LUCIANO MACHADO FARDIN
THIAGO BRAGA CARNEIRO

SPRINGS - ALGORITMO PARA DESENHAR GRAFOS

Curitiba

2008

LUCIANO MACHADO FARDIN
THIAGO BRAGA CARNEIRO

SPRINGS - ALGORITMO PARA DESENHAR GRAFOS

Trabalho apresentado ao curso de Bacharelado em Ciência da Computação da Universidade Federal do Paraná, como requisito parcial à obtenção do título de Bacharel em Ciência da Computação.

Orientador:
André Luiz Pires Guedes

UNIVERSIDADE FEDERAL DO PARANÁ

Curitiba

2008

Sumário

Lista de Figuras

1 Introdução

A área de pesquisa de desenho de grafos tem adquirido grande importância devido à necessidade de representar essas estruturas abstratas de modo concreto para que seja possível a visualização e compreensão do grafo que se está querendo representar.

Existem várias aplicações de desenhos de grafos para as mais diversas áreas dentro e fora da ciência da computação. Os exemplos podem ir desde representação de estruturas de dados como árvores até atividades de gerência de projeto ou representação de árvores genealógicas.

É importante lembrar que existem vários critérios para definir se um desenho de grafo tem boa qualidade. E esses critérios variam de acordo com o tipo de informação que o grafo está representando. Como exemplo, um critério importante no desenho de uma árvore é manter a parte à direta da árvore com o mesmo tamanho da parte esquerda, e esse critério só faz sentido neste tipo de representação, sendo completamente inaplicável no desenho de uma topologia de rede, por exemplo.

Para cada tipo de grafo foram desenvolvidas técnicas diferentes a fim de garantir que o desenho traga a melhor representação seguindo os critérios específicos de cada tipo de grafo.

1.1 Objetivo

Este estudo tem como objetivo apresentar uma solução para o desenho de grafos de forma a satisfazer os critérios estéticos de planaridade, ou seja, redução no cruzamento de arestas, e distância entre vértices aproximadamente igual. Esses critérios estéticos são muito importantes para a compreensão e legibilidade da informação que o desenho do grafo está representando.

Para isso será apresentado um algoritmo que é capaz de manipular o posicionamento dos vértices e arestas para que o grafo apresente esses critérios estéticos. Este algoritmo é chamado *Spring* e faz parte de uma classe de algoritmos baseados nos princípios da física da atração e repulsão de corpos para construir desenhos de grafos que apresentem uma estética agradável para facilitar a visualização da estrutura representada. O algoritmo de *Spring* utiliza o princípio

da força elástica e das cargas elétricas nos vértices e arestas para construir grafos de forma equilibrada.

Para a apresentação deste estudo, o trabalho foi organizado da segunte forma: no Capítulo 1 está a apresentação e objetivo do trabalho; no Capítulo 2, são apresentados os conceitos fundamentais da teoria dos grafos e também uma seção sobre o desenho de grafos e suas aplicações; no Capítulo 3 apresentamos o Graphviz como ferramenta de auxílio para a aplicação do algoritmo estudado; no Capítulo 4 trazemos um estudo comparativo entre outros algoritmos especialmente desenvolvidos para o desenho de grafos e também um estudo aprofundado sobre o algoritmo *Spring*; finalmente, no Capítulo 5, trazemos a conclusão do trabalho.

2 *Grafos*

Para que seja iniciada a discussão sobre as técnicas de desenho e representação de grafos, é necessário que se conheça a definição e os conceitos sobre os grafos. A teoria dos grafos é uma das áreas de maior aplicação na ciência da computação, tratando-se de um ramo cada vez mais estudado entre matemáticos e profissionais da computação.

2.1 Definição Geral

Um grafo não é apenas uma representação gráfica de uma função matemática [6]. Um grafo é uma estrutura composta por um conjunto de nós, denominados vértices, e um conjunto de pares não ordenados de vértices distintos, denominados arestas [1]. Um grafo é definido em termos matemáticos da seguinte forma: Um grafo G = (V,A) onde V é um conjunto finito não vazio de vértices e A é um conjunto de arestas, ou seja, pares não-ordenados de elementos distintos de V. A é um subconjunto de V com exatamente dois elementos [7].

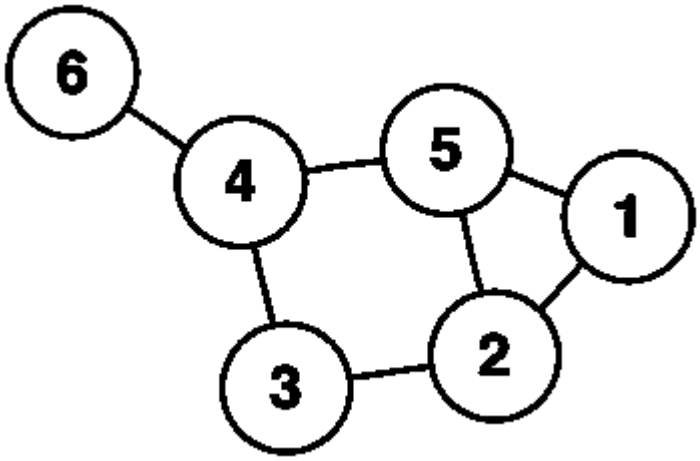

Figura 2.1: Um grafo com 6 vértices e 7 arestas[1]

O grafo acima representado tem como conjunto de vértices V = {1, 2, 3, 4, 5, 6} e como conjunto de arestas E = { {1,2}, {1,5}, {2,3}, {2,5}, {3,4}, {4,5}, {4,6} }. Existem várias

definições para um grafo dependendo das propriedades que os conjuntos de vértices e arestas venham a apresentar. O grafo dado como exemplo é definido como simples, pois não possui mais de uma aresta entre dois vértices quaisquer, além de não possuir laços. Laço é uma aresta que possui as duas extremidades em um mesmo vértice[7].

2.1.1 Grafos Dirigidos

As arestas de um grafo podem ter uma orientação ou sentido. Neste caso o grafo é denominado Grafo Dirigido ou Orientado. Um grafo Direcionado D = (V,A) é um par onde V é um conjunto finito de vértices e A é um conjunto finito de arestas, onde cada aresta "a" $\in$ A é um par ordenado de elementos de V, ou seja, "a" $\in$ V x V. Deste modo, cada aresta possui uma única direção de um certo vértice "v" para um vértice "w"[2].

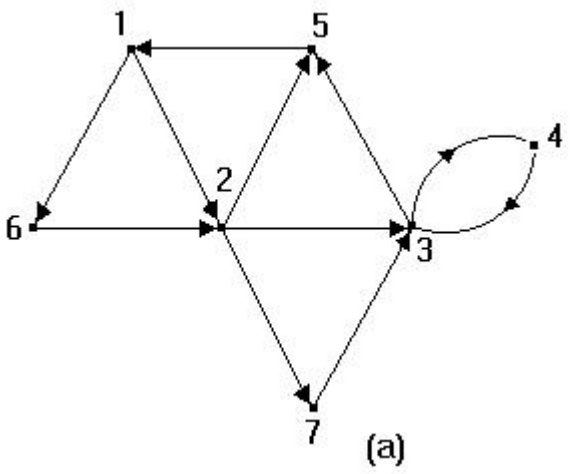

Figura 2.2: Grafo Dirigido[2]

Os grafos podem ainda possuir peso, sendo, neste caso, associado um valor a cada aresta do grafo que representa o custo de se percorrer esta aresta[1]. Esta propriedade é amplamente aplicada a grafos quando se pretende otimizar a criação de rotas.

2.1.2 Grafos Planares

Uma outra propriedade importante para este estudo é a planaridade dos grafos. Pela definição, um grafo planar é aquele que pode ser representado em um plano sem que haja cruzamento de arestas[6]. Essa propriedade deve ser levada em consideração quando se estuda algoritmos de otimização da representação de grafos, como é o caso do algoritmo de *Spring*, pois, quanto

menor a quantidade de intersecção de arestas em um desenho de grafo, melhor será sua legibilidade.

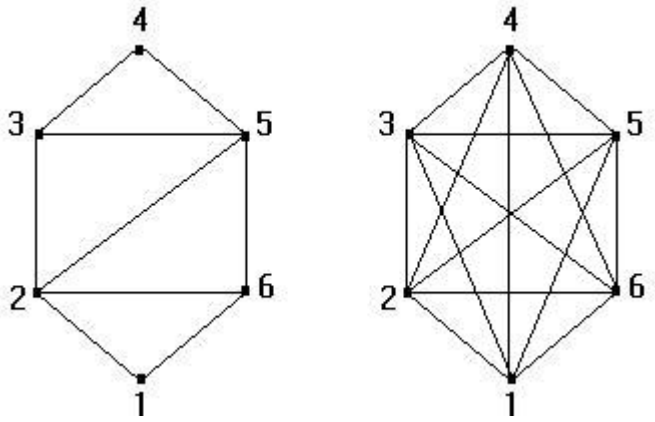

Figura 2.3: Grafo planar à esquerda e Grafo não-planar à direita

2.1.3 Representação de Grafos

Grafos podem ser representados de forma gráfica ou textual. A maneira mais usual de se representar grafos graficamente, como já visto nos exemplos acima, é fazer o uso de círculos como padrão de representação de vértices e linhas com suas extremidades conectadas a vértices para a representação das arestas.

Para a representação de um grafo de maneira textual, não existe padronização para ser adotada, apenas técnicas mais difundidas, como, por exemplo, cada linha ser utilizada para representar uma aresta. Nomes são utilizados para representar os vértices e vírgulas representam a ligação entre eles (a aresta)[7].

Computacionalmente, um grafo pode ser definido utilizando duas estruturas de dados distintas e a forma como essa definição é feita pode ter influência no desempenho da aplicação que faz uso do grafo. Uma das formas mais utilizadas é a matriz de adjacências, que define que, para um grafo G com n vértices, é possivel representá-lo em uma matriz n x n M. As entradas da matriz podem variar de acordo com as propriedades do grafo que se deseja representar. De forma geral o valor M_{ij} guarda informações sobre como os vértices v_i e v_j estão relacionados (isto é, informações sobre a adjacência de v_i e v_j). Para representar um grafo não direcionado, simples e sem pesos nas arestas, basta que as entradas M_{ij} da matriz M contenham 1 se v_i e v_j são adjacentes e 0 caso contrário. Se as arestas do grafo tiverem pesos, M_{ij} pode conter o peso da aresta que houver entre v_i e v_j [8].

Outra estrutura utilizada é a lista de adjacências, que consiste em um vetor com o número

de posições n igual a quantidade de vértices do grafo G. E em cada posição i do vetor existe uma lista ligada contendo os vizinhos de cada vértice v, ou seja, os vértices que têm adjacências com v[9].

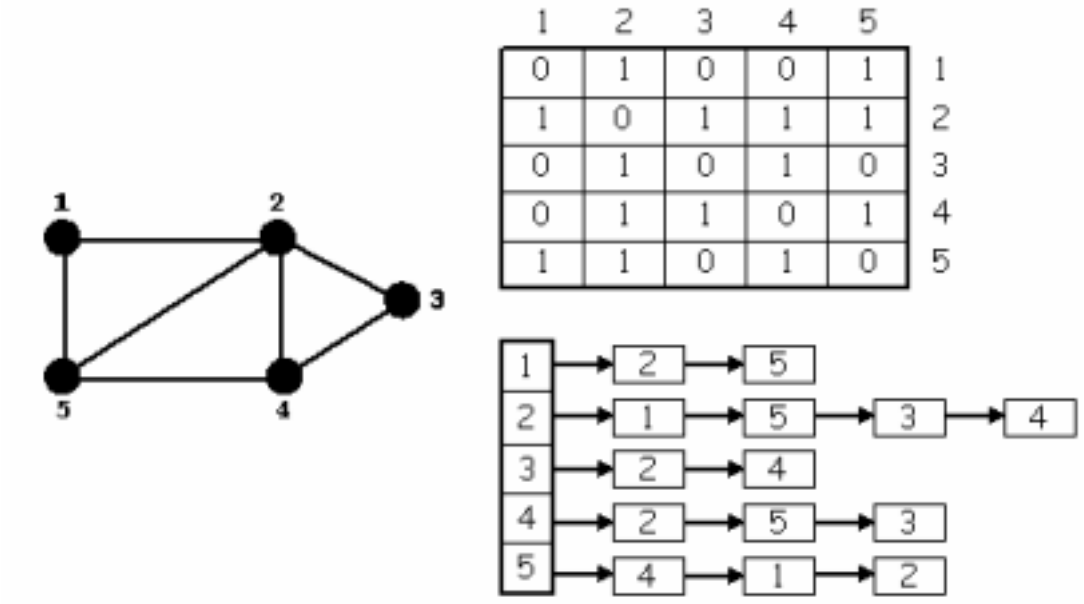

Figura 2.4: Representação do grafo em matriz de adjacências e lista de adjacências

2.2 Desenho de Grafos

O estudo do desenho de grafos está inserido como uma sub-área da Geometria Computacional, e vem ganhando grande destaque nas últimos anos. Esta área de pesquisa combina teoria de topologia de grafos com geometria computacional[7].

O estudo realizado em teoria dos grafos é feito a partir de uma formulação geométrica para se chegar a uma classe de grafos. No estudo de desenho de grafos, o trabalho é no sentido contrário, ou seja, dada uma classe de grafos, deseja-se obter uma geometria que expresse visualmente essa classe.

Este problema, apesar de aparentemente trivial, tem se mostrado em geral bastante difícil, mesmo se for considerado apenas um critério estético. Podemos citar como exemplo o problema de desenhar grafos com o mínimo de cruzamento de arestas e máximo de simetria[10].

2.2.1 Padrão de Desenho

O Padrão de Desenho é definido como sendo um conjunto de características que determinam como os vértices e as arestas são dispostos no espaço.

Os meios mais comumente utilizados para representar vértices são pontos ou círculos e cada aresta(u,v) é representada por uma simples linha ligando os pontos associados aos vértices u e v[7].

Vários padrões para o desenho de grafos têm sido propostos a fim de representar um grafo no plano. Entre os principais, está o padrão poligonal. Nesse padrão, cada aresta é representada por um conjunto de segmentos de retas dispostas de maneira poligonal. O padrão poligonal pode ser modificado de forma que as arestas sejam desenhadas com curvas[7].

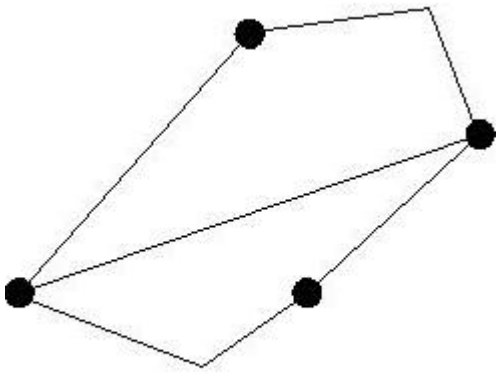

Figura 2.5: Desenho Poligonal

O padrão poligonal pode ser refinado formando dois casos especiais, o linha reta e o ortogonal.

O padrão linha reta é a representação mais comum utilizada em livros de teoria dos grafos. Nesse padrão, cada aresta deve ser representada como um segmento de reta e os vértices como círculos ou outro objeto qualquer[7].

Já o padrão ortogonal mapeia cada aresta em um conjunto de retas verticais e horizontais. Desenhos utilizando o padrão ortogonal são aplicados em várias áreas onde a clareza do desenho é essencial, como as áreas técnicas e de engenharia. Na computação, é comum encontrar este padrão em diagramas de entidade relacionamento no projeto de bancos de dados e na fase de projeto de software.

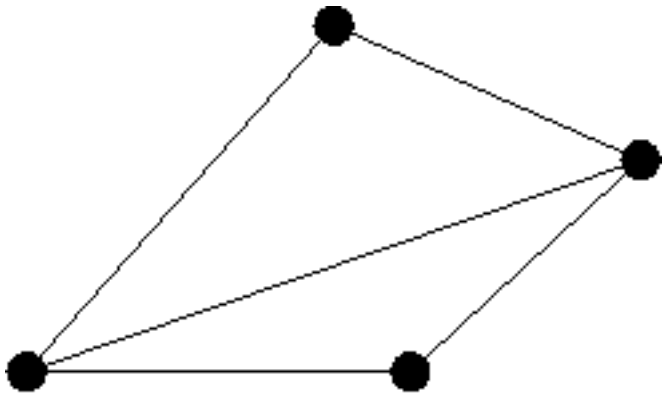

Figura 2.6: Desenho Linha Reta

O padrão ortogonal por sua vez apresenta variações, como é o caso do padrão grade, no qual as coordenadas dos vértices e os cotovelos das arestas possuem valores inteiros. Esse padrão é aplicado no desenho de hardwares e microprocessadores[7].

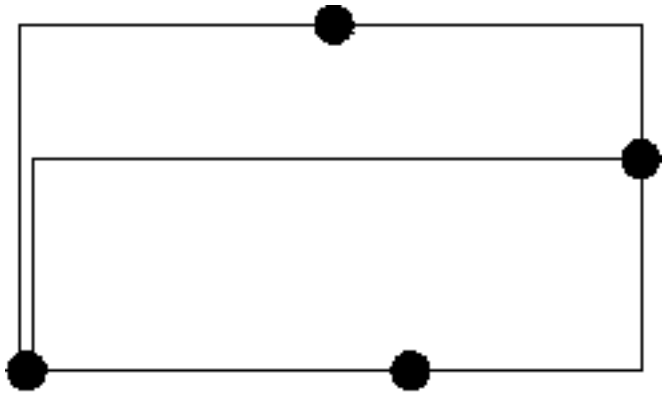

Figura 2.7: Desenho Ortogonal

Por apresentar grande relevância, os grafos em formato de árvore possuem um padrão próprio para sua representação. Tal formato é utilizado para representar estruturas de dados como as árvores de busca, além de representarem estruturas hierárquicas como árvores genealógicas e gráficos organizacionais de empresas. Os padrões utilizados com mais frequência para representar árvores são o linha reta e ortogonal[7].

2.2.2 Características Estéticas

Existem várias maneiras de se desenhar um único grafo, dependendo do padrão de desenho de grafo adotado e da classe do algoritmo utilizado para gerar o grafo[7]. Esses algoritmos são desenvolvidos e otimizados a fim de melhorar a apresentação do desenho do grafo. A

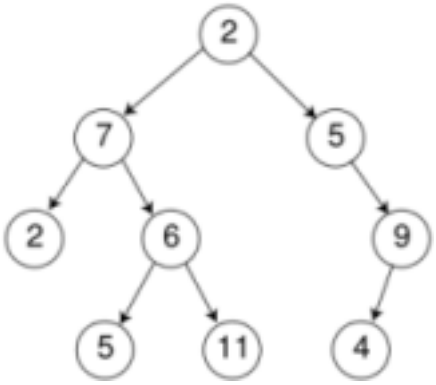

Figura 2.8: Desenho de Árvore
[1]Fonte: http://wiki.sintectus.com/pub/Main/ConcursoProfessorUFG

apresentação (ou estética) do desenho do grafo pode ser dita eficiente se apresenta facilidade de leitura e interpretação dos dados apresentados. Essa eficiência depende também da aplicação para a qual será utilizado o grafo gerado pelo algoritmo.

A característica mais relacionada com um desenho de grafo claro é a redução dos cruzamentos entre arestas. Além disso, é sempre desejável que exista simetria na visualização do grafo.

A figura abaixo mostra como um mesmo grafo pode ser representado de maneiras diferentes, dependendo do padrão estético desejado.

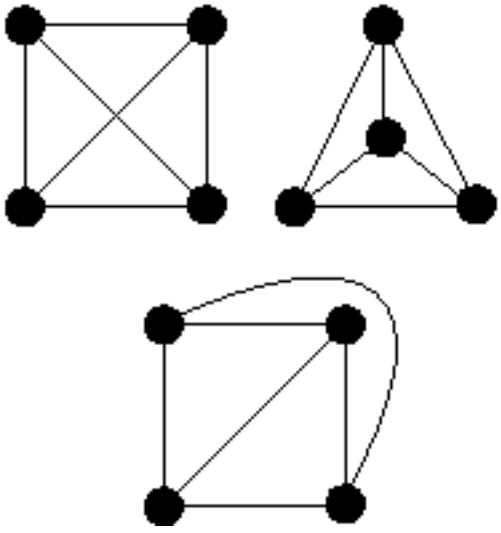

Figura 2.9: Várias representações do mesmo grafo

2.3 Aplicações de Desenho de Grafos

2.3.1 Redes Neurais

Redes Neurais Recorrentes são estruturas de processamento capazes de representar uma grande variedade de comportamentos dinâmicos. A presença de realimentação de informação permite a criação de representações internas e dispositivos de memória capazes de processar e armazenar informações temporais e sinais sequenciais. A presença de conexões recorrentes ou realimentação de informação pode conduzir a comportamentos complexos, mesmo com um número reduzido de parâmetros[3].

Para a visualização do grafo de redes neurais, o critério estético da redução do cruzamento de arestas torna-se dispensável, para que os vértices sejam posicionados de forma a aumentar a legibilidade dos dados. Esta é uma estética muito particular deste tipo de aplicação.

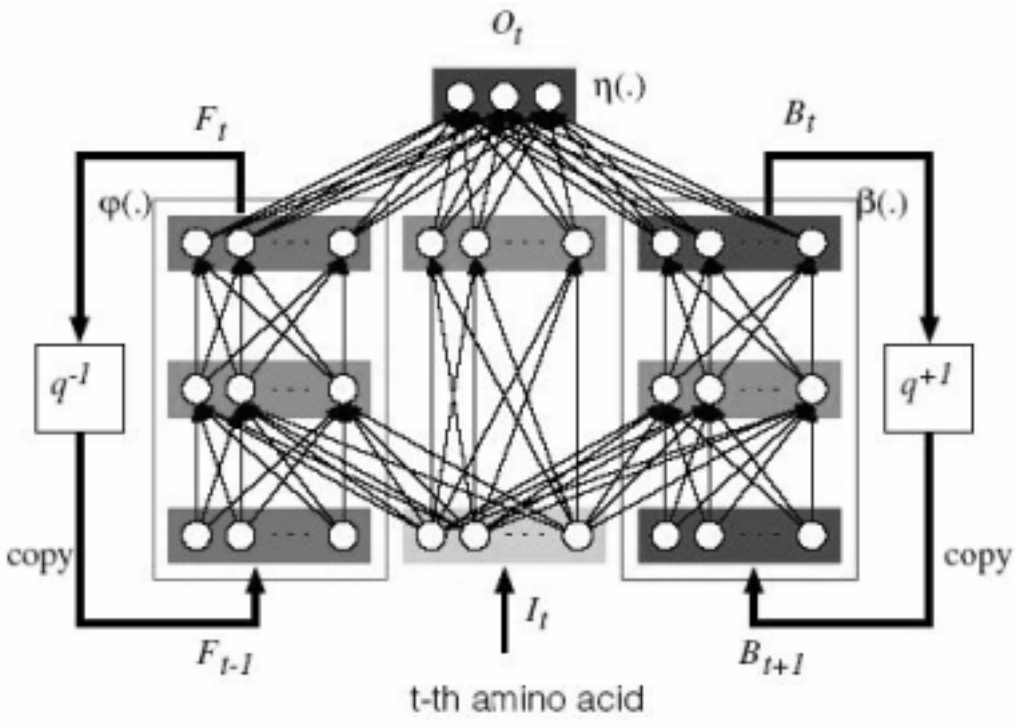

Figura 2.10: Grafo de uma Rede Neural Recorrente[3]

2.3.2 Topologia de Redes

Topologia de redes é a maneira que a rede se apresenta fisicamente, mostrando as conexões existentes. Em grafos que representam topologia de redes, cada vértice representa um dispositivo na rede, podendo ser um computador, hub, gateway, servidor ou outro. As arestas

representam a conexão existente entre os dispositivos.

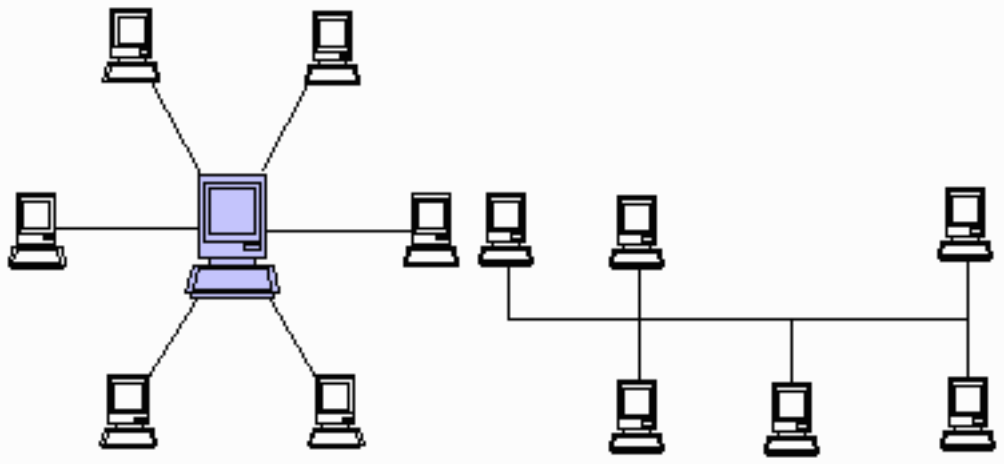

Figura 2.11: Grafos de Diferentes Topologias de Redes

2.3.3 Diagrama de Entidade Relacionamento

Diagrama entidade relacionamento é um modelo diagramático que descreve o modelo de dados de um sistema com alto nível de abstração. Ele é a principal representação do Modelo de Entidades e Relacionamentos. Possui aplicação na Engenharia de Software e também na visualização do relacionamento entre tabelas de um banco de dados, no qual as relações são construídas através da associação de um ou mais atributos destas tabelas[11].

Diagramas de entidade relacionamento são grafos nos quais os vértices possuem formas variadas de acordo com o elemento que se quer representar dentro do relacionamento. Cada aresta representa um relacionamento entre as entidades.

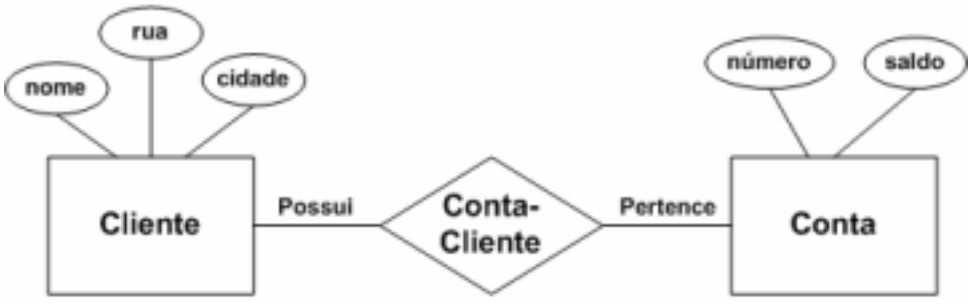

Figura 2.12: Diagrama de Entidade Relacionamento[4]

2.3.4 Diagrama Organizacional

O diagrama organizacional demonstra de forma hierárquica os papéis dentro de uma empresa. Este tipo de diagrama diferencia verticalmente seus papéis em termos de autoridade. Hierarquia é a classificação das pessoas de acordo com autoridade. Papéis no topo da hierarquia organizacional possuem mais autoridade e responsabilidade que os papéis inferiores[12].

O diagrama organizacional é uma das aplicações de desenho de grafos fora da ciência da computação. Utiliza em geral o padrão de desenho em árvore, onde cada vértice representa um papel dentro da empresa ou organização e as arestas representam a relação hierárquica entre os níveis mais altos e os subordinados.

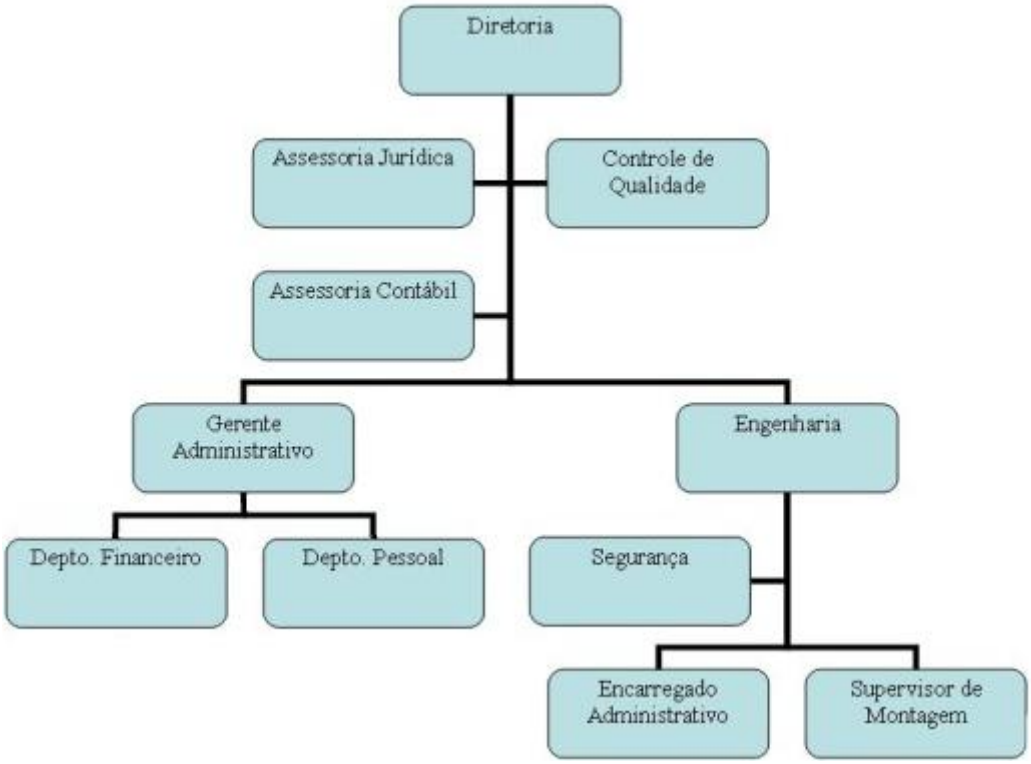

Figura 2.13: Diagrama Organizacional
[2]Fonte: http://www.conenge.net/conteudo/fotos/diagramas/diagrama1.jpg

3 *Graphviz*

O Graphviz é uma coleção de softwares para a visualização e manipulação de grafos abstratos. É capaz de gerar visualização de grafos para aplicativos e *sites* da *web* para as mais diversas áreas, como a engenharia de software, redes, banco de dados, representação do conhecimento e bio-informática.

A essência do Graphviz consiste na implementação de vários padrões de grafos. Estes padrões podem ser utilizados através de uma biblioteca de interface na linguagem C, aplicativos de linha de comandos, interface gráfica e *web browsers*. Os aspectos que diferenciam esse software incluem uma retenção de interfaces baseadas em fluxos em conjunção com uma variedade de ferramentas para a manipulação de grafos, além de suportar uma grande quantidade de ferramentas gráficas e formatos de saída. O primeiro possibilita escrever programas em alto-nível capazes de solicitar, modificar e mostrar um grafo. O último permite ao Graphviz ser útil em várias áreas, indo além das aplicações acadêmicas[13].

3.1 Aplicações

Muitas aplicações utilizam o Graphviz para produzir visualizações de grafos de modo a obter o melhor entendimento de informações e realizar alguma tarefa visualmente. Em particular, o modelo de fluxo do Graphviz permite que ele possa ser inserido em aplicações que necessitam de um serviço externo de visualização de grafos com uma interface gráfica ou *web*. É muito simples gerar modelos de grafos na linguagem *dot* e então carregá-los em um visualizador Graphviz, ou então em conteúdos *web*.

Podemos citar as áreas abaixo como exemplo de sucesso na utilização do Graphviz:

- Engenharia de Software
- Bio-informática
- Internet e estruturas de redes

3.2 Implementação

A implementação do Graphviz reflete a época em que sua parte principal foi escrita, no começo dos anos 90. A maior parte do Graphviz foi escrita em C. As bibliotecas que dão suporte a ele somam aproximadamente quarenta e cinco mil linhas de código.

O *design* do Graphviz incorpora um nível de interface que possibilita o processamento de filtros para o uso de liguagens de *script*. Apesar de possuir uma bibliteca de API, assim como uma interface de usuário, a interface de *scripts* torna o *software* ainda mais útil.

Outro aspecto que distingue o Graphviz de outros *softwares* é a ênfase em prover ao usuário uma grande quantidade de primitivas gráficas e formatos de saída, implementando vários padrões para visualizar um grafo abstrato, formando uma visualização concreta na qual o usuário pode escolher como as informações semânticas e os atributos contextuais serão codificados, etc. Existe uma grande variedade de formatos para serem atribuídos aos vértices, assim como vários tipos de setas e linhas para as arestas.

3.2.1 Arquitetura

O Graphviz possui uma arquitetura convencional em camadas. No centro está a coleção de bibliotecas. A biblioteca *libgraph* provê o modelo fundamental do grafo, implementando grafos, vértices, arestas e subgrafos, assim como seus atributos, além de funções para arquivos de entrada e saída. A biblioteca libcdt é utilizada para implementar operações como o espalhamento de árvores. O Graphviz possui uma versão da biblioteca libgd que permite ao *software* gerar uma saída em *bitmap* em vários formatos. Há também uma biblioteca responsável por separar os grafos em componentes conexas, e depois combinar as componentes em um único desenho.

No próximo nível, há o centro da biblioteca de desenho de grafo. Isto encapsula as partes em comum de todos os algoritmos de desenho de grafos. Esta biblioteca é responsável por receber um grafo da entrada, configurar as estruturas de dados comuns e os atributos, e disponibilizar os *drivers* para todos os tipos de saída suportados pelo Graphviz.

O próximo nível consiste em um programa *stand-alone*. Com as bibliotecas fornecidas, este é basicamente o fluxo principal, que processa uma linha de comando e então utiliza a biblioteca apropriada para ler, gerar e mostrar um grafo.

O nível superior dos visualizadores de grafos e editores é construído, em sua maioria, em linguagens comuns e interface gráfica, utilizando o Graphviz como um co-processo.

3.3 NEATO

O NEATO é uma ferramenta que desenha grafos não-dirigidos, que são muito comuns em telecomunicação e programa de computadores. O NEATO desenha um grafo construindo um modelo físico virtual e tenta de maneira iterativa encontrar a configuração com menor energia[5].

Isto é feito substituindo as arestas por molas ideais, de modo que seu tamanho seja o menor caminho entre os vértices. As molas empurram os vértices de modo que suas distâncias geométricas no desenho se aproximem de suas distâncias correspondentes no grafo, gerando um *layout* razoável.

O NEATO é compatível com o programa DOT, para desenho de grafos dirigidos, compartilhando o mesmo tipo de formato de arquivo de entrada. Desde que o formato do arquivo inclua grafos dirigidos e não-dirigidos, o NEATO desenha grafos preparados para o DOT, e vice-versa. Os dois programas têm as mesmas opções de rótulos, cores, formatos, fontes de texto, paginação e geração de código em linguagens gráficas comuns. Ambos funcionam com o DOTTY, um visualizador de grafos.

3.3.1 Eliminação de Sobreposição

Para melhorar o entendimento, muitas vezes é necessário eliminar a sobreposição de vértices e arestas. Uma maneira de fazer isso é aumentando a escala do *layout* o quanto for necessário. Esta transformação preserva as relações geométricas, mas, nos piores casos, pode requerer fatores escalares muito altos.

Uma outra maneira de se eliminar a sobreposição de vértices é utilizando heurística iterativa. Em cada iteração, um diagrama de Voronoi do ponto central do vértice é computado, e cada vértice é movido para o centro de sua célula de Voronoi. Isso é repetido até que todas as sobreposições sejam eliminadas.

A sobreposição de arestas pode ser prevenida utilizando curvas para desenhar as arestas.

3.3.2 Exemplos

Nesta sessão, serão apresentadas figuras de grafos gerados através do NEATO. A figura 3.1 foi derivada de um tutorial de um sistema operacional. A figura 3.2 mostra a conectividade de uma rede de computadores. A figura 3.3 mostra o compartilhamento de tipos definidos pelo

programador entre procedimentos em um programa em C.

```
graph G {
    run -- intr;
    intr -- runbl;
    runbl -- run;
    run -- kernel;
    kernel -- zombie;
    kernel -- sleep;
    kernel -- runmem;
    sleep -- swap;
    swap -- runswap;
    runswap -- new;
    runswap -- runmem;
    new -- runmem;
    sleep -- runmem;
}
```

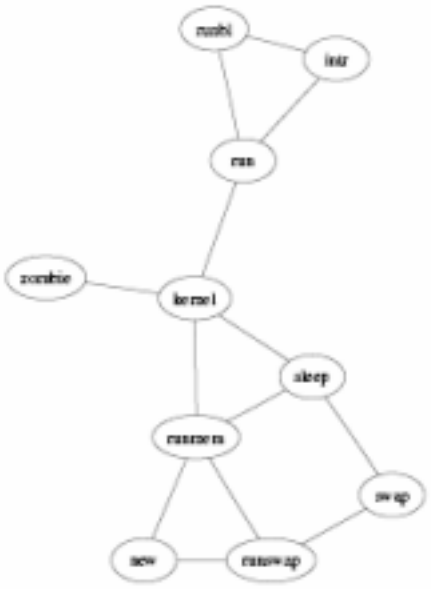

Figura 3.1: Estados de Processos em um Kernel de um Sistema Operacional[5]

Figura 3.2: Backbone da Internet[5]

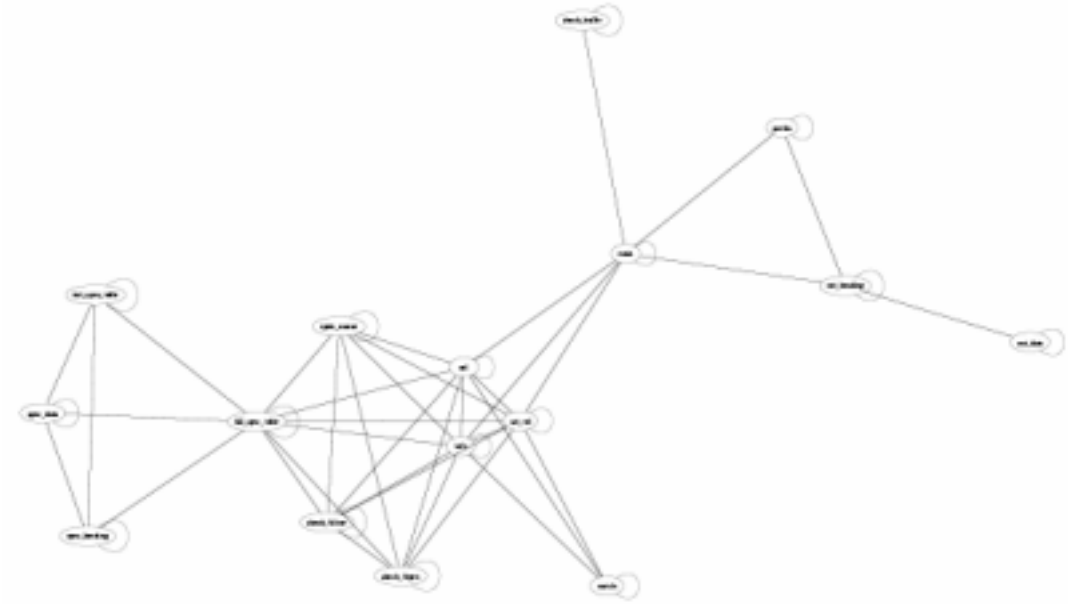

Figura 3.3: Compartilhamento de Tipos Entre Procedimentos em um Programa em C[5]

4 Algoritmos de Desenho de Grafos

4.1 Outros algoritmos

Existem vários algoritmos para desenhar grafos. Cada algoritmo é baseado em um padrão de desenho e possui suas aplicações, vantagens e desvantagens. Citemos alguns tipos de algoritmos:

- **algoritmo de planarização (método baricentro):** retorna um grafo planar. Ele coloca todo vértice no baricentro de seus vizinhos, através da formulação e resolução das equações lineares correspondentes. Contudo, o tempo de execução pode chegar a $O(n^3)$ para grafos que não sejam pequenos (entre outros motivos, o fato de ser difícil reduzir a complexidade da minimização de cruzamentos de arestas), a área de desenho pode se tornar exponencial (deixando vértices muito próximos um do outro), a resolução angular é ruim e a implementação não é simples[14].

- **algoritmo de simulação de temperatura:** o grafo é uma substância cuja temperatura é alterada e os vértices são as partículas da substância. Se a temperatura do grafo é reduzida, os vértices são aproximados; se a temperatura aumenta, o grafo se expande. É útil para o design de circuitos VLSI, particionamento de grafos e o problema do caixeiro viajante. Contudo, tal algoritmo não desenha certos tipos de grafos de maneira convencional. Também não é bom para verificar a regularidade de um grafo[15][16].

- **algoritmo de Sugiyama:** divide o grafo em camadas, sendo que cada camada só forma aresta com as camadas imediatamente inferior e superior. O algoritmo se divide em várias etapas, sendo algumas delas muito complexas[17].

- **algoritmo ortogonal:** as arestas são representadas como se fossem sequências de segmentos de reta verticais e horizontais. Os grafos desenhados são uniformes e claros, tendo poucos cruzamentos de arestas e ocupando uma área pequena. São usados em

áreas técnicas, na engenharia e também em diagramas entidade-relacionamento no projeto de bases de dados. Contudo, assim como em outros algoritmos, é difícil reduzir a complexidade da minimização de cruzamentos de arestas[18].

- **algoritmo dinâmico:** desenha o grafo dinamicamente, permitindo inclusões e remoções de vértices. O problema é testar a planaridade do grafo de forma dinâmica, bem como prover várias outras operações em tempo polilogarítmico[19].

4.2 Algoritmo de Spring

O algoritmo escolhido para esse trabalho foi o algoritmo de Spring (massa-mola), um tipo de algoritmo baseado em força que trata os vértices como partículas eletricamente carregadas e arestas como molas. A carga elétrica dos vértices faz com que haja forças de repulsão entre eles (de acordo com a lei de Coulomb), enquanto as arestas provocam uma força de atração entre seus extremos (de acordo com a lei de Hooke).

Uma vez definidos os vértices (bem como suas cargas elétricas e respectivas posições) e as arestas (com suas constantes elásticas), o algoritmo de Spring promove várias iterações, calculando as forças resultantes em cada vértice e movendo-os de acordo com elas, até que se chegue a um estado de equilíbrio, ou seja, até os vértices não mudarem mais de posição.

O algoritmo de Spring foi escolhido por suas vantagens:

- o desenho final do grafo é simétrico (vértices distribuídos de maneira uniforme, arestas de mesmo tamanho), o que é difícil de se obter a partir de outros algoritmos;
- é ideal para testes e simulações de sistemas físicos ou mecânicos;
- é um algoritmo simples (a idéia principal é fácil de entender) e que pode ser implementado com poucas linhas de código;
- é flexível (com as devidas modificações, pode ser usado para estudar várias propriedades);
- não exige muitos conhecimentos de teoria dos grafos.

Contudo, como todo algoritmo de desenho de grafos, o algoritmo de Spring possui desvantagens, que são:

- o fato das posições iniciais dos vértices exercerem uma influência muito grande no desenho final;

- dependendo do tamanho do grafo, podem ser necessárias muitas iterações para se chegar ao estado de equilíbrio.

4.3 Implementação

Nesta sessão vamos descrever os passos e as decisões que tomamos ao longo do desenvolvimento de nosso trabalho, bem como explicar, de uma forma resumida, o que nosso algoritmo faz.

No início, pesquisamos algoritmos de desenho de grafos, buscando conhecer as aplicações, vantagens e desvantagens de cada um. Depois, buscamos ferramentas que pudessem nos auxiliar no desenho dos grafos, já que nosso trabalho consistiria não no desenho, mas no cálculo das posições dos vértices com base nas forças atuantes sobre eles. Escolhemos o Graphviz, por se tratar de um pacote bastante completo e utilizado em várias áreas, para representação dos mais variados tipos de grafos. Mais especificamente, optamos pelo Dot e pelo Neato, tendo aprendido a escrever arquivos em formato DOT que contivessem a especificação de nossos grafos.

Para programar nosso algoritmo, optamos pela linguagem C, dada a nossa familiaridade com a mesma. Começamos implementando um parser que lesse um arquivo DOT e extraísse os dados do grafo para uma estrutura onde poderíamos manipulá-lo. Depois fizemos o oposto: uma função que lesse a estrutura e escrevesse um novo arquivo DOT com a especificação do grafo alterado. Para facilitar a implementação dessas funções, definimos padrões de formato dos arquivos DOT que leríamos e escreveríamos.

Em seguida, partimos para a implementação do algoritmo de Spring propriamente dito. Basicamente, percorremos a nossa estrutura que armazena os dados do grafo e, para cada vértice, calculamos as forças atuantes sobre o mesmo e definimos sua nova posição. Isso é feito dentro de um loop de 1000 iterações, ou seja, não calculamos a força resultante de todo o grafo para verificar se o mesmo já se encontrava em estado de equilíbrio.

Para calcular a distância entre dois vértices, usamos a distância euclidiana. O resultado é comparado com uma constante que definimos como a constante elétrica, cujo valor é o mesmo para todos os vértices do grafo. Se a distância euclidiana for menor que a constante elétrica, então afastamos o vértice usado como parâmetro. Tendo feito isso para esse mesmo vértice em relação a todos os demais, passamos então a verificar a constante elástica, que é a mesma

para todas as arestas do grafo. Novamente, recorremos à distância euclidiana para verificar se a mesma estaria maior do que a constante elástica; em caso afirmativo, aproximamos o vértice em questão de seu vizinho. E tudo isso é feito em 1000 iterações, onde, em cada uma delas, verificamos todos os vértices em relação aos demais e, em seguida, em relação aos seus vizinhos.

No início da implementação do algoritmo de Spring, procuramos compreender as leis de Hooke e de Coulomb, utilizadas no cálculo das forças atuantes em um sistema massa-mola. Inclusive procuramos implementar funções que calculassem tais forças com base nessas leis; no entanto, os primeiros resultados obtidos nos pareceram bem estranhos. Como não tínhamos muito conhecimento sobre essas leis da Física, não queríamos atrasar o projeto por isso, e, ao mesmo tempo, o cálculo das forças nos parecia bastante intuitivo, decidimos ignorar essas leis e ir, aos poucos, refinando as contas feitas, com base nos resultados parciais. Ao longo dos testes, também fomos alterando os valores usados como as constantes elétrica e elástica.

Para podermos testar nosso algoritmo, procuramos escrever alguns arquivos DOT bastante simples e passá-los ao nosso executável. Ao gerar um novo arquivo DOT, comparávamos os desenhos do arquivo original e desse novo arquivo, para então tentarmos compreender o que o algoritmo havia feito de certo e de errado. Com base nesses resultados, fomos alterando o cálculo da atração/repulsão de vértices e as constantes do nosso trabalho. Em alguns momentos, quando nos aproximávamos de nossos objetivos, alterávamos os arquivos DOT, fosse inserindo e removendo vértices e arestas ou mesmo modificando as coordenadas iniciais de cada vértice, posto que a configuração inicial do grafo exerce grande influência no desenho final.

4.4 Resultados Obtidos

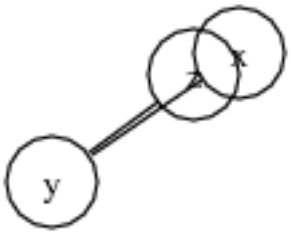

Figura 4.1: Grafo 1 antes

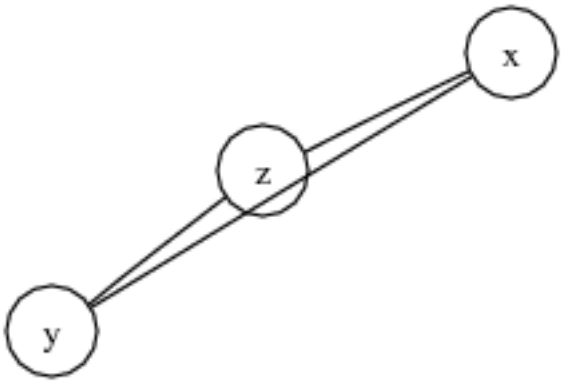

Figura 4.2: Grafo 1 depois

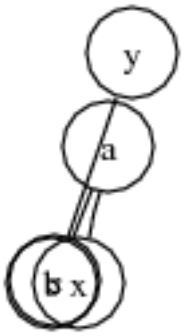

Figura 4.3: Grafo 2 antes

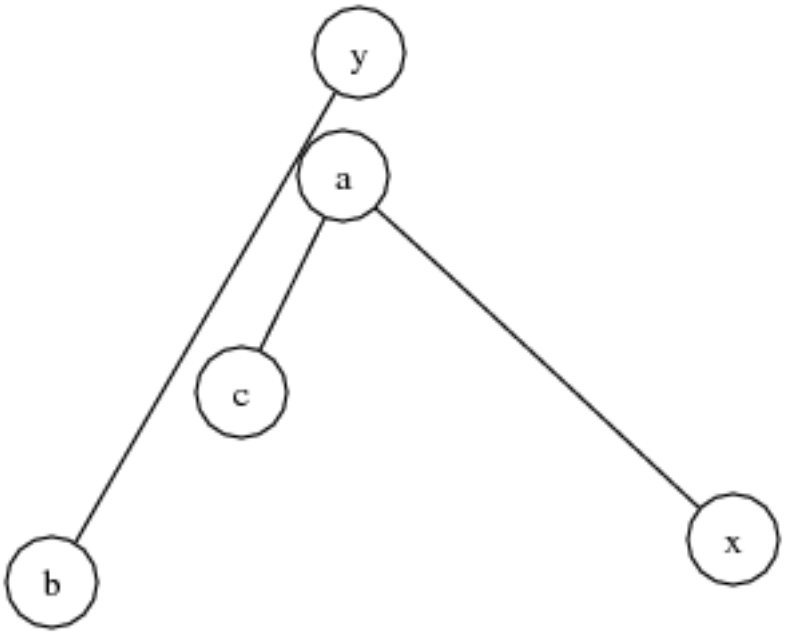

Figura 4.4: Grafo 2 depois

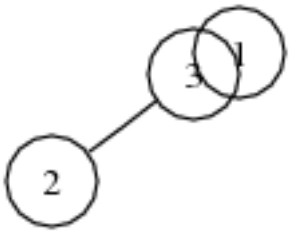

Figura 4.5: Grafo 3 antes

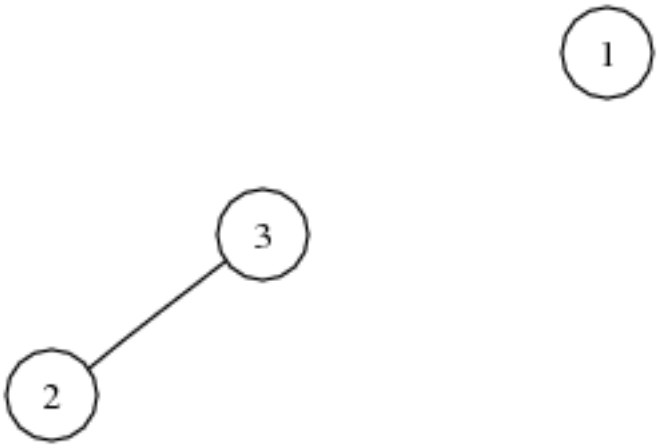

Figura 4.6: Grafo 3 depois

Figura 4.7: Grafo 4 antes

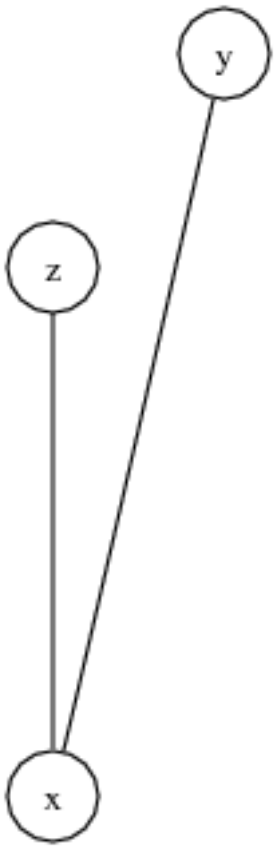

Figura 4.8: Grafo 4 depois

Figura 4.9: Grafo 5 antes

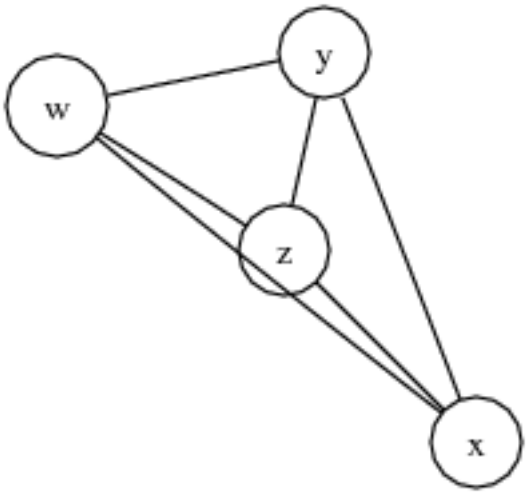

Figura 4.10: Grafo 5 depois

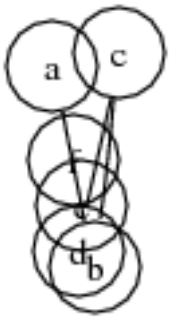

Figura 4.11: Grafo 6 antes

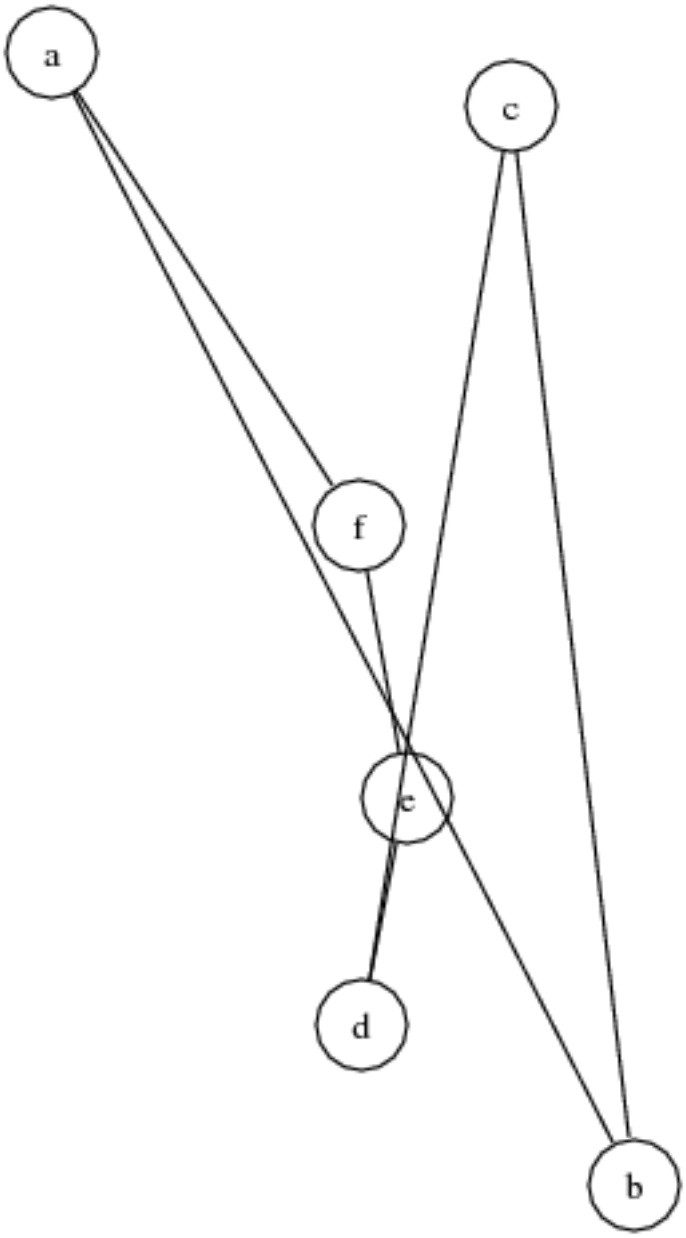

Figura 4.12: Grafo 6 depois

Figura 4.13: Grafo 7 antes

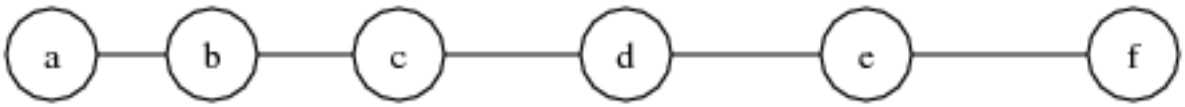

Figura 4.14: Grafo 7 depois

5 *Conclusão*

O desenvolvimento desse trabalho foi interessante para aprendermos sobre a área de desenho de grafos. No início, não tínhamos idéia que havia congressos sobre o assunto, nem que houvesse tantos algoritmos para tal função.

O fato de haver tantas pessoas que estudam algoritmos e ferramentas para desenhar grafos nos proporcionou vários materiais que serviram de base para a execução desse trabalho. Dessa forma, pudemos conhecer e analisar cada tipo de algoritmo, suas características particulares, vantagens e desvantagens.

O Algoritmo de Spring chamou a nossa atenção por tratar grafos como sistemas físicos, com forças de atração e repulsão agindo sobre cada vértice. Nunca havíamos pensado nesse tipo de aplicação, e pudemos assim perceber que os grafos são úteis em diversos tipos de situações nas mais variadas áreas do conhecimento.

Em relação às leis de Hooke e Coulomb, tivemos algumas dificuldades para implementá-las, talvez pelo fato de termos pouco conhecimento em Física. Com isso, fomos calculando as forças de atração e repulsão de outras formas, a princípio com um raciocínio bastante simplório (incrementar ou decrementar as coordenadas de um dos vértices se a distância fosse menor ou maior que a desejada) e, no desenrolar do trabalho, fomos refinando nossos cálculos à medida em que nos aproximávamos daquilo que buscávamos.

Por fim, gostaríamos de dizer que a implementação desse algoritmo foi, ao mesmo tempo, uma diversão e um desafio. Uma diversão no sentido em que manipular e desenhar grafos foi uma atividade, em alguns momentos, prazerosa. Ao mesmo tempo, um desafio, pois não foi fácil fazer o algoritmo deixar os grafos do modo como gostaríamos, com algumas características importantes que o resultado final deveria ter, tais como vértices afastados entre si, arestas de tamanhos semelhantes e com o mínimo possível de cruzamentos, entre outras. Infelizmente, nosso algoritmo não atingiu a perfeição nesses critérios estéticos, mas fizemos o que pudemos para chegar o mais próximo possível ao nosso objetivo.

Referências Bibliográficas

[1] TEORIA dos Grafos. 2008. [Acessado em 10/09/2008]. Disponível em: <http://pt.wikipedia.org/wiki/Teoria_de_grafos>.

[2] INSTITUTO de Ciências Matemáticas e de Computação. Departamento de Computação e Estatística. 2008. [Acessado em 10/09/2008]. Disponível em: <http://www.icmc.sc.usp.br/manuals/sce183/gfdig.html>.

[3] JUNIOR, F. do N.; AZEVEDO, R. R. de. Rede neural recorrente bidirecional enriquecida por grafos de interações para predição de estruturas secundárias de proteínas. *VII Encontro Regional de Matemática Aplicada e Computacional*, 2007.

[4] FUNDAMENTOS de Armazenamento e Manipulação de Dados. 2008. [Acessado em 11/09/2008]. Disponível em: <http://www.ime.usp.br/∼andrers/aulas/bd2005-1/aula6.html>.

[5] NORTH, S. C. *Drawing graphs with NEATOs*. [S.l.], 1991.

[6] TEORIA dos Grafos e Aplicações. 2008. [Acessado em 08/07/2008]. Disponível em: <http://www.dimap.ufrn.br/∼dario/arquivos/Cap2_Grafos-2001.pdf>.

[7] STIVANIN, Z. *Traçado Automático de Hipergrafos Direcionados*. [S.l.: s.n.], 2006.

[8] MATRIZ de Adjacência. 2008. [Acessado em 10/09/2008]. Disponível em: <http://pt.wikipedia.org/wiki/Matriz_de_adjacência>.

[9] SISTEMAS Informatizados. 2008. [Acessado em 11/09/2008]. Disponível em: <http://www26.brinkster.com/provisorio/gsi/09aula07a.htm>.

[10] NASCIMENTO, H. A. D. do; NETO, C. F. X. de M.; SOUZA, P. S. de. *Sinergia em Desenho de Grafos Usando Springs e Pequenas Heurísticas*. [S.l.], October 1996.

[11] DIAGRAMA entidade relacionamento. 2008. [Acessado em 11/09/2008]. Disponível em: <http://pt.wikipedia.org/wiki/Diagrama_Entidade_Relacionamento>.

[12] TEORIA das Organizações. 2008. [Acessado em 11/09/2008]. Disponível em: <http://www.pr.gov.br/batebyte/edicoes/2002/bb122/teoria.htm>.

[13] GANSNER, E. R.; NORTH, S. C. An open graph visualization system and its applications. *Software - Practice and Experience*, v. 30, p. 1203–1233, 1999.

[14] CARSON, D. On o(n²) algorithms for planarization. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, v. 12, p. 1300–1302, 1999.

[15] CERNY, V. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, v. 45, p. 41–51, 1985.

[16] ZHAO, Y. et al. An efficient simulated annealing algorithm for graph bisectioning. *Symposium on Applied Computing*, p. 65–68, 1991.

[17] SEEMANN, J. Extending the sugiyama algorithm for drawing uml class diagrams: Towards automatic layout of object-oriented software diagrams. *Lecture Notes In Computer Science*, v. 1353, p. 415–424, 1997.

[18] ADENEY, K. M.; KORENBERG, M. J. An easily calculated bound on condition for orthogonal algorithms. *IEEE-INNS-ENNS international joint conference on neural networks*, v. 3, p. 620–624, 2000.

[19] CUI, L. L. e J.-H. Reducing multicast traffic load for cellular networks using ad hoc networks. *IEEE Transactions on Vehicular Technology*, v. 55, p. 822–830, 2006.

[20] JENSSEN, T.-K. et al. A literature network of human genes for high-throughput analysis of gene expression. *Nature Genetics*, v. 28, p. 21–28, 2001.

[21] TAKAHASHI, T. O. e H. A graph-planarization algorithm and its application to random graphs. *Lecture Notes in Computer Science*, v. 108, p. 95–107, 1981.