de valores para a chave à qual estão associados. O Exemplo 3 ilustra o funcionamento de um programa simples que utiliza o modelo *MapReduce*.

Exemplo 3 *Considere o problema de contar o número de ocorrências de cada palavra em uma grande coleção de texto. O arcabouço de programação MapReduce pode ser utilizado para resolver o problema com o pseudo-código a seguir.*

Algoritmo 1 Algoritmo para contagem de número de ocorrências de palavras em uma coleção de texto

```
1:  Map(chave, valor)
2:  começo
3:     para todo palavra p em valor do
4:        emitir(p, 1)
5:     fim para
6:  fim
7:  Reduce(chave, valores)
8:  começo
9:     resultado ← 0
10:    para todo v em valores do
11:       resultado ← resultado + 1
12:    fim para
13:    emitir(resultado)
14: fim
```

Execução

Quando o programa do usuário invoca a função *MapReduce*, ocorre a seguinte seqüência de ações. Em primeiro lugar, os dados de entrada são particionados entre um número de tarefas *Map*. Em seguida, cada tarefa invoca a função *Map* para cada um dos pares chave-valor disponíveis nos dados de entrada. O pares chave-valor intermediários, produzidos pela função *Map*, são particionados e repassados para as tarefas *Reduce*. O passo seguinte é a ordenação dos pares pelas chaves, que agrupa os pares com a mesma chave. Esse passo é necessário já que, tipicamente, diversas chaves distintas são enviadas para uma mesma tarefa *Reduce*. O último passo é o processamento de cada chave e seu conjunto de valores pela função *Reduce*, que escreve os pares finais em arquivos particionados pelas chaves.

5.2.5 Algoritmo para Computação dos Vetores de Distâncias

Para gerar a lista completa de distâncias até todas sementes para cada um dos usuários, o primeiro passo é adicionar uma entrada com distância 0 às listas de cada uma das sementes. Tais entradas correspondem às próprias sementes. Essa fase é o ponto de partida para completarmos as listas dos demais usuários do grafo (alcançados a partir das sementes) e também das sementes. Chamaremos essa fase de *marcação de sementes*. O passo seguinte consiste de uma série de *propagações* de distâncias que serão explicadas a seguir.

Durante a primeira propagação, o grafo é expandido a partir das sementes em direção a seus amigos, que recebem uma entrada em suas respectivas listas com distância 1 correspondente à semente de onde foi iniciada a expansão. Em cada propagação subseqüente, o grafo é expandido por mais um nível de amizade, a partir dos usuários alcançados na propagação anterior. Ao final da $f-ésima$ propagação, temos os vetores de distâncias parciais de todos os usuários alcançados pelas sementes em até f níveis de relacionamento.

Na prática, ao invés de executarmos uma busca no grafo a partir de cada semente, utilizamos um processo *MapReduce*[9] que executa algumas iterações sobre todos os usuários propagando suas distâncias em direção a seus amigos. O pseudo-código correspondente à fase de marcação de sementes e às propagações descritas acima estão ilustrados nas Figuras 2 e 3 respectivamente.

Algoritmo 2 Algoritmo de marcação de sementes

```
1: Map(id_usuario, amigos[1..a], sementes)
2: começo
3:    se encontra(sementes, id_usuario) então
4:       distancias[id_semente] ← 0
5:       emitir(id_usuario, amigos, distancias)
6:    senão
7:       emitir(id_usuario, amigos, ∅)
8:    fim se
9: fim
10: Reduce(id_usuario, amigos[1..a], distancias)
11: começo
12:    emitir(id_usuario, amigos, distancias)
13: fim
```

Como descrito na Subseção anterior, a entrada e saída de um programa *MapReduce* consistem de um conjunto de pares chave-valor. Em nosso algoritmo de marcação de sementes, as chaves de nossos pares correspondem aos identificadores de todos os usuários de uma rede social, e os valores correspondem às listas de amigos dos usuários correspondentes. Na fase *Map*, cada um dos pares é processado pela função *Map*

definida acima. Além dos pares de usuários e listas de amigos, provemos também a lista de sementes pré-selecionadas para a fase *Map*. As chaves intermediárias enviadas para a fase *Reduce* são exatamente as mesmas chaves recebidas como entrada pela fase *Map*, acrescidas de uma lista de distâncias até as sementes, onde cada distância é representada como um par composto pelo identificador da semente e pelo valor da distância até essa semente.

O processamento feito pela função *Map* é bastante simples. Primeiramente, a função verifica se o usuário processado corresponde a uma das sementes. Em caso afirmativo, uma distância de valor 0 até o próprio usuário é adicionada à lista de distâncias do usuário e, em seguida, o par composto pelo usuário e suas listas de amigos e distâncias é enviado à fase *Reduce*. Caso contrário, o par é enviado para a fase *Reduce* inalterado. Na fase *Reduce*, os pares não sofrem qualquer alteração e são, em seguida, armazenados em arquivos particionados pelos identificadores dos usuários.

Algoritmo 3 Algoritmo de propagação

```
1:  Map(id_usuario, amigos[1..a], distancias)
2:  começo
3:    para todo dist em distancias do
4:      se dist.distancia = num_fase − 1 então
5:        para i = 1 até a do
6:          nova_distancia.id_semente ← dist.id_semente
7:          nova_distancia.distancia ← dist.distancia + 1
8:          envia(amigos[i], ∅, ∅, nova_distancia)
9:        fim para
10:     fim se
11:   fim para
12:   envia(id_usuario, amigos, distancias, ∅)
13: fim
14: Reduce(id_usuario, amigos[1..a], distancias, novas_distancias[1..m])
15: começo
16:   para dist in novas_distancias do
17:     se não encontra(distancias[dist.id_semente]) então
18:       distancias[dist.id_semente] ← dist.distancia
19:     fim se
20:   fim para
21:   envia(id_usuario, amigos, distancias)
22: fim
```

Tanto a entrada como a saída do algoritmo de propagação corresponde a um conjunto de pares chave-valor, onde as chaves correspondem a identificadores de usuários e os valores correspondem às listas de amigos e de distâncias até as sementes dos usuários correspondentes. Além dos pares de usuários e listas de amigos e distâncias, provemos também o número da sub-fase executada para a fase *Map*, necessário para que o proces-

samento de cada par seja executado corretamente. Cada par é processado pela função *Map* da seguinte forma. Primeiramente, a função percorre a lista de distâncias até as sementes já encontradas para o usuário, verificando, para cada distância, se seu valor foi criado na sub-fase anterior (no caso do valor 0, sua criação ocorre na marcação de sementes). Nesse caso, a distância deve ser propagada para todos os amigos do usuário processado. Dessa forma, para cada um dos amigos do usuário é criado um par onde a chave é o identificador do amigo e o valor é uma distância até a semente em questão com valor correspondente ao número da sub-fase executada. Além da propagação das distâncias até as sementes, todos os pares recebidos como entrada pela fase *Map* são enviados inalterados para a fase *Reduce*. Como descrito anteriormente, a função *Reduce* recebe como entrada uma chave e um conjunto de valores correspondentes a essa chave. Em nosso algoritmo, a chave corresponde a um identificador de usuário e os valores correspondes a listas de amigos, listas de distâncias e nova distâncias propagadas. O processamento feito pela nossa função *Reduce* consiste em incorporar as novas distâncias à lista de distâncias pré-existentes. Note que podem existir distâncias repetidas para uma mesma semente, já que pode haver mais de um caminho entre dois usuários. Nesse caso, apenas a distância de menor valor é mantida.

5.2.6 Função de Similaridade

Podemos agora usar os vetores de distâncias até as sementes para definir nossa função de similaridade. Como mencionamos anteriormente, a fórmula utilizada para a ordenação de respostas na busca por pessoas em uma rede social não está restrita a utilizar apenas a informação de proximidade no grafo para a ordenação dos resultados da busca. Apesar disso, distâncias sociais são um sinal crítico para a busca de usuários em redes sociais. Dessa forma, o foco deste trabalho será restrito a este único sinal. Funções de similaridade mais abrangentes serão abordadas em trabalho futuros. Assim, utilizaremos uma função de similaridade entre pessoas baseada somente nas distâncias até as sementes, definida a seguir.

variável

Definição 7

$$sim(i,j) = \frac{\sum_{s=1}^{n} k_{d_{is}+d_{js}}}{log(sementes(j))} \tag{5.1}$$

$$\sum_{s=1}^{n} k_{d_{is}+d_{js}} \tag{5.2}$$

onde i é o usuário executando a consulta e j é um usuário entre os resultados da consulta. O número n indica o número de sementes utilizadas e d_{is} indica a distância

do usuário i até a semente s. A soma $d_{is}+d_{js}$ nos dá uma estimativa da distância entre os usuários i e j. Assim, o termo $k_{d_{is}+d_{js}}$ corresponde ao peso dado para a ocorrência de uma uma distância estimada como $d_{is}+d_{js}$. A função $sementes(j)$ indica o número de sementes com distância finita até o usuário j, e é usada como um fator de normalização.

Como foi mostrado na Subseção 5.2.3, as possíveis distâncias estimadas são 1, 2, 3, 4 e ∞. Distâncias infinitas não adicionam informação à função de similaridade. Portanto, k_∞ é fixado com 0. Dessa forma, apenas as distâncias 1, 2, 3, 4 são consideradas em nossa função. A função de similaridade pode então ser representada como se segue.

$$\begin{aligned} sim\_parcial(i,j) &= k_1 * distâncias\_1 + \\ &\quad k_2 * distâncias\_2 + \\ &\quad k_3 * distâncias\_3 + \\ &\quad k_4 * distâncias\_4 \\ sim(i,j) &= \frac{sim\_parcial(i,j)}{log(sementes(j))} \end{aligned}$$

onde $distâncias\_i$ indica o número de sementes cuja soma das distâncias até os usuários i e j é exatamente i. As constantes k_1, k_2, k_3 e k_4 são pesos ajustados empiricamente.

Sabemos que usuários mais próximos devem receber pesos maiores. Dessa forma, queremos escolher os pesos de tal forma que k_1 k_2 k_3 k_4. Além disso, queremos que, em nossa ordenação final, nenhum usuário esteja à frente de um segundo usuário com alguma distância estimada menor que sua distância estimada mínima até o usuário efetuando a consulta. Em nossos experimentos, os valores das constantes k_1, k_2, k_3 e k_4 são fixados em 10^6, 10^4, 10^2 e 1 respectivamente, valores que atendem as restrições descritas acima. Ilustraremos com o Exemplo 4.

Exemplo 4 *Ordenação dos resultados pela função de similaridade definida acima para a consulta 'Maria', feita por John, e vetores de distâncias do Exemplo 2.*

$$\begin{aligned} sim(John, Maria\ A) &= \frac{10^6*0+10^4*1+10^2*2+1*0}{log3} \\ &= 9284.44 \\ sim(John, Maria\ B) &= \frac{10^6*0+10^4*1+10^2*0+1*0}{log3} \\ &= 9102.39 \\ sim(John, Maria\ C) &= \frac{10^6*0+10^4*0+10^2*2+1*0}{log3} \\ &= 182.05 \end{aligned}$$

Nossa função gera a ordenação 'Maria A.', em seguida 'Maria B.' e, por fim, 'Maria C.', o que, neste caso, é a ordenação desejada. Embora funções de ordenação de respostas mais sofisticadas possam ser desenvolvidas, verificamos empiricamente a efetividade da função proposta na Seção 5.3.

5.2.7 Análise de Complexidade de Nossa Solução

Nesta Seção fazemos uma análise de complexidade em tempo e espaço de de nosso algoritmo de cálculo de distâncias até as sementes, fazendo separadamente a análise do passo de *marcação das sementes* e do passo de *propagação de distâncias*. As operações consideradas para a análise serão pesquisas em tabelas *hash*, com complexidade $O(1)$, e pesquisas e inserções em mapas ordenados, com complexidade $O(log(h))$, onde h é o tamanho do mapa. Consideramos ainda a operação de enviar um usuário para a fase seguinte, com complexidade $O(1)$. Por fim, consideramos ordenação de conjuntos de elementos, com complexidade $O(log(h))$. Com essa análise, queremos prever o comportamento dos requisitos de tempo e espaço com o aumento do número de usuários no grafo de uma rede social.

Considere n como o número de usuários no grafo e a o número médio de amigos por usuário. Seja ainda s o número de sementes utilizadas no algoritmo.

Na fase *map* da marcação das sementes, para cada um dos usuários do grafo, executamos uma pesquisa na tabela *hash* contendo as sementes, como é possível verificar no pseudo-código mostrado na Figura 2. Dessa forma, são executadas $2n$ operações. Na fase *reduce*, os n usuários são simplesmente movidos para a fase de propagação. Note que antes da fase *Reduce* é feita a ordenação dos pares. Assim, a complexidade deste passo é $O(n \times log(n))$.

Na fase de propagação das distâncias, podemos ter um número variável de sub-fases de propagação. Seja f o número de sub-fases de propagação (por exemplo, se $f = 2$, propagaremos as sementes por 2 níveis, alcançando até os amigos dos amigos de todas as sementes). Para uma dada sub-fase m, com m variando de 1 a f, (sendo $m = 1$ a fase correspondente à propagação até os amigos diretos da semente), temos o seguinte número de operações:

$$\begin{aligned} op(m) &= 2n + sa^m + sa^m \times (2 \times log\,(s)) + sa^m \times log\,(sa^m) \\ &= 2n + sa^m + 2sa^m \times log\,(s) + sa^m \times log\,(sa^m) \end{aligned}$$

O primeiro termo, $2n$, se refere ao fato de que todos os usuários são enviados uma vez na fase *map* e outra na fase *reduce*. O segundo termo, sa^m, corresponde ao envio dos termos que devem receber novas distâncias na fase *map*, provenientes da propagação

anterior. O termo $sa^m \times (2 \times log\,(s))$ corresponde à pesquisa das novas distâncias nos mapas ordenados, que contêm as distâncias de cada usuário, e à inserção das novas distâncias nos mapas de distâncias, com complexidade dada por $log(s)$. Note que o número de sementes s corresponde ao número máximo de distâncias para um dado usuário. Na prática, para um número pequeno de fases de propagação, como 1 ou 2, esse número é em média muito inferior a s. No entanto, para facilitar o cálculo da complexidade, utilizaremos o número s. Por fim, o termo $sa^m \times log\,(sa^m)$ corresponde à ordenação que precede a fase *Reduce*. Dessa forma, para f fases de propagação, temos:

$$\begin{aligned}
T(n,f) &= \sum_{m=1}^{f} (2n + sa^m + 2sa^m \times log(s) + sa^m \times log\,(sa^m)) \\
&= 2nf + \sum_{m=1}^{f} sa^m + \sum_{m=1}^{f} (2sa^m \times log(s)) + \sum_{m=1}^{f} (sa^m \times log\,(sa^m)) \\
&= 2nf + s \times \sum_{m=1}^{f} a^m + 2s \times log(s) \times \sum_{m=1}^{f} a^m + \sum_{m=1}^{f} (sa^m \times log\,(sa^m)) \\
&= 2nf + (s + 3s \times log(s)) \times \left(\frac{a^{f+1} - a}{a - 1}\right) + \sum_{m=1}^{f} (sa^m \times log\,(sa^m)) \\
&= O\left(nf + s \times log(s) \times a^f + sa^m \times log(sa^m)\right)
\end{aligned}$$

Unindo a fase de marcação de sementes e a fase de propagação, temos a seguinte complexidade:

$$O\left(n \times log(n) + nf + s \times log(s) \times a^f + sa^m \times log(sa^m)\right)$$

No caso do Orkut, como mostrado na Subseção 5.2.3, utilizamos $f = 2$, já que dessa forma conseguimos estimar distâncias até 4, o que é suficiente para nossa aplicação. Assim, nossa complexidade também pode ser expressa como:

$$O\left(n \times log(n) + s \times a^2 \times log(s) + s \times a^2 \times log(s \times a^2)\right)$$

Se desejarmos manter constante o número médio de sementes alcançadas a partir de um dado usuário, s deve ser uma função de n. Na verdade, mesmo se não desejarmos manter esse número constante, sabemos que s é menor ou igual a n e assim nossa complexidade pode ser expressa como:

$$\begin{aligned}
& O\left(n \times log(n) + n \times a^2 \times log(n) + n \times a^2 \times log(n \times a^2)\right) \\
= \; & O\left(n \times a^2 \times log(n \times a^2)\right)
\end{aligned}$$

A última consideração que devemos fazer é a respeito do número médio de amigos a. É esse número que determina a complexidade final de nosso algoritmo. No caso da rede social utilizada em nossos experimentos, o Orkut, o número médio de amigos possui o limite superior de 1000 imposto pelo sistema (embora a média atual seja de 90 amigos) e, assim, a pode ser considerado constante. Dessa forma, nosso algoritmo possui complexidade de tempo $O(n \times log(n))$. Mesmo que não exista um limite no número de amigos que cada usuário pode ter, estudos indicam que esse número tende a se estabilizar ou crescer muito lentamente([17]).

Com relação ao espaço, no passo de marcação das sementes precisamos das listas de amigos dos usuários e da lista de sementes. Considerando que x e y representam o espaço necessário para armazenar o identificador de um usuário ou semente e a informação de distância até uma semente respectivamente, precisamos de um espaço correspondente a $xna+xs$. Ao final desse passo são geradas s distâncias, que requerem um espaço ys. No passo de propagação, cada fase m gera o equivalente a ysa^m. Dessa forma, ao final de f fases, o espaço total utilizado pelo algoritmo é:

$$\begin{aligned} E(n) &= xna + xs + ys + \sum_{i=1}^{f} ysa^m \\ &= xna + xs + ys + ys \times \left(\frac{a^{f+1} - a}{a - 1}\right) \\ &= O\left(n + sa^f\right) \end{aligned}$$

De maneira análoga à análise de complexidade com relação ao tempo, podemos assumir que s é função de n dada por kn, onde $0 < k \leq 1$. Considerando também que nosso f é 2 e a é constante temos:

$$\begin{aligned} & O\left(n + na^2\right) \\ = \; & O(n) \end{aligned}$$

5.3 Experimentação

5.3.1 Ambiente de Experimentação

Para comparar nosso algoritmo com as outras soluções apresentadas, executamos experimentos em um *cluster* de 128 máquinas como ilustrado na Figura 5.2. A arquitetura do *cluster* é semelhante à utilizada em[4], na qual o *broker* é responsável por receber as consultas dos clientes, repassá-las para os servidores responsáveis pelo processamento das consultas, fazer a intercalação dos resultados recebidos dos servidores e, por fim,

retornar os resultados finais das consultas para os clientes.

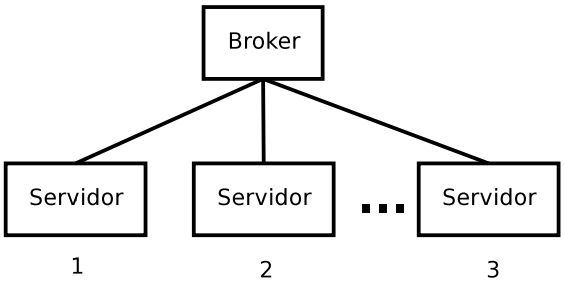

Figura 5.2: *Cluster* de 128 máquinas: processadores de 2GHz x86-64, rede ethernet 1 Gbps, 40M/128 perfis de usuários por máquina.

Para avaliar o desempenho das soluções apresentadas, foram utilizadas 40.000 consultas da busca por usuários do Orkut e, para avaliar a precisão, foram utilizadas 100 consultas. Tanto as consultas quanto os usuários submetendo a consulta foram escolhidos aleatoriamente.

5.3.2 Algoritmos avaliados

Além da solução proposta baseada em sementes, avaliamos dois dos algoritmos descritos na Seção 5.1. O primeiro deles é a *solução força bruta*, que consiste em computar as distâncias entre o usuário buscando por pessoas e os usuários dos resultados em tempo de consulta. O segundo algoritmo avaliado é a *interseção de listas*, que consiste em armazenar as listas de amigos e amigos de amigos de todos os usuários e, em tempo de consulta, computar a interseção dessas listas com a lista de amigos de amigos no usuário efetuando a consulta. A terceira solução descrita na Subseção 5.1.1, *pré-computação das distâncias entre todos os pares*, não foi avaliada devido ao requisitos de espaço, que tornam a solução inviável.

5.3.3 Requisitos de Espaço

Algoritmo Baseado em Distâncias com Relação às Sementes

Uma das grandes vantagens da abordagem baseada proposta, baseada em sementes, é que os requisitos de espaço são consideravelmente inferiores aos requisitos das outras possíveis soluções apresentadas. Os vetores de distâncias até as sementes são bastante esparsos devido a que a)o limite imposto para as distâncias é de no máximo 2 e b)cada distância demanda apenas 3 *bytes* de armazenamento, sendo 22 *bits* para o identificador da semente e 2 *bits* para a distância até a semente. Além disso, a computação é

facilmente distribuível, já que os vetores de distâncias podem ser particionados exatamente da mesma forma que os usuários, dado que existe exatamente um vetor de distâncias associado a cada usuário.

Em nossos experimentos, variamos o número de sementes usando os seguintes valores: 100.000, 500.000, 1.000.000, 2.000.000 e 3.000.000. A Tabela 5.3 mostra o espaço necessário para o armazenamento dos vetores de distâncias até as sementes bem como o número médio de sementes com distâncias finitas para cada usuário. Note que o número médio de sementes com distâncias finitas para cada usuário é bem menor que o total de sementes utilizadas e cresce linearmente com relação ao total de sementes.

# sementes	# médio de sementes	**E**spaço (GB)
*1*00.000	53,28	4,35
*5*00.000	264,97	21,20
1.000.000	532,50	42,49
2.000.000	1.063,83	84,79
3.000.000	1.597,21	127,25

Tabela 5.3: Requisito de espaço para diferentes números de sementes.

Algoritmo Força Bruta

A solução força bruta necessita apenas do grafo de relacionamentos, que ocupa aproximadamente 16GB de espaço. Entretanto, como a computação do caminho mínimo não pode ser facilmente distribuída entre as máquinas, essa solução demanda que cada máquina do *cluster* tenha sua própria cópia do grafo. Como resultado, na prática, os requisitos de espaço são bem maiores que da solução baseada em sementes, já que precisamos de N cópias do grafo. Em nosso caso, $N = 128$. Dessa forma, o espaço necessário para a solução força bruta é $128 * 16GB = 2048GB$.

Algoritmo de interseção de listas

Para a solução de interseção de listas, os requisitos de espaço são descritos a seguir. Cada usuário do Orkut tem em média 90 amigos. O identificador de um usuário possa ser representado por um inteiro de 4 *bytes*. Precisamos armazenar a lista de amigos de amigos para cada um dos 40 milhões de usuários. Dessa forma, a solução de interseção de listas requer aproximadamente $40M * 90 * 90 * 4 = 1.260GB$.

Análise

É possível ver que os requisitos de espaço para as demais soluções apresentadas são muito superiores aos da nossa solução. Enquanto as estruturas auxiliares da nossa solução baseada em sementes podem ser facilmente mantidas na memória disponível em nosso *cluster* ($128 * 4GB = 952GB$), o mesmo não é verdade para as outras solu-

ções. Essa é obviamente uma das vantagens da nossa solução. Entretanto, para fazer com que as avaliações de desempenho sejam justas, usamos a seguinte estratégia para conseguir armazenar as estruturas auxiliares de todos as soluções em memória. Tanto para a solução força bruta quanto para a interseção de listas, criamos um passo anterior de *aquecimento* antes da execução de nossos experimentos. Nesse passo, executamos todas as consultas que serão avaliadas e geramos, para cada solução, uma nova estrutura auxiliar que contém apenas as porções da estrutura original relevantes para as consultas em questão. A nova estrutura, que possui informações suficiente apenas para o cálculo das distâncias para o conjunto de consultas avaliadas, é muito menor que a estrutura original e pode ser facilmente mantida em memória. Por exemplo, se a consulta avaliada contém os usuários 'Maria A.', 'Maria B.' e 'Maria C.' como resultados e 'John' é o usuário que submeteu a consulta, vide Figura 1.1, na solução força bruta armazenaremos somente as partes do grafo necessárias para calcular o caminho mínimo entre 'John' e as três Marias. De maneira análoga, apenas 4 listas de amigos precisariam ser armazenadas para a solução de interseção de listas.

5.3.4 Resultados

Precisão

Nesta seção apresentamos uma avaliação da precisão comparando tanto a solução baseada em sementes quanto a interseção de listas com a solução força bruta, já que esta última gera resultados "perfeitos"de acordo com a nossa noção de relevância. Além disso, é importante ressaltar que não comparamos nossos resultados com uma solução que não leva em conta a distância social entre usuários no grafo, devido aos resultados, nesse caso, não possuírem nenhum critério de ordenação exceto pelo casamento de texto.

Não podemos utilizar métricas tradicionais de comparação de listas de resultados ordenados como *Kendall Tau* simplesmente porque listas muito diferentes podem ser igualmente relevantes de acordo como nossa métrica de distância social. Uma alternativa é a utilização da métrica P@10, uma métrica comumente utilizada para a avalição de sistemas de busca na *Web*. Embora P@10 seja muito útil por ter um significado claro na comunidade de Recuperação de Informação, ainda precisamos definir o conjunto de documentos relevantes contra os quais devemos fazer a comparação de nossos resultados. Uma possível definição para este conjunto são os 10 primeiros documentos retornados pela solução força bruta, o que tem uma grande desvantagem. Suponha as listas de resultados abaixo:

Se aplicarmos a métrica P@10 diretamente, como sugerido acima, a P@10 para a 'Solução de teste' seria 90% (note que o último resultado das duas listas são diferen-

Resultados **F**orça bruta	**D**istância	**R**esultados **S**olução de teste	**D**istância
'Maria A'	1	'Maria A'	1
'Maria B'	1	'Maria B'	1
'Maria C'	2	'Maria C'	2
'Maria D'	2	'Maria D'	2
'Maria E'	2	'Maria E'	2
'Maria F'	2	'Maria F'	2
'Maria G'	2	'Maria G'	2
'Maria H'	2	'Maria H'	2
'Maria I'	2	'Maria I'	2
'Maria J'	2	'Maria K'	2

Tabela 5.4: Exemplos de listas de resultados ordenados.

tes). Entretanto, do ponto de vista do usuário, as duas listas são igualmente relevantes. Dessa forma, propomos uma pequena alteração na definição do conjunto de documentos relevantes. A nova definição, mostrada abaixo, permite a troca de documentos igualmente relevantes, ao mesmo tempo em que mantém todas as propriedades da P@10 padrão.

Definição 8 *O conjunto de documentos relevantes é definido pelos 10 primeiros resultados retornados pela solução força bruta mais todos os resultados seguintes nos quais a distância social é a mesma do décimo resultado.*

A Tabela 5.5 mostra a precisão média obtida pela abordagem baseada em sementes e pela interseção de listas. Por definição, a solução força bruta gera 100% de precisão.

Número de sementes	**P**@10 média (%)
*1*00.000	71,48
*5*00.000	81,98
1.000.000	86,53
2.000.000	90,50
3.000.000	92,08
*I*nterseção de listas	87,02

Tabela 5.5: Precisão em 10 média.

Embora os resultados sejam muito bons, eles ocultam um problema fundamental freqüentemente ignorado por pesquisadores que utilizam métricas de precisão. Suponha que, ao invés dos resultados mostrados na Tabela 5.4, a solução de teste apresentasse os seguintes resultados:

Resultados **F**orça bruta	**D**istância	**R**esultados **S**olução de teste	**D**istância
'Maria A'	1	'Maria K'	2
'Maria B'	1	'Maria B'	1
'Maria C'	2	'Maria C'	2
'Maria D'	2	'Maria D'	2
'Maria E'	2	'Maria E'	2
'Maria F'	2	'Maria F'	2
'Maria G'	2	'Maria G'	2
'Maria H'	2	'Maria H'	2
'Maria I'	2	'Maria I'	2
'Maria J'	2	'Maria A'	1

Tabela 5.6: Lista de resultados ordenados para solução de teste modificada.

Os resultados acima são piores do ponto de vista do usuário de acordo com o critério que estabelecemos de distância social e, no entanto, apresentam a mesma P@10. Para atacar esse problema, além da métrica de P@10 padrão, também apresentamos nossos resultados baseados na *precisão generalizada* proposta por Kekalainen e Järvelin em [19]. Na precisão generalizada, a decisão de relevância não é binária e podemos atribuir pesos para diferentes níveis de relevância. Assim, se apresentarmos os resultados para P@1, P@5 e P@10 utilizando essa nova métrica, seremos capazes de capturar a situação mencionada acima. Mesmo com um valor alto de P@10, os resultados apresentados na Tabela 5.3.4 resultariam em um valor de P@1 mais baixo. A definição da nova métrica de precisão (gPR) é:

Definição 9 *A métrica de precisão gPR é dada por* $gPR = \sum_{i=1}^{n} peso(r_i)/peso(b_i)$, *onde n é o número de resultados,* r_i *é o resultado na posição i e* b_i *é o resultado na posição i da base de comparação. O peso de um resultado não relevante é* 0.

Isso ainda nos deixa com a difícil decisão de quais pesos atribuir para cada distância social. Qual a relevância de um amigo comparada à relevância de um amigo de amigo? Nos resultados a seguir, utilizamos os pesos triviais 1, 2, 3 ,4 e 5 para distâncias sociais de 5 a 1 respectivamente. O ajuste desses pesos é um passo importante mas está fora do escopo deste trabalho e será deixado para trabalhos futuros.

As novas precisões generalizadas gPR@1, gPR@5 e gPR@10 são apresentadas na Tabela 5.7.

Novamente, podemos ver que nosso método é altamente competitivo, principalmente quando levamos em conta os requisitos de tempo e espaço muito menores.

Desempenho

Número de sementes	**g**PR@1(%)	**g**PR@5 (%)	**g**PR@10 (%)
*1*00.000	60,03	57,55	63,37
*5*00.000	72,88	71,71	74,26
1.000.000	78,87	76,65	78,75
2.000.000	85,21	83,30	83,36
3.000.000	87,44	84,12	85,07
*L*ist intersection	81,02	82,23	80,96

Tabela 5.7: Precisão generalizada média em 1, 5 e 10 resultados.

Faremos agora uma comparação do desempenho da solução baseada em sementes proposta com as soluções apresentadas na Seção 5.1. A solução 1 - pré-computação de todas as distâncias, não foi considerada simplesmente porque os requisitos de espaço tornam a experimentação impossível. Para avaliar o desempenho, medimos o tempo médio de execução por consulta para 40.000 consultas aleatórias. A Tabela 5.8 mostra os resultados. Note que os tempos apresentados abaixo se referem apenas à reordenação dos resultados de acordo com as soluções e não incluem o tempo de recuperação dos resultados, que é igual para todas as soluções.

Número de sementes	**T**empo médio de execução(ms)
*1*00.000	1,15
*5*00.000	2,38
1.000.000	2,84
2.000.000	4,89
3.000.000	6,77
*F*orça bruta	2.018
*I*nterseção de listas	202

Tabela 5.8: Tempo médio de execução por consulta para todas as soluções.

Podemos notar que o tempo médio de execução por consulta, para o *cluster* de experimentação, é de 1,15 ms quando usamos 100.000 sementes. Aumentando esse número em cinco vezes (para 500.000 sementes) o tempo de execução dobra (2,38 ms). O aumento o número de seeds para 2.000.000 implica em quadruplicar o tempo médio de execução (4,89 ms).

Os tempos referentes à computação dos vetores de distâncias até as sementes também são de interesse. Não há necessidade dessa fase ser extremamente rápida, já que é executada como uma etapa de indexação. Entretanto, a computação não pode levar, por exemplo, dias para terminar, já que deve ser rápida o suficiente para ser continuamente regerada em um sistema real. A Tabela 5.9 mostra os tempos de execução para

a computação dos vetores de distâncias.

Número de sementes	**T**empo (s)
*1*00.000	205
*5*00.000	367
1.000.000	517
2.000.000	725
3.000.000	1.068

Tabela 5.9: Tempo de execução para o cálculo dos vetores de distâncias.

5.3.5 Análise dos Resultados

Pelas Tabelas 5.5 e 5.8 observamos que a adoção de 100.000 sementes nos leva a uma perda em precisão de aproximadamente 28% mas gera um ganho em desempenho de aproximadamente 1.754 e 175 vezes quando comparado às soluções força bruta e interseções de listas respectivamente. Nesse caso, o número de sementes utilizadas representa apenas 0,25% do total de usuários do grafo utilizado.

Quando aumentamos o número de sementes para 500.000, reduzimos a perda em precisão a menos de 20% ao custo de dobrar o tempo médio de execução e reduzir o ganho em desempenho a 847 e 85 vezes, com relação às soluções força bruta e interseção de listas, respectivamente, o que ainda representa um ganho considerável. Em compensação, a precisão fica neste caso acima de 80%. Aumentando o número de sementes para 2 milhões, o que representa 5% da população de usuários, reduz a perda em precisão a 12% enquanto eleva o tempo médio de execução por consulta a 4,89 ms, reduzindo o ganho em desempenho a um fator de 412 e 41 respectivamente.

Nossos resultados indicam que é possível alcançar excelentes ganhos de desempenho e economia de espaço, mantendo precisões na faixa de 70 a 80% e utilizando um número de sementes que varia entre 0,35 e 5% do total da população de usuários. Esses resultados claramente indicam que nosso algoritmo é uma solução viável para a busca em grandes redes sociais.

Capítulo 6

Conclusões e trabalhos futuros

Redes sociais são uma nova tendência na *Web*. Esse novo espaço impõe uma série de desafios aos pesquisadores, desafios que vão desde problemas de escalabilidade até detecção de comportamento do usuário. Neste trabalho, nos concentramos no problema de melhorar a experiência de busca por usuários em redes sociais. Nossa solução foi avaliada utilizando dados do *site* de relacionamento Orkut, uma rede social com mais de 40 milhões de usuários registrados.

Ao longo do trabalho, mostramos que soluções mais diretas, presentes na literatura, não podem ser utilizadas para resolver o problema na escala do Orkut. Nossa solução, desenvolvida utilizando uma arquitetura de busca distribuída, é capaz de aproximar distâncias de relacionamento, mantendo os custos computacionais e de espaço bastante baixos. Dadas as dimensões do Orkut, a solução que apresentamos é a única que é prática. Este é o primeiro trabalho que utiliza a estrutura do grafo para melhorar a experiência de busca em uma grande rede social.

Existem ainda muitas melhorias que podem ser aplicadas ao nosso trabalho. Uma dessas melhorias é desenvolver uma seleção de sementes mais sofisticada, que possa manter bons índices de precisão com um menor número de sementes, ou ainda melhorar a precisão da solução. Outra extensão direta deste trabalho é o desenvolvimento e a avaliação de novas funções de ordenação de respostas que gerem melhores resultados. Apesar de haver margem para melhorias, os resultados obtidos neste trabalho mostram que a solução já pode ser diretamente aplicada em redes sociais de grande porte.

Um outro problema interessante que pode ser investigado são as características de grafos reais de grandes redes sociais, por exemplo, a distância média entre os usuários e a distribuição das conexões de amizade, além da evolução desses grafos no tempo. O melhor entendimento dessas características pode ajudar a entender melhor os requisitos para melhorar a solução apresentada, adequando-a às características encontradas. Existem ainda diversos problemas abertos relacionados a redes sociais, e este trabalho

é um primeiro passo no sentido de identificar e propor soluções viáveis que resultem em uma melhor experiência para milhões de pessoas que utilizam tais serviços diariamente.

Referências Bibliográficas

[1] R. Baeza–Yates and B. Ribeiro–Neto. *Modern Information Retrieval*. Addison Wesley, 1998.

[2] K. Bharat and M. R. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval*, pages 104–111, Melbourne, AU, 1998.

[3] S. Brin and L. Page. The anatomy of a large–scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[4] B. Cahoon and K. S. McKinley. Performance evaluation of a distributed architecture for information retrieval. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 110–118, New York, NY, USA, 1996. ACM Press.

[5] S. J. Carrière and R. Kazman. Webquery: searching and visualizing the web through connectivity. In *Selected papers from the sixth international conference on World Wide Web*, pages 1257–1267, Essex, UK, 1997. Elsevier Science Publishers Ltd.

[6] S. Chakrabarti, B. Dom, P. Raghavan, S. Rajagopalan, D. Gibson, and J. Kleinberg. Automatic resource compilation by analyzing hyperlink structure and associated text. In *WWW7: Proceedings of the seventh international conference on World Wide Web 7*, pages 65–74, Amsterdam, The Netherlands, The Netherlands, 1998. Elsevier Science Publishers B. V.

[7] T. M. Chan. All-pairs shortest paths for unweighted undirected graphs in o(mn) time. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithms*, pages 514–523, New York, NY, USA, 2006.

[8] E. Cohen and U. Zwick. All-pairs small-stretch paths. In *SODA '97: Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pages 93–102, Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.

[9] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI '04*, pages 137–150, 2004.

[10] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[11] D. Dor, S. Halperin, and U. Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000.

[12] F. egas, M. Wattenberg, and K. Dave. Studying cooperation and conflict between authors with history flow visualizations, 2004.

[13] N. Eiron and K. S. McCurley. Analysis of anchor text for web search. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 459–460, New York, NY, USA, 2003. ACM Press.

[14] A. V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 156–165, Philadelphia, PA, USA, 2005.

[15] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths in graphs. *IEEE Trans. Syst. Sci. and Cybernetics*, SSC-4(2):100–107, 1968.

[16] M. Holzer, F. Schulz, D. Wagner, and T. Willhalm. Combining speed-up techniques for shortest-path computations. *Journal of Experimental Algorithmics*, 10, 2005.

[17] E. M. Jin, M. Girvan, and M. E. J. Newman. The structure of growing social networks. Working Papers 01-06-032, Santa Fe Institute, June 2001. available at http://ideas.repec.org/p/wop/safiwp/01-06-032.html.

[18] K. S. Jones and C. van Rijsbergen. Report on the need for and provision of an "ideal"information retrieval test collection. Technical report, British Library Research and Development Report, 1975.

[19] J. Kekäläinen and K. Järvelin. Using graded relevance assessments in ir evaluation. *Journal of the American Society for Information Science and Technology*, 53(13):1120–1129, 2002.

[20] J. M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM*, 46(5):604–632, 1999.

[21] C. Lampe, N. Ellison, and C. Steinfield. A face(book) in the crowd: social searching vs. social browsing. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 167–170, New York, NY, USA, 2006.

[22] Y. Li. Toward a qualitative search engine. *IEEE Internet Computing*, 2(4):24–29, 1998.

[23] O. A. McBryan. GENVL and WWWW: Tools for Taming the Web. In O. Nierstarsz, editor, *Proceedings of the first International World Wide Web Conference*, page 15, CERN, Geneva, 1994.

[24] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.

[25] M. J. Rattigan, M. Maier, and D. Jensen. Using structure indices for efficient approximation of network properties. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 357–366, New York, NY, USA, 2006.

[26] R. Rosenzweig. Can history be open source?: Wikipedia and the future of the past. *Journal of American History*, 93(1), June 2006.

[27] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[28] G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. Technical report, Ithaca, NY, USA, 1987.

[29] G. Salton and M. J. McGill. *An Introduction to Modern Information Retrieval*. McGraw–Hill, 1983.

[30] G. Salton, A. Wong, and C. Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[31] R. Seidel. On the all-pairs-shortest-path problem. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 745–749, 1992.

[32] E. Spertus, M. Sahami, and O. Buyukkokten. Evaluating similarity measures: a large-scale study in the orkut social network. In *Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 678–684, New York, NY, USA, 2005.

[33] R. L. R. Thomas H. Cormen, Charles E. Leiserson and C. Stein. *Introduction to Algorithms*, chapter Seção 22.2: Breadth-first search, pages 531–539. MIT Press and McGraw-Hill, 2001.

[34] M. Thorup and U. Zwick. Approximate distance oracles. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 183–192, New York, NY, USA, 2001.

[35] I. H. Witten, I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, May 1999.

[36] B. Yuwono and D. L. Lee. Search and ranking algorithms for locating resources on the world wide web. In *ICDE '96: Proceedings of the Twelfth International Conference on Data Engineering*, pages 164–171, Washington, DC, USA, 1996. IEEE Computer Society.

[37] U. Zwick. Exact and approximate distances in graphs - a survey. In *Proceedings of the 9th Annual European Symposium on Algorithms*, pages 33–48, London, UK, 2001.