SENAI CIMATEC

PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM COMPUTACIONAL E TECNOLOGIA INDUSTRIAL

Mestrado em Modelagem Computacional e Tecnologia Industrial

Dissertação de mestrado

Visualização de Informação em Redes Sociais e Complexas Utilizando GuaráScript

Apresentada por: Rodrigo Paixão Vilas Bôas
Orientador: Hernane Borges de Barros Pereira
Co-orientador: Roberto Luiz Souza Monteiro

Dezembro de 2016

Rodrigo Paixão Vilas Bôas

Visualização de Informação em Redes Sociais e Complexas Utilizando GuaráScript

Dissertação de mestrado apresentada ao Programa de Pós-graduação em Modelagem Computacional e Tecnologia Industrial, Curso de Mestrado em Modelagem Computacional e Tecnologia Industrial do SENAI CIMATEC, como requisito parcial para a obtenção do título de **Mestre em Modelagem Computacional e Tecnologia Industrial**.

Área de conhecimento: Interdisciplinar

Orientador: Hernane Borges de Barros Pereira
SENAI CIMATEC

Salvador
SENAI CIMATEC
2016

Ficha catalográfica elaborada pelo Centro Universitário SENAI CIMATEC

B662v Bôas, Rodrigo Paixão Vilas

Visualização de informação em redes sociais e complexas utilizando guaráscript / Rodrigo Paixão Vilas Bôas. – Salvador, 2018.

87 f. : il. color.

Orientador: Prof. Dr. Hernane Borges de Barros Pereira.
Coorientador: Prof. Dr. Roberto Luiz Souza Monteiro.

Dissertação (Mestrado em Modelagem Computacional e Tecnologia Industrial) – Programa de Pós-Graduação, Centro Universitário SENAI CIMATEC, Salvador, 2018.
Inclui referências.

1. Algoritmos de visualização. 2. Software open source. 3. Visualização de informação. 4. Visualização de redes. I. Centro Universitário SENAI CIMATEC. II. Pereira, Hernane Borges de Barros. III. Monteiro, Roberto Luiz Souza. IV. Título.

CDD: 620.00113

NDI - 06

Nota sobre o estilo do PPGMCTI

Esta dissertação de mestrado foi elaborada considerando as normas de estilo (i.e. estéticas e estruturais) propostas aprovadas pelo colegiado do Programa de Pós-graduação em Modelagem Computacional e Tecnologia Industrial e estão disponíveis em formato eletrônico (*download* na Página Web http://ead.fieb.org.br/portal_faculdades/dissertacoes-e-teses-mcti.html ou solicitação via e-mail à secretaria do programa) e em formato impresso somente para consulta.

Ressalta-se que o formato proposto considera diversos itens das normas da Associação Brasileira de Normas Técnicas (ABNT), entretanto opta-se, em alguns aspectos, seguir um estilo próprio elaborado e amadurecido pelos professores do programa de pós-graduação supracitado.

Faculdade de Tecnologia SENAI CIMATEC

Mestrado Acadêmico em Modelagem Computacional e Tecnologia Industrial

A Banca Examinadora, constituída pelos professores abaixo listados, aprova a Defesa de Mestrado, intitulada **"Visualização de Informação em Redes Sociais e Complexas Utilizando GuaráScript"**, apresentada no dia 07 de dezembro de 2016, como parte dos requisitos necessários para a obtenção do Título de Mestre em Modelagem Computacional e Tecnologia Industrial.

Orientador: Prof. Dr. Hernane Borges de Barros Pereira
SENAI CIMATEC

Coorientador: Prof. Dr. Roberto Luiz Souza Monteiro
SENAI CIMATEC

Membro Externo: Prof. Dr. Hugo Saba Pereira Cardoso
UNEB

Membro Externo: Prof. Dr. Eduardo Manuel de Freitas Jorge
UNEB

Dedico este trabalho a minha família que me apoiou nos momentos difíceis que passei durante a realização deste trabalho.

Agradecimentos

Sou extremamente grato a minha família e amigos que me deram suporte para que a realização desse sonho fosse possível. Gostaria de agradecer ao meu orientador Hernane Borges de Barros Pereira pelos ensinamentos, pela paciência e por acreditar que a realização deste trabalho fosse possível. Obrigado Mestre. Agradeço ao meu coorientador Roberto Monteiro pelos ensinamentos de GuaráScript, que foi a linguagem de programação escolhida para o desenvolvimento do software resultante dessa pesquisa.

A FAPESB, pelo auxílio financeiro que foi fundamental para conclusão deste trabalho.

Um agradecimento especial a Eduardo Jorge, Uedson Reis e Daniel Sales que foram os primeiros incentivadores, amigos de coração ao qual tenho enorme carinho e que me apoiaram de todas as formas possíveis. Agradeço muito a vocês. Não poderia deixar de agradecer com muito carinho ao meu grande amigo, parceiro e incentivador Thiago Cajahyba, uma pessoa que tenho um enorme carinho e que fez de tudo para que eu pudesse crescer profissionalmente, academicamente e como pessoa. Muito Obrigado.

Salvador, Brasil
dia de Dezembro de 2016

Rodrigo Paixão Vilas Bôas

Resumo

Recursos matemáticos e estatísticos têm sido a base para a identificação de informações relevantes da rede, e com isso obter informações úteis para compreensão de seu comportamento. No entanto, o advento da visualização de redes sociais e complexas apresenta uma nova perspectiva para análise das redes, auxiliando os pesquisadores a entender melhor seu comportamento. Apesar de ao longo das ultimas décadas o número de publicações sobre o tema de visualização de redes se manterem constantes, novos estudos são necessários para abordar novas formas de atuação sobre as redes. O propósito da pesquisa é a especificação de um modelo computacional e o desenvolvimento de um software Open Source que agregue algoritmos para visualização de redes sociais e complexas em ambiente bi-dimensional. Para isso, uma revisão sistemática foi realizada para levantamento dos algoritmos, e os mesmo foram implementados em GuaráScript integrando uma ferramenta de visualização de redes sociais e complexas.

Palavras-chave: Visualização de Redes, Algoritmos de Visualização, Visualização de Informação, Modelagem Computacional

Abstract

Mathematical and statistical resources have been the basis for identify relevant information from the network and through this gaing useful information for understanding their behavior. However, the advent of the visualization of social and complex networks show a new perspective for the analysis of networks that helping the researchers to better understand the their behavior. Although over the last decades the number of publications about the theme of network visualization has remained constant, new studies are needed to approach new ways of acting about the networks. The purpose of the research is the specification of a computational model and the development of an Open Source software that contains algorithms for visualizing social and complex networks in a bi-dimensional environment. A systematic review was performed to search the algorithms and they were implemented in GuaráScript integrating a visualization tool of social and complex networks.

Keywords: Network Visualization, Visualization Algorithms, Information Visualization, Computational Modeling

Sumário

1 Introdução **1**
- 1.1 Definição do problema . . . 2
- 1.2 Objetivo . . . 2
- 1.3 Importância da pesquisa . . . 3
- 1.4 Motivação . . . 3
- 1.5 Limites e limitações . . . 4
- 1.6 Aspectos Metodológicos . . . 4
  - 1.6.1 Conversão e integração dos algoritmos . . . 5
- 1.7 Organização da Dissertação de mestrado . . . 6

2 Revisão sobre visualização de Redes Sociais e Complexas **7**
- 2.1 Trabalhos Correlatos . . . 9
- 2.2 Método de Pesquisa . . . 10
  - 2.2.1 Questão da Pesquisa . . . 11
  - 2.2.2 Identificando e Selecionando Estudos . . . 11
  - 2.2.3 Critérios de Escolha e Procedimentos . . . 11
  - 2.2.4 Estratégia e Extração de Dados . . . 12
  - 2.2.5 Conduzindo a Revisão da Literatura . . . 12
- 2.3 Resultados e Análise Crítica . . . 13

3 Visualização de Informação **16**
- 3.1 Visão Geral . . . 16
- 3.2 Tipos de Visualização . . . 18
  - 3.2.1 Visualização Científica . . . 18
  - 3.2.2 Visualização de Redes . . . 19
    - 3.2.2.1 Algoritmos de Força . . . 21
    - 3.2.2.2 Estáticos e Circulares . . . 21
    - 3.2.2.3 K-Core . . . 22

4 Modelo Proposto **23**
- 4.1 Modelagem de Software . . . 23
- 4.2 GuaráScript . . . 25
- 4.3 OpenGL . . . 26
- 4.4 SCNTools . . . 27

5 Algoritmos de Visualização de Redes Sociais e Complexas **28**
- 5.1 Algoritmo Force-Direct Fruchterman Reingold . . . 28
- 5.2 Circular Layout . . . 33
- 5.3 K-core . . . 35
- 5.4 Random Layout . . . 37
- 5.5 Expansion e Retraction . . . 38

6 Resultado e Discussões **43**
6.1 Funcionamento do software Network Visualization 43
6.2 Algoritmo de Force Direct - Fruchterman Reingold 44
6.3 Algoritmo Circular 48
6.4 Algoritmo K-core 49

7 Considerações finais **51**
7.1 Contribuições 52
7.2 Atividades Futuras de Pesquisa 52

A Documentos **53**
A.1 Instruções de Instalação 53

B Algoritmo Circular **55**
B.1 Linha de Comando 56

C Algoritmo Expansion **57**
C.1 Linha de Comando 59

D Algoritmo Fruchterman **60**
D.1 Linha de Comando 64

E Algoritmo K-core **65**
E.1 Linha de Comando 68

F Algoritmo Random **69**
F.1 Linha de Comando 70

G Algoritmo Retraction **71**
G.1 Linha de Comando 73

Referências **74**

Lista de Tabelas

2.1 Publicações separadas por tema e construção 13
2.2 Periódico onde as publicações foram encontradas 14
2.3 Conferências onde as publicações foram encontradas 14

6.1 Algoritmos Disponíveis na Aplicação 44

A.1 Algoritmos implementados no DrawNet 53

Lista de Figuras

2.1 Resultado das pesquisas com descritores específicos 8
2.2 Variação da quantidade de publicações de 1998 a 2015 13

3.1 Imagem de Charles Joseph Minard representando a perda do exército de Napoleão Bonaparte no caminho até Moscou em 1812 17
3.2 Imagem representando as mortes de homens nos Estados Unidos entre os anos de 1970 a 1994 pelo cancer 18
3.3 Imagem de dois exemplos de visualização científica. Estrutura de DNA a esquerda e a da direito representa o buraco da camada de ozônio. 20

4.1 Macro arquitetura do modelo computacional 24
4.2 Representação dos vetores de posição do vértices do eixo X e Y 25
4.3 Comparativo entre matriz de adjacências e vetor de arestas 26

5.1 Exemplo do algortimo de Fruchterman 33
5.2 Exemplo do algoritmo de layout circular 35
5.3 Figura retirada do artigo K-core decomposition: a tool for the visualization of large scale networks 37
5.4 Representação da execução do algoritmo Random com rede de 100 vértices executado no Gephi 39
5.5 Representação do plano cartesiano com valores variando de -1 a 1 nos dois eixos 40
5.6 Representação do plano cartesiano com valores variando de 0 a 1 nos dois eixos 41
5.7 Figura representando a execução do algoritmo Expansion 41
5.8 Figura representando a execução do algoritmo Expansion 42

6.1 Excerto de um arquivo .net do Pajek 45
6.2 Figura comparando a imagem gerada pela aplicação DrawNet com a imagem gerada pelo Gephi. Rede com 100 vértices 46
6.3 Figura comparando a imagem gerada pela aplicação DrawNet com a imagem gerada pelo Pajek. Rede com 100 vértices 46
6.4 Figura comparando a imagem gerada pela aplicação DrawNet, Gephi, Pajek e imagem retirada do artigo *Graph Drawing by Force-directed Placement* 47
6.5 Figura comparando a imagem gerada pela aplicação DrawNet, Gephi, Pajek e imagem retirada do artigo *Graph Drawing by Force-directed Placement*. Grafos completos K5, K6 e K8 47
6.6 Figura comparando a imagem gerada pela aplicação DrawNet com a imagem gerada pelo Pajek. Rede com 100 vértices 48
6.7 Figura comparando a imagem gerada pela aplicação DrawNet com a imagem retirado do artigo *Circular Drawing Algorithms* 48
6.8 Figura comparando a imagem gerada pela aplicação DrawNet do algoritmo K-core com os círculos coloridos adicionado posteriormente para comparação a imagem retirada do artigo *K-core decomposition: a tool for the visualization of large scale networks*. Rede com 100 vértices 50

6.9 Figura comparando a imagem gerada pela aplicação DrawNet do algoritmo K-core com os círculos coloridos adicionado posteriormente para comparação a imagem retirada do artigo *K-core decomposition: a tool for the visualization of large scale networks*. Rede com 500 vértices 50

Lista de Siglas

API	Application Program Interface
APL	Arranjo Produtivo Local
MVC	Model, View & Controller
DNA	Deoxyribonucleic Acid

Capítulo Um

Introdução

Redes complexas são úteis para descrever uma ampla gama de sistemas na natureza e na sociedade (Albert; Barabási, 2002). Seu estudo iniciou-se com o desejo de compreender vários sistemas reais. Alguns exemplos comumente citados incluem redes de dispersão de doenças, a internet, redes de difusão de conhecimento e redes de ligação entre elementos químicos.

Nessa área, os trabalhos de Solomonoff e Rapoport (1951), Erdős e Rényi (1959) e Watts e Strogatz (1998) são os alicerces que contribuem para a identificação e categorização de redes complexas. Entretanto, os mesmos não se ocupam em definir modelos de visualização para as mesmas. Nesse contexto, diversos algoritmos de visualização de redes foram criados. Dentre eles destacam-se Jacomy et al. (2014), Nawaz e Jha (2007), Freeman (2004), Guo et al. (2015) e Hu (2011) onde cada um defende e cria sua própria forma de exibir informações de redes levando em consideração características inerentes da rede.

Segundo Chen (2010), a visualização de informação é um campo de pesquisa que busca aplicar a transformação gráfica de dados, em sua maioria abstratos e não espaciais, em uma produção gráfica que transmita com rigor a essência do dado bruto em uma interação intuitiva, precisa e fiel.

Através da pesquisa, foi possível observar que a área de visualização de informação, mas especificamente no campo que trata da visualização de redes, diversos pesquisadores vêm contribuindo com significativas ideias, propostas e modelos que reforçam a importância da visualização de redes quando se estuda seu comportamento. Dentre essas pesquisas podemos citar Giacomo et al. (2014) que apresenta uma proposta para minimizar o cruzamentos das arestas de uma rede propondo uma disposição em camadas sobrepostas, atribuindo peso variável as arestas para que as mesmas sejam dispostas contornando umas as outras. Outro estudo é o de Shelley e Gunes (2012) que propõe a plotagem da rede em uma espécie de globo, onde o observador tem a mesma perspectiva de visualização de um visitante a um planetário. A rede é disposta em forma de esfera, onde o observador permanece em uma posição fixa observando, enquanto a rede se movimenta ao seu redor. Além da forma com que as redes são desenhadas, outros estudos buscam melhorar a performance de algoritmos para exibição de grandes redes como o artigo de Didimo e Montecchiani (2012) que apresenta uma série de algoritmos otimizados como algoritmos de força, bem como modelagens que podem ser implementadas de forma paralela, ou seja, que apresentam melhor performance ao serem executados. Com isso uma visualização de rede bem montada, pode revelar comportamentos que até então seriam possíveis de

descobrir ou inferir apenas a partir de cálculos e associações às topologias da rede.

Este trabalho tem como tema principal a visualização de redes sociais e complexas como ferramenta para extração de informação a partir de desenhos gráficos gerados por meios de algoritmos. Entende-se que apesar dos largos recursos matemáticos disponíveis para os estudos das redes, a inspeção visual pode contribuir com a investigação de redes, dando suporte à problemas de pesquisa encontrados durante investigação.

1.1 Definição do problema

O estudo de redes sociais e complexas tem crescido ao longo dos anos, tanto pelo seu carácter interdisciplinar quanto pelo sólido fundamento matemático o que possibilita a correlação de estudos com diversas áreas do conhecimento. Esses estudos são pautados, em sua maioria, pelos cálculos das propriedades de redes e a sua caracterização topológica (e.g. Redes *Small World*, Livres de Escala e Aleatórias). Essas redes possuem comportamentos próprios segundo sua topologia. Os cálculos de propriedades são feitos através de softwares que geram resultados através de console ou ferramenta de relatório em formato de texto, onde alguns dispões de ferramentas gráficas para dar suporte ao estudo da rede provendo uma outra perspectiva de análise. Outra questão é que alguns softwares utilizadas para o estudo de rede, o Pajek por exemplo, não tem seu código fonte aberto, o que impossibilita que outros pesquisadores possam contribuir para evolução do software.

1.2 Objetivo

O objetivo principal do trabalho é o desenvolvimento de um modelo computacional capaz de plotar redes sociais e complexas em ambientes bi-dimensionais. A partir do desenvolvimento do modelo computacional outro objetivo é a criação de um software Open Source para visualização de redes sociais e complexas em ambiente bi-dimensional em GuaráScript de nome DrawNet capaz de prover recursos de exploração visual das redes geradas no software e irá se integrar a ferramenta de estudo de redes SCNTools cujas funcionalidades até o momento dão conta de cálculos de propriedades das redes, criando assim, uma plataforma integrada de manipulação de redes sociais e complexas e a implementação de algoritmos para geração gráfica de redes sociais e complexas.

1.3 Importância da pesquisa

O grande número de publicações sobre o tema de visualização de informação não é impeditivo para que novos estudos ou modelos computacionais sejam propostos e esta pesquisa apresenta mais um modelo computacional buscando contribuir no desenvolvimento de novas pesquisas e de novos softwares. Esse trabalho busca especificar um modelo computacional desde sua macro-arquitetura a pseudo-códigos que auxilie no desenvolvimento de softwares de visualização de redes 4. O intuito é atrair ainda mais pesquisadores na evolução do modelo proposto pela pesquisa.

Após a especificação do modelo, esta pesquisa se propõe a desenvolver um software em linguagem GuaráScript aplicando técnicas de desenvolvimento de software fazendo com que a aplicação pudesse ser evoluída com a adicão de recursos computacionais e a possibilidade de usar a aplicação como uma componente que poderá ser acoplado a outra solução pré-existente no intuito de fornecer os recursos visuais implementados.

1.4 Motivação

Após a realização da revisão sistemática apresentada no capítulo 2, a pesquisa na área de visualização de redes vem sendo mantida em atividade ao longo das duas últimas décadas, mas foi possível perceber que existem lacunas que necessitam de contribuições para fortalecer o campo de pesquisa, algumas delas são:

- Baixo número de publicações sobre o tema;
- Escassez de softwares *open source* para visualização de redes.

Esses itens demonstram a necessidade de novas contribuições na área de pesquisa com intuito de fomentar o estudo de visualização de redes sociais e complexas como auxiliador na obtenção de informações comportamentais da rede estudada.

Aplicações *Open Source* têm como grande objetivo atrair novos pesquisadores a trabalharem em conjunto para evolução da ferramenta, o que dificilmente acontece com software proprietários, e a ideia de desenvolver um software que pudesse agregar recursos úteis para a visualização de redes sociais e complexas surgiu perante as necessidades anteriormente citadas.

1.5 Limites e limitações

O estudo apresentado possui como princípio seu carácter multidisciplinar pois perpassa o estudo de Redes Sociais e Complexas fazendo-se valer de técnicas discutidas no campo de pesquisa de visualização de informação, sistemas de informação e a matemática. Os limites da pesquisa se caracterizam em: (1) busca e coleta de algoritmos de visualização de redes; (2) especificação do modelo computacional; (3) implementação de algoritmos capazes de distribuir espacialmente os vértices e arestas da rede usando a linguagem GuaráScript.

No decorrer do desenvolvimento e especificação do modelo computacional, percebeu-se algumas limitações impostas pelas linguagem de programação utilizada e tecnologias escolhidas. Com relação à linguagem de programação, a limitação se dá pelo fato da mesma não dispor, até o presente momento, do paradigma de orientação a objetos nem de recursos de paralelização. Ela é uma linguagem com característica estruturada e dispõe de um interpretador, o que faz com que o código não precise ser compilado previamente.

Outra limitação que já era esperada, é no que diz respeito a utilização do OpenGL em sua forma mais primitiva, sem a utilização de *Engines*(ferramentas de software que dispões de recursos integrados para o desenvolvimento de aplicações gráficas como jogos, realidade virtual, entre outros) para a geração gráfica. A adoção de *Engines* como plataforma para desenvolvimento gráfico traz consigo diversas bibliotecas gráficas acopladas que nem sempre serão utilizadas, isso faz com que o uso de recurso computacional seja desperdiçado. Com isso fizemos a opção por não utilizar nenhum componente adicional para geração gráfica motivado pela tentativa de economizar recursos computacionais e assim obter uma melhor performance na execução do software.

1.6 Aspectos Metodológicos

Para o desenvolvimento desta pesquisa uma revisão sistemática foi realizada com intuito de gerar um compilado de publicações utilizado como suporte teórico, metodológico e técnico para o desenvolvimento da pesquisa. Um modelo computacional foi proposto com intuito de especificar uma macro arquitetura e algoritmos em pseudo código que serviu de base para o desenvolvimento de um software para visualização de redes em ambiente bi-dimensional. A partir da separação e escolha das publicações que iriam fazer parte da pesquisa, foi definido quais algoritmos seriam utilizados para serem implementados. A escolha dos algoritmos levou em consideração sua aplicabilidade, resultado final da visualização, velocidade de execução e que a sua implementação pudesse ser comparada com outras implementações em softwares já conhecidos como *Pajek e Gephi*. Os algoritmos

foram catalogados em três categorias: Algoritmos de Força, circulares e K-core. O conceito dos Algoritmos de Força é a utilização das forças de repulsão e atração para dispor os vértices e arestas no espaço dinamicamente, ou seja, os cálculos de posicionamento são realizado a medida que a execução do algoritmo é realizada. Os algoritmos circulares possui característica estática, ou seja, o posicionamento de todos os vértices são calculados antes da exibição da disposição final. Os algoritmos de K-core tem comportamento e disposição muito parecida com os algoritmos circulares mas com uma particularidade de apresentar no centro do plano cartesiano os vértices que possuem a maior quantidade de ligações com outros vértice afim de facilitar a identificação dos vértices que possuem maior "importância"da rede.

Após a categorização dos algoritmos, os mesmos foram implementados na linguagem GuaráScript para fazer parte da biblioteca de algoritmos disponível no software de visualização de redes. Os algoritmos foram implementados de forma independente evitando o acoplamento dos códigos o que facilita o reaproveitamento dos algoritmos em outras aplicações ou evolução individual do algoritmo.

O software de visualização de redes foi desenvolvido seguindo a base do padrão de projeto MVC(*Model View Controller*), com adaptações onde a biblioteca de algoritmos foi atribuída a camada que seria originalmente para os modelos de dados (*Model*) e as características das outras camadas foram mantidas. Na camada (*Controller*) foi desenvolvido um módulo de serviço responsável pela operação e comunicação entre a camada visual (*View*) e a camada da biblioteca de algoritmos (*Model*) fornecendo uma gama de recursos comuns. O modelo MVC, conhecido como modelo 3 camadas é muito utilizada para que cada módulo possa ser executando de forma independente facilitando a reutilização dos módulos para outras aplicações. Já a camada visual (*View*) foi implementado seguindo os princípios definidos na API do OpenGL para exibir os vértices e arestas.

Após o desenvolvimento da aplicação, foram realizados testes com algumas redes para comparação do resultado obtido com outros software já conhecidos e também com o que é encontrado na literatura para validação se a implementação dos algoritmos, atendem aos requisitos exigidos pelos seus autores.

1.6.1 Conversão e integração dos algoritmos

Após a coleta e separação dos algoritmos obtidos através da revisão sistemática, alguns procedimentos precisaram ser realizados antes de portar o algoritmo para a linguagem GuaráScript. Primeiro foi verificado se a publicação apresentava um algoritmo em pseudo-código, caso positivo, o pseudo-código foi transcrito para uma linguagem estruturada de alto nível como Portugol que é mais compreensível para os humanos do que para as

máquinas para facilitar o entendimento do funcionamento do algoritmo e com isso se definir a estratégia de implementação na linguagem GuaráScript. Caso a publicação não apresente pseudo-código, as funções do algoritmo são separadas para realização de transcrição em Portugol e posteriormente implementadas em GuaráScript.

Com o algoritmo já escrito e refatorado na linguagem GuaráScript, o mesmo precisaria ser integrado ao módulo de serviços que dispões de funcionalidades já implementadas e de uso comum a todos algoritmos para que o mesmo seja utilizado pela aplicação.

1.7 Organização da Dissertação de mestrado

A presente pesquisa está organizada da seguinte forma:

- **Capítulo 2 - Revisão Sistemática**: Capítulo com dados da revisão sistemática realizada para identificação de publicações na área de pesquisa;
- **Capítulo 3 - Visualização de Informação**: Capítulo responsável por apresentar os conceitos sobre Visualização de informação e suas características multidisciplinares;
- **Capítulo 4 - Modelo Proposto**: Capítulo responsável por apresentar a proposta do modelo computacional, arquitetura e estrutura de dados;
- **Capítulo 5 - Algoritmos de Visualização de Redes Sociais e Complexas**: Capítulo Responsável por apresentar os algoritmos implementados em GuaráScript com pseudo-código baseado no paradigma estruturado;
- **Capítulo 6 - Resultados e Discussões**: Capítulo responsável por apresentar os resultados obtidos com a geração de visualizações com outros softwares bem como um comparativo com a literatura para validação dos dados.
- **Capítulo 7 - Considerações Finais**: Capítulo responsável por apresentas as considerações finais bem como as contribuições da pesquisa e atividades futuras a serem desenvolvidas a partir dos resultados obtidos pela pesquisa.

Capítulo Dois

Revisão sobre visualização de Redes Sociais e Complexas

Em seus livros, Mazza (2009) e Chen (2006) discorrem sobre a quantidade de dados que nos deparamos diariamente em nossas atividades e as diversas maneiras que encontramos para representá-los. Essas formas vão de simples desenhos em folhas de papel a gráficos elaborados em programas profissionais. Tudo dependerá de como se deseja comunicar a ideia. Todavia, ambos autores demonstram que a forma como os dados são exibidos influencia na percepção dos mesmos, ou seja, influencia em como obtemos informação e geramos conhecimento a partir da mesma.

Sendo o estudo de Redes Sociais e Complexas uma área multidisciplinar, a utilização de técnicas de visualização de informação contribui de forma significativa para a compreensão e aprendizagem dos tipos de redes(e.g.Redes Biológicas e Redes Sociais) (Newman, 2003). Ao fazer uso das técnicas de visualização de informação propriedades e/ou características pertinentes ao comportamento da rede analisada podem emergir, e essas técnicas propiciam uma nova ótica de análise sendo um fator motivador para o estudo correlato das áreas de Visualização de Informação e Redes Sociais e Complexas.

A partir da abordagem interdisciplinar que as áreas de pesquisa de Visualização de Informação e Redes Sociais e complexas possuem, este capítulo apresenta uma revisão das metodologias utilizadas por pesquisadores, com intuito de obter assim um apanhado de artigos, teses e livros que irão orientar futuras pesquisas. A revisão teórica teve como plataforma de pesquisa o Portal Periódicos Capes/MEC, Web of Science, Science Direct e IEEE, e tem como objetivo levantar informações úteis para o tema de interesse prezando a imparcialidade.

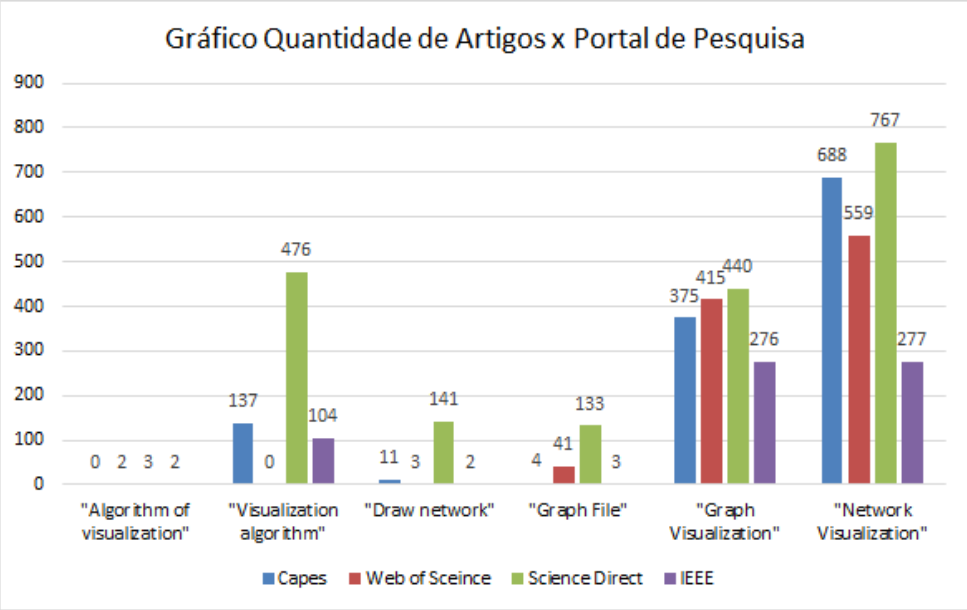

Figura 2.1: Resultado das pesquisas com descritores específicos

Os descritores escolhidos para realizar a pesquisa, tiveram o intuito de reunir publicações que auxiliasse no desenvolvimento da dissertação e servissem de suporte teórico e prático das propostas desenvolvidas. Os descritores "Algorithm of visualization"e "Algorithm visualization"tiveram como intuito principal encontrar publicações na área de Visualização de Informação que pudessem apresentar técnicas úteis para utilização nos algoritmos de visualização de redes. Os descritores "Draw network", "Graph visualization"e "Network Visualization"tiveram como intuito encontrar publicações mais específicas na área de visualização de redes, que pudessem apresentar técnicas ou algoritmos para plotagem das redes. O descritor "Graph file"foi escolhido com intuito de obter publicações que pudesse contribuir com o tratamento e organização dos atributos (vértices e arestas) das redes em formato de arquivo para facilitar o armazenamento e recuperação dos dados da redes através de softwares e podem ser conferidos na Figura 2.1.

Outros descritores foram utilizados, mas não apresentaram resultado em todos os portais pesquisados que foram: Algorithm of visualization of complex networks; Algorithm of visualization of social networks; Algorithm of visualization of graphs; Visualization algorithm of complex networks; Visualization algorithm of social networks; Visualization algorithm of graph; Algorithm of 2D Visualization; Algorithm of 3D Visualization.

O estudo buscou analisar as pesquisas feitas na área de Visualização de informação aplicada na área de conhecimento de Redes Sociais e Complexas que pudessem responder a seguinte questão: O que tem sido empregado no campo de visualização de Redes Sociais e Complexas por pesquisadores que pode ajudar na obtenção de informação?

Na seção 2.1, apresentamos um apanhado de trabalhos relacionados ao tema de visualização de informação e visualização de redes sociais e complexas. Na seção 2.2, apresentamos um método de pesquisa e classificação das publicações encontradas. Na seção 2.3, analisamos os resultados obtidos após a seleção das publicações.

2.1 Trabalhos Correlatos

Foram encontrados na literatura diversos trabalhos que abordam visualização de informação e Redes Sociais e/ou Complexas. Podemos citar o livro *Information Visualization – Beyond the Horizon* de Chen (2006) que discorre sobre diversas perspectivas e métodos de exibição de informações com intuito de manter uma imagem fiel dos dados.

Outro estudo encontrado que propõe um algoritmo de visualização de redes é o *ForceAtlas2, A Graph Layout Algorithm for Handy Network Visualization* de Jacomy et al. (2014). Os autores propõem um modelo de algoritmo de vetor de força que segue o princípio da força atrativa e de repulsiva para determinar o posicionamento dos vértices no espaço. Os vértices não possuem locais fixos, eles são posicionados relativamente aos outros vértices atendendo a fórmula do algoritmo de atração e repulsão. Os vértices possuem um coeficiente de repulsão entre si seguindo o princípio do modelo de partículas elétricas carregadas ($Fr = k/d^2$) onde a força de repulsão Fr é determinada pelo grau do vértice dividido pela distância entre eles ao quadrado. Já as arestas possuem um coeficiente de atração entre os vértices aos quais estão conectadas seguindo o modelo de mola ($Fa = -k*d$), onde a força de atração Fa é determinada pelo grau do vértice com sinal negativo multiplicado pela distância entre os dois nós. Essa técnica segue o princípio da velocidade versus precisão no intuito de encontrar o equilíbrio da rede. Quanto mais rápido se quiser encontrar o equilíbrio da rede menor será a precisão visual da mesma e vice-versa. Esse algoritmo é muito utilizado no estudo das redes sociais, pois a atração entre dos vértices geram aglomerados que fazem analogia ao sentido de comunidade de pessoas.

Outro importante trabalho foi o de Freeman (2004) intitulado de *Graphical Techniques for Exploring Social Networks Data*, onde é feito uma análise da evolução das imagens visuais (como o autor se refere às formas de exibição) de redes desde Moreno, Whitin e Jennings (1932). Em seu trabalho, Moreno, Jennings et al. (1934) analisou estudantes da quarta série do ensino fundamental, onde alterou o formato dos vértices para diferenciar meninos e meninas. Ele atribuiu direção nas arestas com intuito de demonstrar o vértice que buscou iniciar uma amizade e descobriu que isso contribuía para uma melhor representação do que de fato ocorreu. Todavia, Freeman (2004) percebeu que Moreno, Jennings et al. (1934) não possuía uma preocupação sistemática com o posicionamento dos vértices no espaço, mas sim apenas se preocupou exibir as relações entre os vértices.

Freeman (2004) evoluiu o modelo proposto por Moreno, Jennings et al. (1934) ao constatar a importância de ter um método sistemático de posicionamento dos vértices e o quanto isso melhorava a percepção das informações. A partir disso Freeman (2004) propõe o desenvolvimento de procedimentos que aplicados sobre a mesma rede diversas vezes, gerem os mesmos resultados. Ele defende que ao serem alocados os vértices no espaço destinado, as relações mais fortes entre os atores sejam preservadas, ou seja, se houver 1 par de vértices ligados diretamente no conjunto de dados, isso deve ser contemplado visualmente.

Seguindo a linha de visualização de redes, encontramos a pesquisa de Alvarez-Hamelin et al. (2005) intitulado: *K-core decomposition: a tool for visualization of large scale networks* que tem o intuito de fornecer uma ferramenta para análise de redes de grande escala. A ferramenta propõe a retirada recursiva dos vértice de menor grau, ou seja, os vértices com menores ligações seriam retirados do núcleo da rede e deslocado para as margens da visualização, até obter um núcleo (core) da rede com vértices mais conectados. Seguindo a lógica de reposicionamento dos vértices onde os vértices com maior grau estão mais ao centro da visualização e os vértices com menor grau as margens, pretende-se com isso facilitar a análise da rede com intuito de obter assim a impressão digital da rede.

Segundo Cheng et al. (2011) os técnicas para encontrar o core da rede não seguem um padrão de remoção recursiva dos vértices, e com isso há um desperdício de recurso computacional e acaba por consequência limitando o tamanho da rede para ter uma eficiência satisfatório em encontrar seu core. Com isso Cheng et al. (2011) em seu trabalho intitulado *Efficient Core Decomposition in Massive Networks* apresenta um algoritmo denominado de "Algoritmo de Memória Externa"(Tradução livre) que tem complexidade de $\theta(Kmax)$ onde $Kmax$ é o grau máximo da rede e se mostrou eficiente em uma rede de mais de 52.9 milhões de vértices e 1.65 bilhões de arestas. Em síntese o algoritmo divide a rede em blocos no tamanho de memória pré-determinado, efetua o calculo do grau máximo da rede e depois realiza a separação dos vértices com maior grau da rede e os exibe.

2.2 Método de Pesquisa

Revisão sistemática é um modo de pesquisa que utiliza uma fonte de dados sobre determinado assunto com intuito de identificar, avaliar e interpretar todas as relevantes pesquisas sobre o tema em questão (Kitchenham, 2004). É possível identificar a importância da revisão sistemática quando o objetivo é identificar como estão as pesquisas sobre determinado tema até o presente momento, seguindo uma estratégia rigorosa.De acordo com Kitchenham (2004) uma revisão sistemática possui três macro-etapas a serem seguidas, são elas: Planejamento, Condução da Revisão, Relatório da Revisão.

Na etapa do planejamento, os estágios associados são: a identificação da necessidade de

uma revisão e o desenvolvimento do protocolo dela, na etapa de condução da revisão, os estágios associados são: a identificação da pesquisa, seleção dos estudos primários, avaliação da qualidade do estudo, extração e monitoramento de dados e a síntese dos dados já na etapa de relatório da revisão é uma fase única que consiste em apresentar os resultados a uma revista, congresso e/ou a uma seção na dissertação de mestrado ou tese de doutorado.

2.2.1 Questão da Pesquisa

O objetivo da revisão sistemática foi pesquisar e trazer uma análise atual sobre o tema de Visualização de Redes Sociais e Complexas em ambientes 2D e 3D com intuito de verificar o estágio dos estudos sobre o tema. O que irá orientar a pesquisa é a tentativa de identificar quais algoritmos de visualização de redes sociais e complexas estão sendo usados para melhor extrair informações e/ou comportamento das redes.

2.2.2 Identificando e Selecionando Estudos

As bases de artigos utilizadas para o desenvolvimento dessa revisão foram: Portal Periódicos da Capes/MEC, Science Direct, Web of Science, IEEE. Os Livros: *Information Visualization – Beyond the Horizon de Chaomei Chen 2006*, Livro: *Introduction to Information Visualization de Riccardo Mazza 2009* foram inseridos propositadamente por serem arcabouços teóricos de extrema importância na área de visualização de informação.

Os estudos levantados através dos portais de pesquisa, listados acima, foram obtidos com base nos descritores no intuito de reunir a maior quantidade de pesquisas relevantes ao tema pesquisado. Os descritores utilizados foram: Algorithm of visualization, Visualization algorithm, Draw network, Graph File, Graph Visualization, Network Visualization.

Não foram utilizados filtros de pesquisa de tempo, país ou qualquer outro. A seleção dos trabalhos foram feitos pelo critérios definidos na seção 2.2.3.

2.2.3 Critérios de Escolha e Procedimentos

Para inclusão ou exclusão das produções bibliográficas, foram definidos os seguintes critérios:

Critérios de Inclusão:

1. Estudos sobre algoritmos de visualização 2D e 3D de redes sociais e complexas que apresentaram metodologia de implementação;
2. Estudos sobre a teoria dos grafos, redes sociais e complexas que foram utilizados como estado da arte;
3. Estudos sobre visualização de informação que continham propriedades pertinentes as utilizadas para visualizar redes sociais e complexas.

Critérios de Exclusão:

1. Estudos que não tratavam sobre a teoria, aplicação e/ou proposição de algoritimos de visualização de redes sociais e complexas;
2. Estudos que não tratavam sobre a teoria dos grafos, redes sociais e complexas que são utilizados como estado da arte;
3. Estudos que não tratavam o tema de visualização de informação com propriedades pertinentes a visualização de redes sociais e complexas.

2.2.4 Estratégia e Extração de Dados

O tema de visualização de Redes Sociais e Complexas é a questão central da pesquisa. Todavia, outros critérios ou questões secundárias se fazem presentes no intuito de melhor selecionar os estudos de modo a inclui-los nesta revisão sistemática.

As questões secundárias são: é um método/algoritmo para visualização de redes sociais e complexas? O foco do artigo é uma abordagem teórica, metodológica ou uma modelagem computacional?

2.2.5 Conduzindo a Revisão da Literatura

O período de pesquisa nos portais sobre o tema de visualização de redes sociais e complexas, foi de 01 de abril a 31 de julho de 2015. As pesquisas foram realizadas nos portais de pesquisa da CAPES, IEEE, Web of Science e Science Direct gerando um total de 4859 potenciais publicações sobre o tema pesquisado sendo: 1215 do Periódicos Capes, 1020 da Web of Science, 1960 da Science Direct, 664 da IEEE, 3 publicações não indexadas.

Após o levantamento das publicações em potencial, realizamos a seleção das publicações respeitando os critérios de inclusão e exclusão estabelecidos na seção 2.2.3 com a qual

obtivemos 45 publicações entre os anos de 1998 e 2015. Na Figura 2.2, verificamos a quantidade de publicações selecionadas ao longo do período supracitado onde se observa que os anos de 2009 e 2012 foram os que mais tiveram publicações selecionadas.

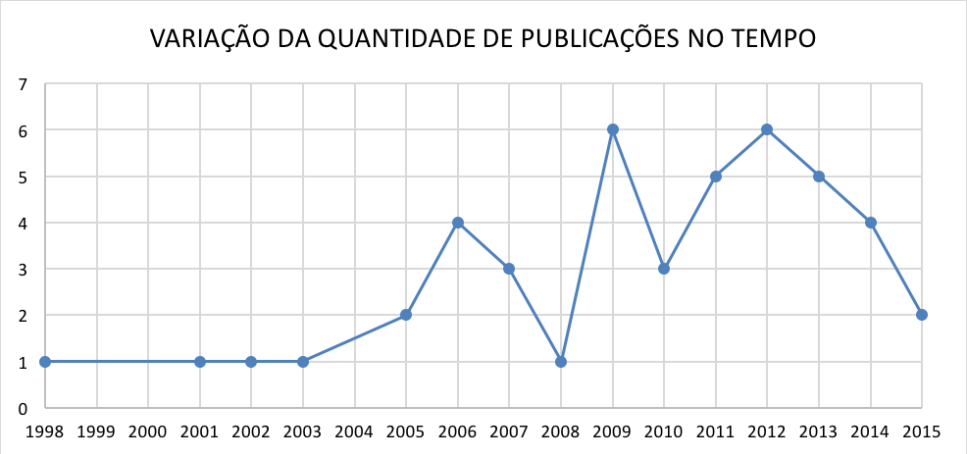

Figura 2.2: Variação da quantidade de publicações de 1998 a 2015

2.3 *Resultados e Análise Crítica*

Após a seleção das 45 publicações, as classificamos em diferentes categorias para melhor compreensão dos resultados obtidos. As classificações das publicações foram definidas para atender as expectativas da revisão de encontrar arcabouços teóricos e aplicações práticas de algoritmos de visualização de redes. Essas classificações seguiram os critérios da seção 2.2.4.

Tabela 2.1: Publicações separadas por tema e construção

TEMA ABORDADO	CAPES/MEC	SCIENCE DIRECT	WEB OF SCIENCE	IEEE	NÃO INDEXADO	TOTAL
VISUALIZAÇÃO DE INFORMAÇÃO	2	2	1	0	0	5
ALGORITMOS DE VISUALIZAÇÃO (REDES)	10	5	7	15	3	40
CONSTRUÇÃO						
ABORDAGEM TEÓRICA	2	1	1	0	1	5
MODELAGEM	10	6	7	15	2	40

Analisando a Tabela 2.1, podemos inferir características importantes das publicações selecionadas. É possível observarmos que a maioria das publicações selecionadas, em torno de 89%, possui como tema Algoritmos de Visualização e 11% referente ao tema de Visualização de Informação. Observamos também, que essa mesma proporção é observada quando as publicações são classificadas pelo método de construção. A modelagem foi o método de construção mais utilizadas pelos autores.

As Tabelas 2.2 e 2.3 mostram respectivamente os periódicos e as conferências ao qual foram realizadas as publicações. Na Tabela 2.2 é apresentado o quantitativo de publicações por periódicos, o Qualis Interdisciplinar e o JCR que no caso é o fator de impacto para o ano de 2014.

Tabela 2.2: Periódico onde as publicações foram encontradas

Periódicos	**Quantidade**	**Qualis Interdisciplinar**	**JCR**
Journal of Visual Languages and Computing	4	Não Classificada	0.893
Plos One	2	A1	3.534
Tsinghua Science & Technology	1	Não Classificada	0.000
American Journal of Sociology	1	Não Classificada	4.045
El Profesional de la Información	1	B1	0.330
Wiley Interdisciplinary Reviews: Computational Statistics	1	Não Classificada	0.000
Information Processing & Management	1	Não Classificada	1.069
Information Sciences	3	A1	3.893
Decision Support Systems	1	B1	2.036
Cell	1	A1	33.116
Computational Geometry	1	Não Classificada	0.082
Computer Networks	1	A2	1.282
Computers & Graphics	2	B1	1.029
IEEE Transactions on Visualization and Computer Graphics	6	Não Classificada	1.919
Physica A	1	A2	1.722
Procedia Computer Science	1	C	0.000
Statistica Sinica	1	Não Classificada	1.226

Tabela 2.3: Conferências onde as publicações foram encontradas

Conferências	**Quantidade**
Combinatorial Scientific Computing	1
10th Web Information System and Application Conference	1
2007 IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems	1
2nd International Multi-Symposiums on Computer and Computational Sciences	1
3rd International Conference on Pervasive Computing and Applications	1
5th International Conference on Intelligent Networking and Collaborative Systems	1
Advances in Neural Information Processing Systems 18	1
IEEE Pacific Visualization Symposium, PacificVis 2009	1
Proceedings - Computer Graphics, Imaging and Visualisation: Techniques and Applications	1
Proceedings - International Conference on Data Engineering	2
Proceedings of the 2011 IEEE 1st International Network Science Workshop	1
Proceedings of the International Conference on Information Visualisation	1
Proceedings on Seventh International Conference on Information Visualization	1
System(Tese)	1
The 10th International Conference on Digital Technologies	1

A partir do total de 45 publicações selecionadas, foi possível observar que o periódico que mais teve publicações foi o IEEE Transactions on Visualization and Computer Graphics com 6 publicações seguido do Journal of Visual Languages & Computing com 4 publicações. Outro periódico que destaca-se dos demais é o Information Sciences com 3 publicações, já os demais possuem 1 ou 2 publicações.

Foi possível observar que ao longo dos anos, as publicações de visualização de redes sociais e complexas suportada pela área do conhecimento de visualização de informação, apesar de variar a quantidade de publicações observa-se que esse tema é bastante estudado e tem como tendência se manter ativa ao longo dos anos, levando-se em consideração que o tema é bastante amplo e com isso esses estudos podem-se relacionar com diversas áreas e evoluir a forma com que os dados podem ser visualizados e/ou manipulados.

O tema visualização de redes sociais e complexas apesar de já ter sido bastante estudado, permite o surgimento de novas propostas partindo do pressuposto de que tudo depende do que se pretende obter com a visualização. Se o estudo foca as redes sociais, por exemplo, a visualização de conglomerados pode ser importante para identificação de comunidades, e com isso o algoritmo precisa contemplar essa possibilidade. No próximo capítulo serão abordados os tipos de visualização de informação incluindo técnicas que auxiliam na extração de informações.

Capítulo Três

Visualização de Informação

Nos dias atuais, nos deparamos com uma grande quantidade de informação que precisa de nossa atenção, como e-mails, mensagens no smartphone, fatura de cartão de crédito, notícias, etc (Zhu; Watts; Chen, 2010). Esse conjunto de informações, nos faz criar mecanismo ou estruturas como sistemas de gestão, geração gráfica de dados para que decisões sejam tomadas e surgem questões como: Minhas políticas de spam estão adequadas ou precisam ser revista? Será que esse e-mail é importante para que eu guarde por mais tempo?

Quando as informações que nos são providas não possuem uma determinada organização ou estruturação visual, a tendencia é que dados que poderiam ser importante para tomada de decisões, sejam ignorados ou simplesmente não sejam facilmente encontrados. São nesses pontos que os estudos no campo de visualização de informação podem ajudar a melhorar essas situações (Chen, 2002).

De acordo com Mazza (2009) as informações que são muito valiosas e importantes para nossas vidas, são construídas a partir de um fluxo contínuo e constante de dados ao qual somos passivamente ou ativamente submetidos. Para que esses dados gerem insumos para tomada de decisão, se faz necessário sua estruturação que possibilite transformar a massa de dados em informações úteis para a tomada de decisão.

3.1 Visão Geral

O termo Visualização de Informação refere-se a gráficos interativos gerados computacionalmente que procuram representar fielmente a informação, atraindo um leque diversificado de pesquisadores a atuar no desenvolvimento de novos estudos. Conceitualmente a visualização de informação se preocupa com a metodologia adotada desde a concepção do modelo computacional até a aplicação final no intuito de garantir que o processo criativo não gere representações destoantes dos dados utilizados (Olmeda-gómez, 2014).

As representações gráficas são utilizadas há muitos séculos atrás e um exemplo clássico do uso dessas representações foi na época de napoleão quando o Charles Joseph Minard representou graficamente as perdas do exército de Napoleão Bonaparte no caminho que o mesmo fez até Moscou em 1812, demonstrando como dados abstratos podem ser representados graficamente com intuito de facilitar o entendimento e apoiar a tomada de

decisão para quem os analisa. Na Figura 3.1 é possível identificar (fazendo a leitura da esquerda para direita) que, com o passar do tempo, o exército de Napoleão, que no início do percurso era bastante volumoso, vai perdendo seu quantitativo durante o percurso (Chen, 2010).

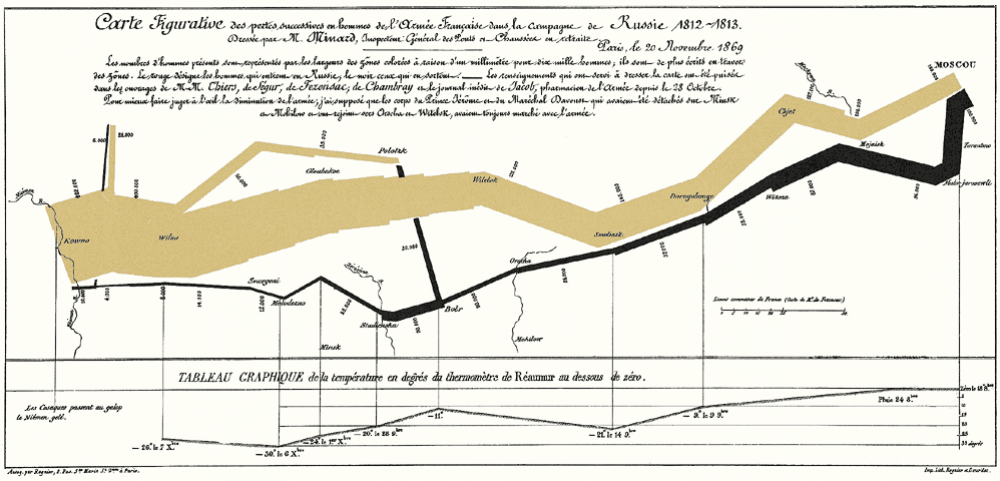

Figura 3.1: Imagem de Charles Joseph Minard representando a perda do exército de Napoleão Bonaparte no caminho até Moscou em 1812. Chen (2006)

Além da visualização de informação ser estudada e utilizada a bastante tempo, existem aplicações das técnicas de visualização em diversas áreas do conhecimento como biológica (mapeamento de epidemias), posicionamento geográfico, análises quantitativas para confirmação de alguma suspeita, entre outros por se tratar de uma área interdisciplinar. De acordo com Mazza (2009) a análise exploratória de dados é uma das aplicações que mais se beneficiam da capacidade do sistema cognitivo humano em analisar as visualizações geradas, e isso vem sendo usado a muitos anos para identificar propriedades, relacionamentos e padrões.

Jacques Bertin, um cartógrafo francês, escreveu uma obra em 1967 que definia os conceitos básicos para ter uma representação gráfica que mostrasse com fidelidade e de forma intuitiva, os dados a serem analisados. Com base nessa obra, Mazza (2009) ilustra o conceito definido por Jacques Bertin com a Figura 3.2 que representa estatisticamente dados da mortalidade de homens nos Estados unidos relacionados ao câncer entre os anos de 1970 a 1974, onde o mapa do país foi usado como referência geográfica e variações de cores representavam maiores e menores taxas de mortalidade. As cores variavam do azul ao vermelho, sendo o azul as menores taxas, passando pelo branco com taxa intermediárias e o vermelho com as maiores taxas de mortalidade. O intuito da ilustração, é demonstrar com facilidade onde estão concentrados os maiores e menores índices de mortalidade, para que políticas de controle da doença fossem adotadas, ou que se pudesse levantar conclusões ou hipóteses sobre os fatos apresentados. Com base na Figura 3.2, é possível notar que a maior concentração de mortalidade se encontra na costa leste e sudeste dos Estados

Unidos, e isso pode ser um indicativo de que o estilo de vida, alimentação ou fatores ambientais provenientes daquela região, pudessem influenciar os índices de mortalidades encontrados.

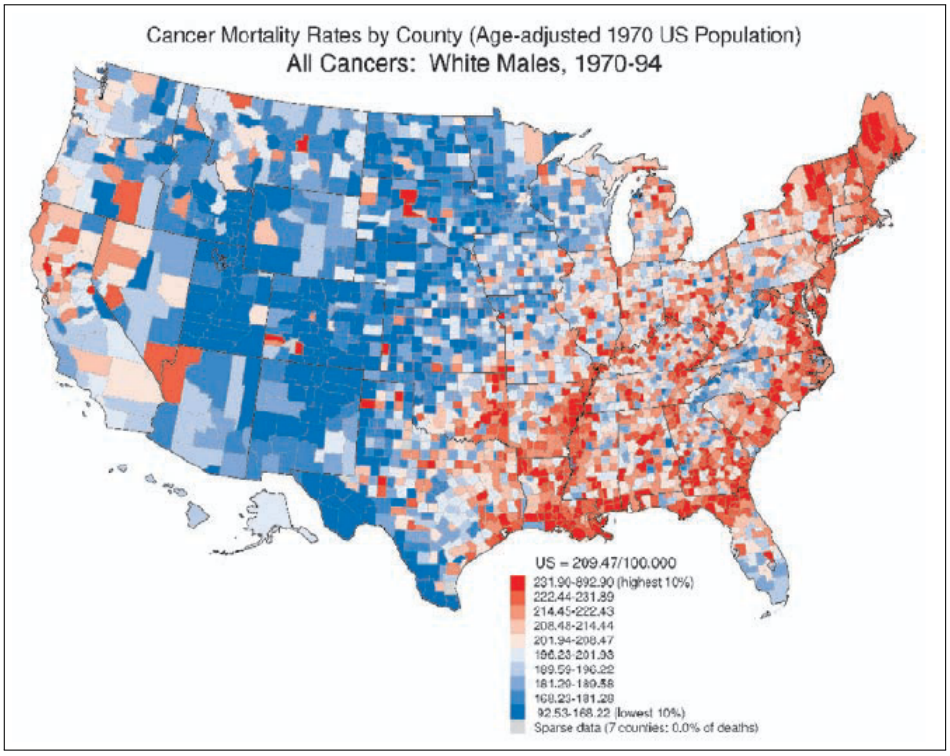

Figura 3.2: Imagem representando as mortes de homens nos Estados Unidos entre os anos de 1970 a 1994 pelo câncer. Fonte http://www3.cancer.gov/atlas/ acesso em 23/03/2016 as 20:30 hrs

3.2 Tipos de Visualização

A visualização de informação pode ser divida ou classificada em tipos de visualização como a visualização científica, visualização de redes e visualização computacional. De acordo com Zhu e Chen (2005) a visualização de informação pode ser classificada por uma ou mais classificações por não possuir fronteiras linearmente definidas entre elas. A visualização científica é um caso onde a visualização de elementos físicos como uma molécula ou um DNA, faz uso de recursos da visualização de informação que possui padrões embutidos para tratamento de uma grande escala de dados.

3.2.1 Visualização Científica

A visualização científica ajuda cientistas e engenheiros a conseguir o entendimento de fenômenos físicos em grandes volumes de dados. A principal diferença entre a visua-

lização de informação e a visualização científica é que enquanto a visualização de informação trata os dados de forma genérica, ou seja, possui métodos generalistas para tratamentos da massa de dados, uma vez que esses dados podem representar qualquer tipo de informação. Como visto na Figura 3.1, as técnicas utilizadas para demonstrar as perdas do exército napoleônico foram métodos generalistas como a relação de quantidade X tempo que poderia ser aplicados a outros tipos de visualização. Já quando se trata de dados que possuem correspondência física ou estão de certa foma ligados com modelos matemáticos, estrutura e modelos, como por exemplo o fluxo de ar em torno de um avião ou uma estrutura molecular, a classificação dada para esse tipo de visualização é a visualização científica (Mazza, 2009).

A visualização científica é a forma mais eficiente de compreender os fenômenos físicos provenientes de modelos matemático e simulações que podem ser captados por sensores, satélites, exames médicos, etc. Além dos modelos matemáticos, a visualização cientifica utiliza recursos como a adição de cores e outras indicações visuais para facilitar o entendimento do objeto estudado e esses recursos são essenciais para representar de forma mais clara e objetiva, o que se pretende apresentar. Na Figura 3.3 é possível observar o uso das técnicas de adição de cores, tanto no caso do DNA quanto na representação do buraco na camada de ozônio e especificamente no segundo caso, outra técnica pode ser notada que é a "Isosurfaces"que é utilizada para representar a ideia de volume ou tamanho (Zhu; Chen, 2005). Ao representar o buraco da camada de ozônio Mazza (2009) destaca de cor azul uma área da figura que seria a representação o seu tamanho e as áreas da cor predominantemente verde, seria o restante do globo, com isso fica mais fácil entender a proporção do tamanho do buraco da camada de ozônio em relação ao globo terrestre.

3.2.2 Visualização de Redes

A visualização de redes, sendo elas sociais ou complexa, é considerada uma sub-área da visualização de informação, onde a representação da rede se dá pela definição de um grafo onde os vértices são representados por pontos e arestas por linhas que conectam um par de vértices em um determinado plano. A exibição final da rede em uma plano, seja ele bi-dimensional ou tri-dimensional não é algo trivial, para que seja possível a extração de informações mais precisas e fieis, o posicionamento dos vértices precisam ser definidos por métodos que levem em consideração propriedades intrínsecas da rede como o grau do vértice por exemplo. Visualização de redes é um campo de pesquisa estudados a muitos anos, desde Moreno em 1932 com as primeiras representações gráficas de uma rede social até os dias atuais com o advento de softwares para gerar graficamente redes proveniente de uma base de dados.

Para a definição de métodos e/ou procedimentos para trabalhar com visualização de redes,

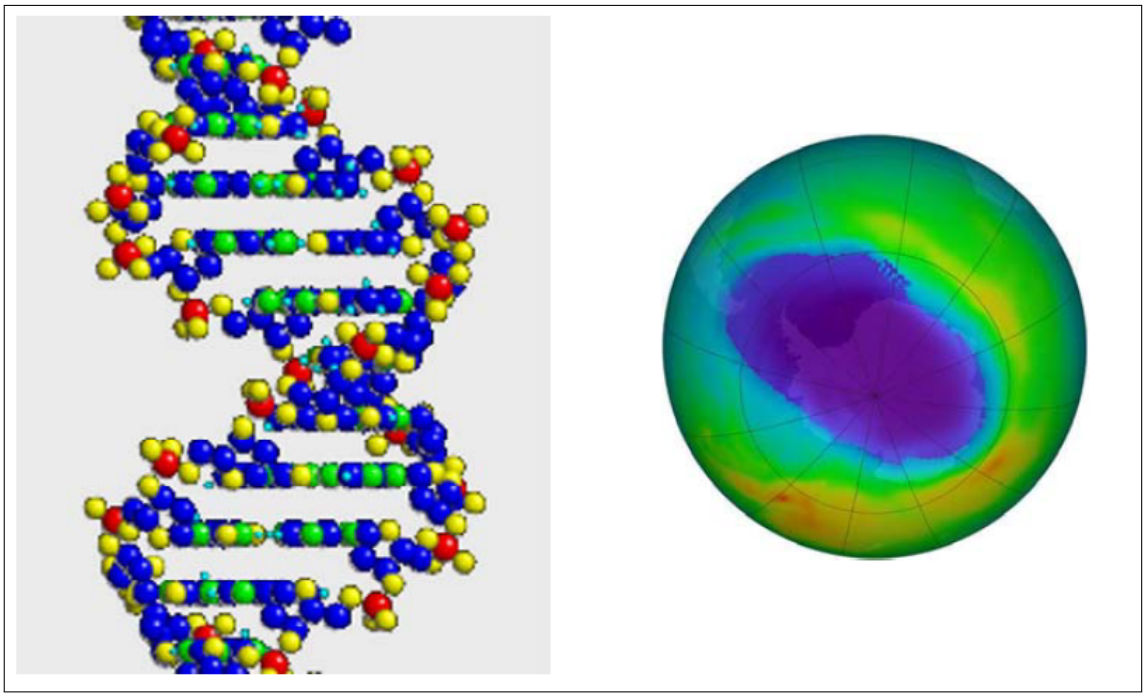

Figura 3.3: Imagem de dois exemplos de visualização científica. Estrutura de DNA a esquerda e a da direito representa o buraco da camada de ozônio. Fonte: Mazza (2009)

Mazza (2009) cita como necessário o entendimento de alguns conceitos. São eles:

- **Posicionamento dos vértices** - A visualização de rede representa tipo de dados abstratos e sem uma posição espacial natural e caso seja necessário usar algum critério de arranjo dos vértices no plano, faz necessário a utilização de algoritmos para realizar essa atividade. Considerando as coordenadas cartesianas, os vértices podem ser posicionados em uma, duas ou três dimensões no plano.
- **Representação das arestas** - A representação da aresta é a relação entre dois vértices que graficamente são representadas por linhas que podem ser direcionadas ou não. É possível associar a característica de peso a aresta modificando a cor da aresta ou até mesmo a largura dela.
- **Interações gráficas** - Com a grande complexidade de algumas redes, os usuários dos softwares de visualização, precisam ter disponíveis recursos que facilitem a manipulação da rede como o zoom, movimentação e reposicionamento dos vértices, etc.

A evolução do estudo de visualização de redes fez com que diversos pesquisadores desenvolvessem algoritmos para que a visualização das redes fossem geradas automaticamente. Esses algoritmos serão discutidos, com mais detalhe no capitulo 5. Neste estudo, serão abordados algoritmos das seguintes classificações:

- **Algoritmos de Força**
- **Estáticos e Circulares**
- **K-core**

3.2.2.1 Algoritmos de Força

Os algoritmos de força são conhecidos como force-direct algorithms (algoritmo de força direta - tradução livre) e seguem princípios físicos para calcular o posicionamento dos vértices da redes. Todo algorítimo baseado em force-direct tem como características forças de atração e repulsão entre os vértices (Brandes; Wagner, 2000). Na fórmula de repulsão, a inspiração vem de partículas elétricas carregadas que quando estão próximas umas das outras, exercem uma força de repulsão entre elas que pode ser calculada utilizando a fórmula ($F_r = k/d^2$) onde k é a constante da força e d é a distância ente o par de partículas. Então podemos inferir que a medida que a distância entre o par de partículas aumenta, a força de repulsão entre elas diminui.

Já a inspiração para a força de atração é um sistema de mola, que pode ser entendida da seguinte maneira: imagine uma mola e duas esferas uma em cada ponta, à medida que se puxa uma das esferas, o efeito da mola é exercido sobre a outra esfera atraindo a outra esfera para perto. Essa força de atração segue a fórmula ($F_a = -k * d^2$) onde k é a constante da foça e d é a distância entre o par de esferas. Então podemos inferir que a medida que a distância entre o par de esferas aumenta, a força de atração também aumenta (Chernobelskiy et al., 2012).

Esse tipo de algoritmo necessita de diversas iterações para que se possa encontrar o equilíbrio e tem como limite inferior e superior complexidade θ (n^2). A utilização desse modelo de algoritmos é utilizado com intuito de melhorar o arranjo dos vértices ao ser gerada graficamente a rede.

3.2.2.2 Estáticos e Circulares

Os algoritmos estáticos e circulares possuem características similares. O algoritmo de forma circular é uma especialização do algoritmo estático. Esses algoritmos têm por característica básica a realização do cálculo de prévio do posicionamento dos vértices no plano cartesiano sem a necessidade de diversas iterações para chegar ao resultado proposto. Por exemplo, para calcular o posicionamento dos vértices utilizando o algoritmo circular é necessário saber o raio máximo que o círculo terá de acordo ao plano disponível. Após

a definição do raio, Considerando que o menor angulo é zero e o maior 360 graus, é preciso calcular a posição dos vértices para que a disposição final seja circular. Uma vez calculado, esses valores, os mesmos não serão mais alterados. Esses algoritmos têm como limite inferior e superior complexidade θ(n) (Belviranli; Dilek; Dogrusoz, 2013).

3.2.2.3 K-Core

O termo K-core é proveniente do núcleo da rede onde o grau do vértice é representado pelo K e o core entende-se pelos vértices com maior grau na rede. Esse algoritmo tem como premissa que o posicionamento no centro do arranjo final da rede, os vértices que possuam o mais número de conexões. Para que tal arranjo seja possível, o algoritmo realiza três operações: primeiro é calculado a distribuição de graus da rede. De posse desse distribuição, o algoritmo realiza a segunda operação que é posicionar ao centro do arranjo os vértices com maior grau em formato circular. A terceira operação é o arranjo dos demais vértices, também em formato circular, ao redor do arranjo inicial formando uma sequencia de círculos ao redor do núcleo da rede. Esses algoritmos possuem o limite inferior e superior de complexidade $\theta(n^2)$ (Alvarez-hamelin et al., 2005).

Após o entendimento dos conceitos e tipos de visualização de informação, o próximo capítulo irá apresentar a proposta do modelo computacional.

Capítulo Quatro

Modelo Proposto

Como proposta de um modelo computacional capaz de gerar visualizações de redes sociais e complexas em ambiente bi-dimensional, iremos apresentar a proposta de macro arquitetura, padrão de projeto e estrutura de dados apontando as vantagens de seguir o modelo proposto.

A proposta do modelo é ser uma aplicação que realize as funções de calcular a posição dos vértices e arestas e ao final renderizar a rede utilizando componentes gráficos como o OpenGL 4.3 ou ser um componente capar de atender demandas específicas de outras aplicações. Para atender esses princípios o modelo foi desenvolvido de forma com que as funções estejam separadas em camadas e que não dependam de um outra camada para seu funcionamento.

4.1 Modelagem de Software

Como proposta de arquitetura para o modelo computacional adotamos o padrão de projeto MVC (3 camadas) no intuito de facilitar a manutenção, evolução do modelo e desacoplamento de código, atribuindo diferentes atividades a cada camada da aplicação e a criação de métodos genéricos que pudessem ser utilizado por diversos algoritmos. Conforme a Figura 4.1 as 3 camadas são: Interface Visual, Controlador e Biblioteca de Algoritmos e serão descritas a seguir.

- Interface Visual: Módulo responsável pela renderização de componentes a partir da leitura de posição dos vértices no plano cartesiano, e a relação de ligação entre eles, que são as arestas. O módulo de visualização é composto por 2 funcionalidades, a primeira responsável por criar uma via de comunicação com o usuário, entendendo os comandos passados por ele via mouse, teclado ou comandos via console, para execução de um determinado algoritmo. A segunda funcionalidade é composta por um loop que permanece em execução durante todo ciclo de vida do funcionamento do software, composto por comandos de renderização dos componentes, aplicação de cor entre outros.
- Controlador: Tem como característica ser o controlador onde é responsável por atender a solicitações do usuário como a escolha do algoritmo desejado, carregamento da rede a partir de uma arquivo .net (padrão do Pajek), manipulação dos vetores que armazenam o posicionamento dos vértices e armazenam das arestas. Esse módulo

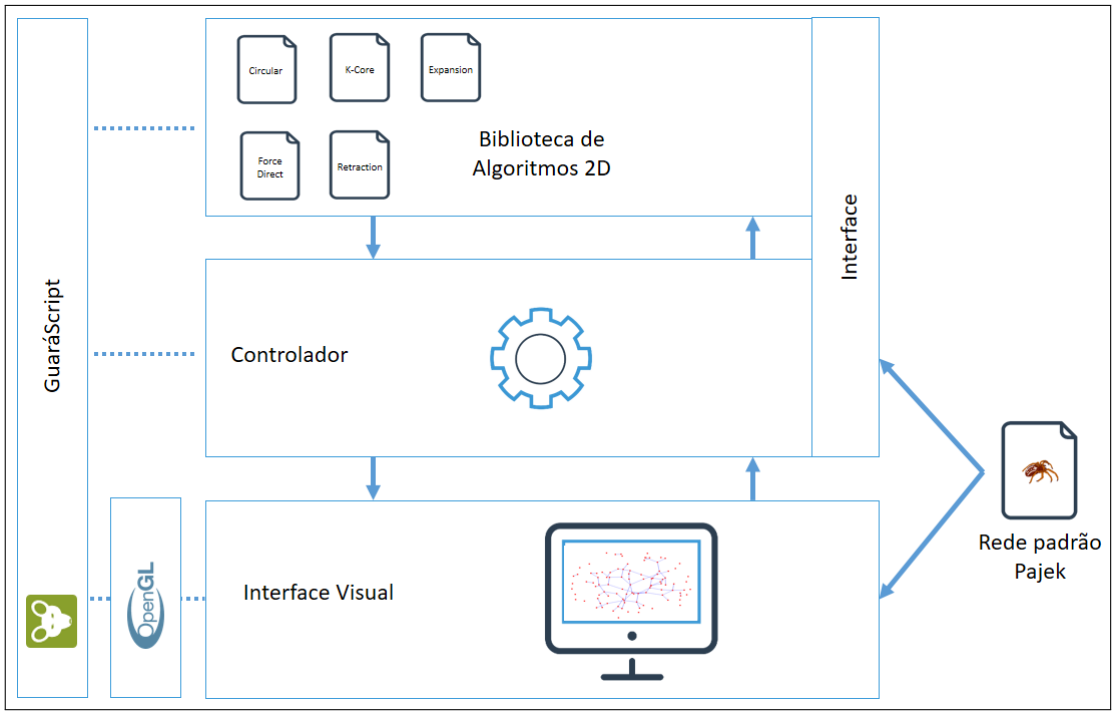

Figura 4.1: Macro arquitetura do modelo computacional. Fonte: Próprio autor

atua como uma interface de comunicação entre o módulo de visualização e o módulo dos algoritmos, desacoplando os módulos e facilitando a integração de qualquer algoritmo que se deseje. Nesse módulo, estão funções necessárias para obtenção os vetores de dados do posicionamento dos vértices, realização de cálculos, conversão de valores, etc.

- Módulo de Algoritmos: Esse módulo atua como uma biblioteca de algoritmos, onde todos os algoritmos implementados em GuaráScript, são armazenados e disponibilizados para os outros módulos. Para o desenvolvimento de um novo algoritmo de visualização de redes, o mesmo precisa ser escrito em linguagem GuaráScript, conhecer as funções de integração do Módulo de Serviços, para não serem reimplementados desnecessariamente, e fornecer ao final de sua execução, os vetores necessários para a geração da rede no Módulo de Visualização.

A Figura 4.1 além de demonstrar a separação das camadas da arquitetura, apresenta como proposta de linguagem de programação para todas as camadas o GuaráScript 4.2 que já tem implementado em seu core a API do OpenGL 4.3 que será responsável pela renderização dos componentes gráficos na camada de Interface Visual. Outra proposta do modelo é a disponibilização de uma interface que pode ser acessada por outra aplicação apenas respeitando os protocolos de comunicação com as camadas para que obtenha o resultado esperado sem a necessidade de conhecimento do funcionamento interno. Essa interface está disponível na camada Controlador e de Biblioteca de Algoritmos. Outra questão apresentada na Figura 4.1, é que o arquivo padrão para leitura da rede, é o

padrão de arquivo usado no software Pajek. O mesmo se comunica tanto com a camada de visualização através de comandos informados pelo usuário, bem como através da chamada via interface da camada Controlador para obtenção dos vértices e arestas da rede.

Essa separação em camadas é necessária devido ao princípio de que o funcionamento delas sejam independentes e possam ser facilmente integradas com outro tipo de solução.

Após a leitura do arquivo no padrão do Pajek, o modelo computacional propões a adoção de estrutura de dados em formado de vetor, onde cada posição do vetor é equivalente a 1 vértice. Considerando que a proposta do modelo computacional é a exibição de redes em ambiente bi-dimensional, se no arquivo tiver 5 vértices, serão criados dois vetores de tamanho 5, uma para o eixo X e outro para o eixo Y conforme Figura 4.2, onde em cada posição do vetor irá conter a posição que o vértice será posicionado no plano cartesiano.

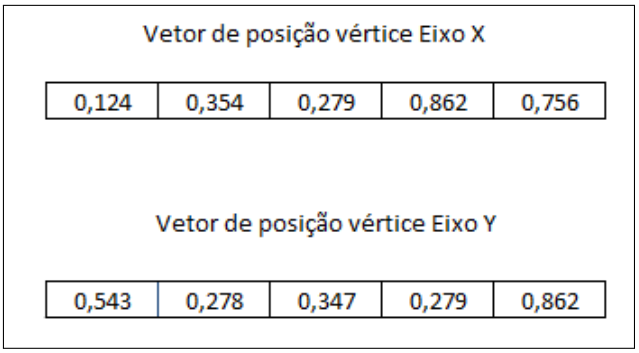

Figura 4.2: Representação dos vetores de posição do vértices do eixo X e Y. Fonte: Próprio autor

Para identificação da ligação entre os vértices, as arestas, a proposta também é um vetor, só que a cada par de posição do vetor, seria a representação das arestas. A opção dessa estrutura de dados em detrimento da matriz de adjacências se dá pelo fato de que há muito desperdiço de memória conforme exemplificado na Figura 4.3 onde é preciso ocupar espaço em memória para demonstrar que não há ligação do vértice A para o vértice C por exemplo. Já no vetor de arestas só é necessário haver a ligação entre o vértices, então para representar a ligação entre o vértice A ao vértice B, na primeira posição do vetor precisa ter a referencia para o vértice A e a segunda posição do vetor, conter a referência para o vértice B poupando recurso computacional.

4.2 GuaráScript

De acordo com Monteiro (2005) a linguagem GuaráScript foi desenvolvida voltada para a resolução de problemas científicos, pois dispões de ferramentas para cálculos numéricos avançados. O seu desenvolvimento levou em consideração características de 4 outras

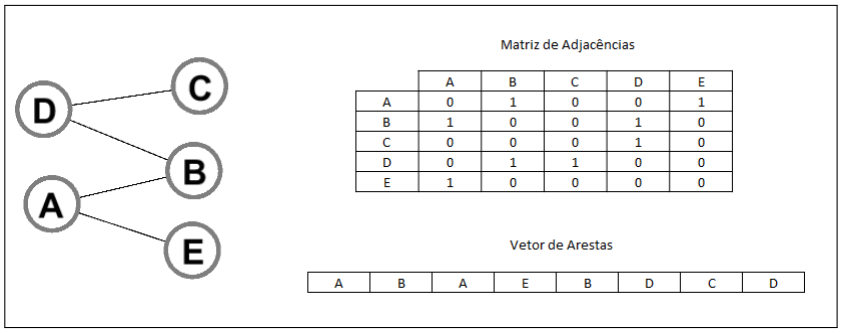

Figura 4.3: Comparativo entre matriz de adjacências e vetor de arestas. Fonte: Próprio autor

linguagens de programação, sendo elas C, TCL, JavaScript e Octave.

Monteiro (2005) percebeu em sua pesquisa, que apesar da comunidade científica dispor de diversas linguagens de programação, naquele momento, nenhuma delas reunia todas as características necessárias para o desenvolvimento científico. Ele percebeu que com frequência aplicações desenvolvidas em TCL que necessitavam exibir gráficos em ambiente tri-dimensional, usavam bibliotecas da linguagem C.

A proposta do GuaráScript é ser uma linguagem híbrida, totalmente escrita em ANSI C, sob licença GNU GPL contendo características das linguagens citadas anteriormente. O objetivo inicial da linguagem era além de fornecer uma ferramenta para operação com matrizes, números complexos, equações lineares, dispor de subsídios para que a mesma pudesse evoluir adicionando novas funcionalidades. A linguagem tem por paradigma a programação estruturada, e a mesma não precisa ser compilada, ela dispõe de um interpretador para executar os comandos escritos e possui portabilidade para qualquer plataforma que disponha de um compilador ANSI C disponível.

4.3 OpenGL

OpenGL é uma tecnologia definida como uma biblioteca de funções gráficas responsável por criar projeções gráficas com excelente qualidade e rapidez, tendo como desenvolvedora a *Silicon Graphics, Inc* empresa conhecida pela qualidade de seus algoritmos para computação gráfica.

OpenGL não se trata de uma linguagem de programação, de acordo a especificação da tecnologia (Segal; Akeley, 1994), ela é denominada de API (*Application Programming Interface*) e é responsável pela criação de projeções gráficas em ambiente bi-dimensional (2D), e tri-dimensional (3D). Para sua utilização, é necessário que ela seja adicionada ao pacote de bibliotecas de uma linguagem de programação - no caso da pesquisa, o

GuaráScript. Assim ao realizarmos a escrita do programa na linguagem escolhida, realizamos chamadas às funções para realização de tarefas.

Com OpenGL é possível não só a criação de personagens para Games, Filmes, etc. mas também o uso de formas primitivas como linhas, polígonos e pontos. A biblioteca de funções do OpenGL suporta a função de tratamento de iluminação sobre os componentes renderizados, mapeamento de textura, animação, transparência, rotação, e diversos outros recursos especiais.

4.4 SCNTools

O SCNTools é um software que agrega diversas ferramentas para auxílio no estudo das redes sociais e complexas no intuito de facilitar o entendimento do comportamento das redes, provendo funções para realização de cálculos e a possibilidade de realização de simulações sobre as redes (Monteiro et al., 2010).

O SCNTools foi desenvolvido especificamente para a tese de doutorado de Monteiro (2012) que tinha como objetivo facilitar a difusão do conhecimento para aumentar a competitividade de empresas componentes do APL e foi utilizado para geração de insumos para definição de características topológicas das redes do APL.

De acordo com Monteiro (2012), o SCNTools dispões de funcionalidades como:

- Cálculo de afinidade existente entre os atores de uma rede;
- Cálculo de distribuição de graus médios, a partir de uma amostra da rede;
- Simula a troca de conhecimento em uma rede de afinidade;
- Remove hubs, vértices ou arestas de uma rede;
- Converte um arquivo de rede no formato CSV para um arquivo no formato suportado pelo PAJEK;
- entre outros.

Neste capítulo foi apresentado a proposta de um modelo computacional capaz de gerar visualizações de redes em ambientes bi-dimensionais que serviu de base para o desenvolvimento do software DrawNet desenvolvido em GuaráScript e no próximo capítulo serão apresentados os algoritmos que farão parte da aplicação desenvolvida a partir da do modelo computacional apresentado.

Capítulo Cinco

Algoritmos de Visualização de Redes Sociais e Complexas

Desde de 1932, depois que Jacob Moreno fez uso de imagens para exemplificar uma rede social estudada a época, se analisa-se e propõe novas formas de visualizar graficamente redes sociais e complexas. Os pesquisadores foram percebendo com o passar do tempo que seria necessário a criação de algoritmos para tratar de forma metódica os dados provenientes das redes estudadas.

Em seu artigo denominado de *Graphical Techniques for Exploring Social Network Data* Freeman (2004) afirma a importância da criação de técnicas que apresentassem gráficos com fidelidade aos dados obtidos da rede, mesmo que a técnica seja aplicada diversas vezes sobre o mesmo conjunto de dados. Ele afirma que sem a criação de padrões para gerar graficamente a rede, procedimentos distintos seriam criados para cada massa de dados não sendo possível reaproveitá-la.

Se considerarmos que as redes estudadas atualmente são, em sua maioria, grandes e densas, os algoritmos de visualização de redes são determinantes para o surgimento de propriedades e/ou características da rede estuda possíveis de serem capturadas através de uma inspeção visual.

Neste capítulo, serão abordados diversos tipos de algoritmos onde serão descritos em pseudo código utilizando o paradigma estruturado, o mesmo paradigma da linguagem de programação GuaráScript. Os algoritmos que serão apresentados encontram-se inseridos nas classificações demonstradas no capitulo 3.

5.1 Algoritmo Force-Direct Fruchterman Reingold

Os algoritmos de Força, são muito utilizados por pesquisadores e estudiosos, pois busca apresentar uma arranjo da rede bem distribuído espacialmente. Fruchterman e Reingold (1991) descrevem que deve haver a preocupação em atender 5 critérios estéticos para aceitação do algoritmo, são elas:

- Distribuição dos vértices uniformemente no espaço - o algoritmo se preocupar em não deixar que os vértices fiquem amontoados ou sobreposto, buscando assim um

arranjo visual melhor arrumado.

- Minimização de cruzamento de arestas - como o algoritmo utiliza o conceito de força, os pares de vértices que possuem conexões ficariam concentrados próximos uns aos outros, o que por consequência diminui o cruzamento de arestas.
- Fazer arestas com tamanhos uniformes - o algoritmo utiliza duas forças concorrentes (atração e repulsão) que em um determinado momento se equilibram. Como as forças são iguais para todos os pares de vértices, a tendencia é que o tamanho das arestas sejam similares.
- Refletir inerente simetria - a partir do momento que o algoritmo distribui uniformemente os vértices, minimiza o cruzamento e mantem o tamanho similar das arestas, a visão geral da rede é simétrica tendo em vista que é possível ver similaridade de arranjo de toda a rede.
- Em conformidade com o espaço - O algoritmo implementa condições de contorno, que nada mais é que evitar com que os vértices saiam do plano definido, fazendo com que a rede seja arranjada de acordo ao espaço destinado.

Ao desenvolver o algoritmo Fruchterman e Reingold (1991) não se preocuparam rigidamente pelos princípios citados, mas os autores puderam perceber que o mesmo atende de maneira satisfatória esses princípios estéticos. O algoritmo segue dois macro princípios:

- Vértices ligados por uma aresta devem ser posicionados perto um do outro;
- Vértices não devem ser desenhados muito próximos um dos outros.

O pseudo código desse algoritmo será descrito a seguir e está dividido em três partes para melhor entendimento. Na implementação, essas partes são equivalentes a funções. Na primeira parte está descrito o funcionamento da força de repulsão de um vértices para os demais. Na segunda parte está descrito o funcionamento da força de atração que somente acontece quando um vértice é vizinho do outro, caso não seja, não exite atração entre eles fazendo com isso que a rede tenha um preenchimento uniforme no espaço. Na terceira parte, está descrita a atualização de posição dos vértices no espaço. A configuração inicial da disposição de vértices não foi descrita, pois no geral a posição inicial dos vértices é atribuída aleatoriamente (Algoritmo de posicionamento aleatório disponível na seção 5.4).

A primeira parte do algoritmo que trata do cálculo de repulsão dos vértices é composto por dois *loops*, sendo o primeiro responsável por indicar o vértice de referência para calcular sua repulsão com os demais. No segundo *loop*, é realizado um teste (linha 3) para que não seja aplicada força de repulsão sobre ele mesmo. Caso a condição seja atendida, o

algoritmo inicia a realização dos cálculos. É preciso calcular o distanciamento do vértice de referência para o vértice atual, levando-se em consideração os dois eixos (X e Y). A distância entre os vértices é conseguida através do resultado da subtração da posição do vértice de referência com a posição do vértice corrente. Com intuito de deixar o pseudo código mais simples e com isso facilitar o entendimento, a função de calcular o distanciamento dos vértices foi encapsulada por dois métodos que aparecem na linha 4 e 5 e são: distanciaEixoX(i,j) e distanciaEixoY(i,j) que esperam como parâmetro o vértice de referência e o vértice corrente e retorna o valor da distância por eixo. Os valores retornados por cada função é armazenado temporariamente por duas variáveis de nomes distVerticesEixoX e distVerticesEixoY.

Como o pseudo código foi implementado considerando apenas duas dimensões, é necessário que se calcule o valor da distância ao quadrado, obtida pela equação ($d = \sqrt{(dx^2 + dy^2)}$) conforme linha 6 onde d é a distância ao quadrado dos vértices, o comando *sqrt* é uma função que representa raiz quadrada e dx e dy são as distâncias respectivas pelo sue eixo. Após obtido o valor do distanciamento ao quadrado, o valor é utilizado para cálculo da força de repulsão conforme linha 7 através do método calculaRepulsao(). A fórmula utilizada para cálculo da força de repulsão foi descrito na seção 3.2.2.1 e nesse caso está encapsulada pela função calculaRepulsao() que recebe por parâmetro o valor do distanciamento ao quadrado que se encontra na variável distQuad na linha 6 e executa o cálculo utilizando a fórmula ($F_r = k/d$) armazenando o resultado do cálculo na variável *fr* onde k é um valor constante de força e d é representado na fórmula como o distanciamento ao quadrado. Após obter os valores de força de repulsão e a distância ao quadrado, é preciso realizar um cálculo para atualização dos vetores de força para o eixo correspondente observado na linha 8 e 9. O cálculo consiste em somar o valor da força armazenado no vetor ao resultado da divisão da distância entre os vértices por eixo pela distância ao quadrado, multiplicado pela força de repulsão previamente calculada. Com isso, teremos atualizado os vetores de força para os dois eixos que serão utilizados nas outras partes do algoritmo. O calculo de repulsão do algoritmo pode ser comparada com a implementação em GuaráScript pelo DrawNet no Apêndice D da linha 92 a 118.

A segunda parte do algoritmo que trata do cálculo de atração do vértices vizinhos é composto por um *loop* que irá percorrer o vetor das arestas por eixo. O algoritmo não utiliza a estrutura de matriz de adjacências, mas sim um vetor onde o par de itens é a referência de vizinhança ou conexão. Então o item da posição 0 do vetor e o item da posição 1 onde cada item representa um vértice, são conectados e por consequência são vizinhos, o mesmo se aplica para o item da posição 2 e 3 do vetor de arestas e assim sucessivamente, e com isso a força de atração poderá ser aplicada entre o par. O *loop* dessa função inicia na linha 1 e termina na linha 10 e a cada iteração o índice representado pela varável i é incrementado em 2 posições para assim executar o procedimento nos vértices conectados. Nas linhas 2 e 3 é realizado o procedimento para cálculo de distanciamento entre os

Algorithm 1 Force Direct - Fruchterman (Parte 1 - Força de Repulsão)

```
1:  for i←0 to quantidadeVertices do
2:    for j←0 to quantidadeVertices do
3:      if i != j then
4:        distVerticesEixoX←distanciaEixoX(i,j)
5:        distVerticesEixoY←distanciaEixoY(i,j)
6:        distQuad←sqrt(distVerticesEixoX^2 + distVerticesEixoY^2)
7:        fr←calculaRepulsao(distQuad)
8:        forcaEixoX[i]←forcaEixoX[i] + (distVerticesEixoX /distQuad) * fr
9:        forcaEixoY[i]←forcaEixoY[i] + (distVerticesEixoY /distQuad) * fr
10:     end if
11:   end for
12: end for
```

vértices vizinhos por eixo e atribuído as variáveis *distVerticesEixoX* e *distVerticesEixoY*. Na linha 4 é realizado o cálculo do distanciamento ao quadrado, mesmo procedimento realizado na primeira parte do algoritmo que consiste em somar as distâncias por eixo ao quadrado e calcular a raiz quadrada do resultado obtido pela soma. O valor final é atribuído na variável *distQuad*.

Depois que o cálculo da distância ao quadrado é calculado, o algoritmo calcula a força de atração entre os vértices através da função calculaAtracao() que recebe como parâmetro a distância ao quadrado e retorna o resultado para a variável *fa* conforme visto na linha 5. A função calculaAtracao() encapsula o cálculo da atração que segue a fórmula ($F_a = -k * d^2$) onde k é a constante da foça e d é a distância entre o par de vértices. Após obter os valores de força de atração e a distância ao quadrado, é preciso realizar um cálculo para atualização dos vetores de força para o eixo correspondente observado nas linhas de 6 a 9. A forma de atualização de valores é similar a realizada ao atualizar os valores da força de repulsão com a diferença que o processo precisa ser executado para os dois vértices por eixo o que faz com que seja simulado a força de atração entre eles. O calculo de atração do algoritmo pode ser comparada com a implementação em GuaráScript pelo DrawNet no Apêndice D da linha 120 a 143.

A terceira parte do algoritmo trata-se da atualização de posição dos vértices nos respectivos vetores. Na linha 1 se inicia o *loop* que irá percorrer todo o vetor de vértice. A cada iteração, será atualizado a posição do vértice atual tanto no eixo x quanto no eixo y. O processo de atualização dessas posições se inicia obtendo os valores de distanciamento que o vértice teve após as aplicações das forças de atração e repulsão sobre eles. Então na linha 2 e 3 esses valores são atribuídos as variáveis *distVerticesEixoX* e *distVerticesEixoY* que servirão de insumo para realização do cálculo (linha 4) da distância quadrada desses distanciamentos. Por último é realizado o cálculo para atualização da posição do vértice que será exibida em tela que é visto nas linhas 5 e 6 onde a posição

Algorithm 2 Force Direct - Fruchterman(Parte 2 - Força de Atração)

```
1: for i←0 to quantidadeArestas do
2:     distVerticesEixoX←distanciaEixoX(i,i+1)
3:     distVerticesEixoY←distanciaEixoY(i,i+1)
4:     distQuad←sqrt(distVerticesEixoX^2 + distVerticesEixoY^2)
5:     fa←calculaAtracao(distQuad)
6:     forcaEixoX[arestas[i]]←forcaEixoX[arestas[i]] - (distVerticesEixoX/distQuad) *fa
7:     forcaEixoY[arestas[i]]←forcaEixoY[arestas[i]] - (distVerticesEixoY)/distQuad) *fa
8:     forcaEixoX[arestas[i+1]]←forcaEixoX[arestas[i+1]] + (distVerticesEixoX/distQuad) *fa
9:     forcaEixoY[arestas[i+1]]←forcaEixoY[arestas[i+1]] + (distVerticesEixoY/distQuad) *fa
10:    proximoParVertices()
11: end for
```

do vértice antes da aplicação das forças é somado ao resultado da equação que divide o distanciamento do vértice em cada eixo pelo distanciamento quadrado vezes o limite de distanciamento representado pela constante *limiteDist* que precisa ser definido antes de realizar a execução do algoritmo. A variação no valor da constante *limiteDist* fará com que o algoritmo convirja mais rápido ou menos rápido. Quanto mais o valor dessa constante, mais rapidamente o algoritmo irá convergir, mas também fará com que diminua a precisão em deixar a disposição da rede uniforme. Algumas medidas podem ser tomadas para contornar o problema da perda da uniformidade da rede, como o teste para ver se o limite máximo de distanciamento entre os vértices vizinhos já foi alcançado para que não seja aplicado mais nenhuma força sobre ele. Os teste para ver se o distanciamento dos vértices foi atingido, não foi descrito no pseudo código, sendo necessário a criação no momento de sua implementação. A atualização de posição dos vértices no plano pode ser comparada com a implementação em GuaráScript pelo DrawNet no Apêndice D da linha 67 a 90.

Algorithm 3 Force Direct - Fruchterman(Parte 3 - Atualizando Posições)

```
1: for i ← 0 to quantidadeVertices do
2:     distVerticesEixoX←forcaEixoX[i]
3:     distVerticesEixoY←forcaEixoY[i]
4:     distQuad←sqrt(distVerticesEixoX^2 + distVerticesEixoY^2)
5:     posVerticesEixoX[i] ← posVerticesEixoX[i] + distVerticesEixoX / distQuad * limiteDist
6:     posVerticesEixoY[i] ← posVerticesEixoY[i] + distVerticesEixoY / distQuad * limiteDist
7: end for
```

A Figura 5.1, demonstra a disposição final da execução do algoritmo. Conforme visto na explicação do algoritmo, os vértices que não possuem conexões são repelidos por todos os outros vértices, mas nenhuma força de atração é aplicado sobre eles pois a força de atração é somente aplicada em vértices conectados, isso faz com que esses vértices sejam dispostos mais próximos a margem da tela. Agora se observar o núcleo conectado da rede, o comprimento das arestas tem mais ou menos o mesmo comprimento, dando a

impressão de uniformidade da rede e seguindo os princípios abordados por Fruchterman e Reingold (1991) de que os vértices conectados precisa estar mais próximos mas mantendo um distanciamento dos vértices aos quais não são conectados.

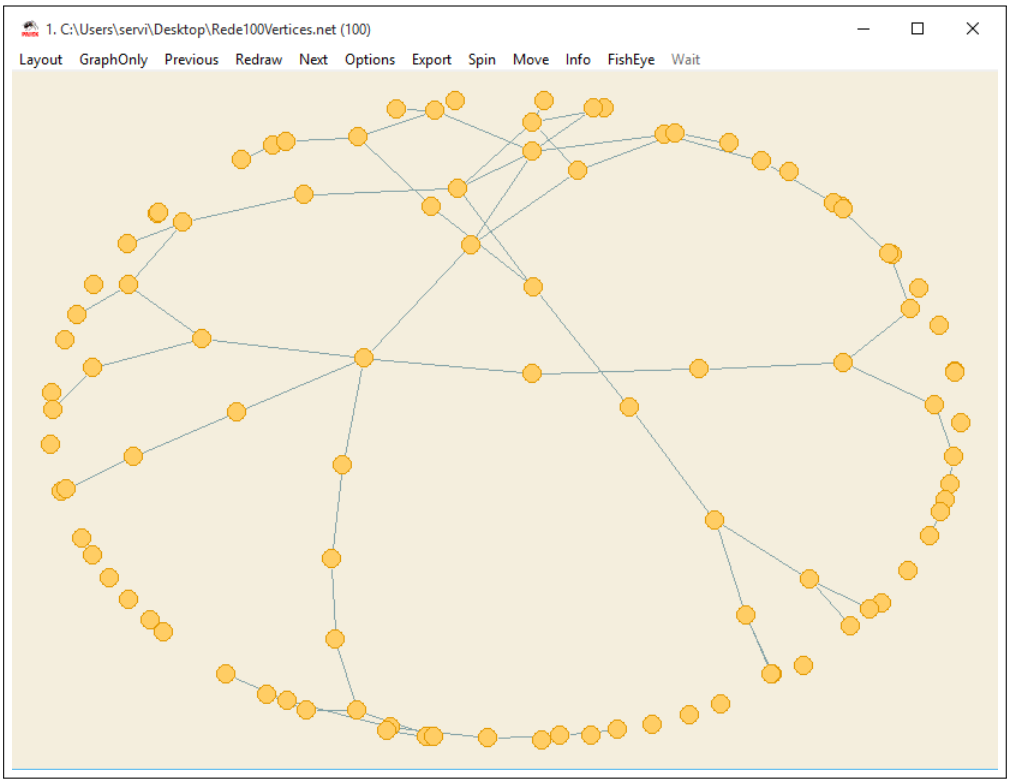

Figura 5.1: Exemplo do algortimo de Fruchterman executado no Pajek. Fonte: Próprio autor

5.2 Circular Layout

O algoritmo Circular layout tem como objetivo dispor os vértices em formato circular conforme Figura 5.2 aproveitando o raio máximo disponível da tela no menor eixo. Por exemplo, se a tela tem a resolução de 800x600 pixels, o algoritmo irá considerar o raio de 300 pixels que é a metade do diâmetro máximo possível da circunferência. A seguir é possível ver o pseudo código do algoritmo:

O algoritmo inicia com a declaração de variáveis que serão utilizadas durante a iteração que irá calcular a posição de cada vértice. As variáveis *anguloInicial* e *anguloFinal* armazenam os valores que representam os ângulos necessários para o desenho circular da rede. Esses valores foram representados em graus, mas podem ser alterados para radiano que não haverá nenhum problema com o código. As variáveis *raioEixoX* e *raioEixoy* armazenam os valores de raio por eixo (eixo X e Y) onde os valores das variáveis largura e altura podem variar a depender do tamanho de tela disponível para a aplicação.

Algorithm 4 CIRCULAR LAYOUT

```
1: anguloInicial←0
2: anguloFinal←360
3: raioEixoX←(L/2)
4: raioEixoY←(A/2)
5: deslocamento←((anguloFinal - anguloFinal)/(quatidadeVertices -1))
6: menorR←min(raioEixoX,raioEixoY)
7: for i ← 0 to quantidadeVertices do
8:    anguloA ← (i * deslocamento)
9:    posVerticesEixoX[i]←(raioEixoX+((menorR-aJ)*cos(anguloInicial+anguloA)))/ L
10:   posVerticesEixoY[i]←(raioEixoY+((menorR-aJ)*sin(anguloInicial+anguloA)))/ A
11: end for
```

Depois da declaração dessas variáveis, é necessário a realização do cálculo que irá definir o deslocamento, em graus, de um vértice em relação ao outro. O deslocamento que será necessário aplicar entre os vértices é obtido através da equação $(d = (aF - aI)/(qV - 1))$ que é visto na linha 5, onde d é o deslocamento, aF é o angulo final, aI o angulo inicial e qV é a quantidade de vértice, então considerando que a rede contem 20 vértices e os ângulos inicial e final são 0 e 360 respectivamente então o deslocamento do vértice em relação ao seu vizinho é de 18 graus. A variável *menorR* receberá o retorno da função *min* que irá retornar o menor valor dos raios disponível para posicionamento dos vértices linha 6.

Após a inicialização das variáveis, o algoritmo inicia o *loop*, linha 7, que irá executar o bloco de código até que a quantidade de vértices seja atingida. A iteração do laço principal do algoritmo, é responsável por transformar os valores de deslocamento em graus, para a posição real no plano cartesiano. Então na linha 8 a variável *anguloAtual* armazenará o valor do grau atual que o vértice precisa ser posicionado. Levando em consideração ao cálculo de deslocamento apresentado como exemplo de 18 graus para 20 vértices, na primeira vez da execução do laço será realizado o cálculo onde i é o índice de referência para saber qual vértice se está trabalhando no momento que irá variar de 0 a 19, onde na primeira execução esse valor será 0.

Então com a multiplicação de i pelo *deslocamento* teremos o valor do *grauAtual* 0 que é a posição do primeiro vértice. Na segunda execução do loop, o cálculo irá resultar no *grauAtual* de 18, na terceira de 36 e assim sucessivamente. Com o valor do *grauAtual*, é necessária a conversão do valor de graus em pontos cartesianos para o eixo X e Y. Esse cálculo é realizado nas linhas 9 e 10 considerando valores normalizados para os eixos (valores que variam de 0 a 1), seguindo a equação $(p = (rX + (mR - aJ) * cos(aI + aA))/L)$ para o posicionamento do vértice no eixo X onde p é a posição do vértice, rX é o valor do raio do eixo X, mR é o menor raio, aJ é uma constante de ajuste para que o vértice não fique sobre a borda da tela, *cos* é referente ao cosseno, aI é o angulo inicial, aA é o angulo

atual e L é a constante do valor da largura da tela, que seria o mesmo valor de rX *2 e para o posicionamento do vértice no eixo Y a equação é $(p = (rX + (mR - a) * sin(aI + aA))/A)$ onde a diferença entre as equações é o cálculo do seno sin e a divisão pela constante A que representa a altura da tela. Após esses cálculos, os vetores $posVerticesEixoX$ e $posVerticesEixoY$ são atualizados com os valores de coordenadas que serão lidos pelo renderizador responsável por desenhar a rede na tela. O algoritmo implementado em GuaráScript pelo DrawNet pode ser analisado no Apêndice B. É possível verificar como fica o arranjo do algoritmo na Figura 5.2.

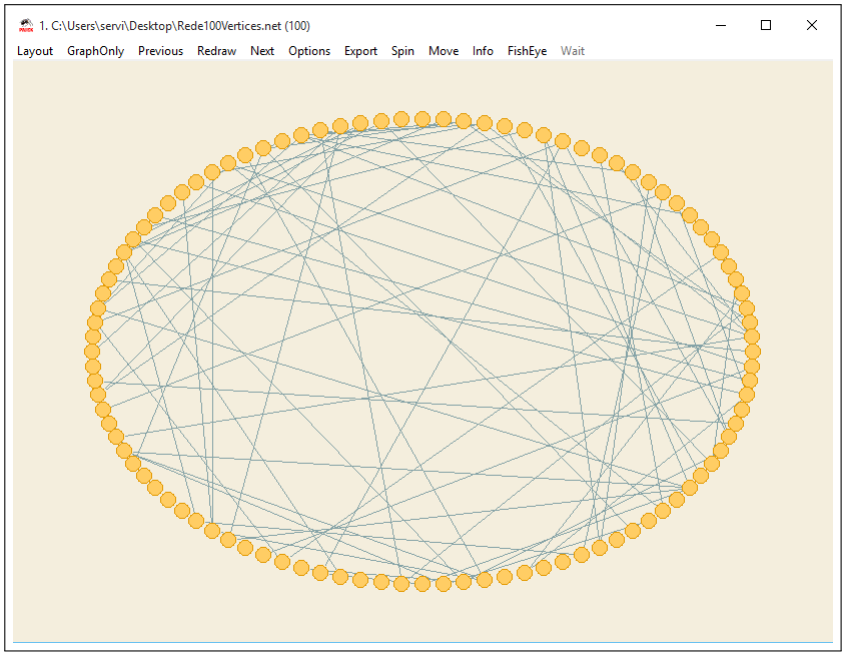

Figura 5.2: Exemplo do algoritmo de layout circular executado no Pajek. Fonte: Próprio autor

5.3 K-core

O algoritmo K-Core tem como objetivo dispor os vértices em formato circular, como o algoritmo Circular Layout 5.2, mas ao invés de apresentar apenas uma circunferência de vértices, são geradas diversas circunferências que podem ter o total igual ou menor ao grau máximo do vértice pertencente a rede. O objetivo do algoritmo é posicionar os vértice que possuem os maiores graus, mais ao centro da tela e os vértices que possuírem menor graus, posicionados mais próximos a borda da tela. Para que seja possível a disposição dos vértices atendendo a esse princípio, se faz necessário a realização do cálculo do grau de todos os vértices da rede para identificar o grau do vértice e também o grau máximo encontrado. Como o algoritmo tem como característica o formato circular, as variáveis encontradas entre as linhas 1 e 5, seguem a mesma lógica do algoritmo Circular.

O que difere o algoritmo k-core do algoritmo circular é a necessidade do cálculo do grau

dos vértices que no algoritmo esse cálculo é encapsulado na função *calculaGrauVertices()* que ao final da execução retorna um vetor com o grau de todos os vértices e armazena no vetor grauMaximo[] conforme visto na linha 6. O método *calculaGrauVertices()* está descrito no algoritmo 6. É possível a obtenção do grau dos vértices de diversas maneiras, se formos pensar na questão de otimização, mas essa não foi uma preocupação para o desenvolvimento dessa função, com isso tem 2 *loops* o primeiro que se inicia na linha 1 e o outro na linha 4, ambos irão percorrer o vetor de vértices para testar se o vértice atual, linha 3, é vizinho do vértice comparado, linha 5. Caso os mesmos sejam vizinho, a variável *grauVertice* é incrementada, então ao encerrar a execução do *loop* da linha 4, o *vertorGrauVertices* é atualizado com o grau do vértice atual e então a função passa a testar o próximo vértice.

Algorithm 5 K-Core

```
1: anguloInicial←0
2: anguloFinal←360
3: raioEixoX←(L/2)
4: raioEixoy←(A/2)
5: menorR←min(raioEixoX,raioEixoY)
6: grausVertices←calculaGrauVertices()
7: grauMaximo[]←calculaGrauMaximo()
8: for i←0 to grauMaximo do
9:    quantidadeVerticePorGrau←calculaQuantidadeVerticePorGrau(i)
10:   deslocamento←calculaDeslocamento(quantidadeVerticePorGrau)
11:   for j ← 0 to quantidaVertices do
12:      raioMaximo ← verificaRaioMaximoAtual(i)
13:      if grausVertices[j] = i then
14:         anguloA ← (proximoVertice * deslocamento)
15:         posVerticesEixoX[i]←(raioEixoX+((menorR-aJ)*cos(anguloInicial+anguloA)))/ L
16:         posVerticesEixoY[i]←(raioEixoY+((menorR-aJ)*sin(anguloInicial+anguloA)))/ A
17:      end if
18:   end for
19: end for
```

Algorithm 6 calculaGrauVertices

```
1: for i←0 to quantidadeVertices do
2:    grauVertice←0
3:    verticeAtual = i
4:    for j←0 to quantidadeVertices do
5:       if verticeAtual vizinho j then
6:          grauVertice←grauVertice+1
7:       end if
8:    end for
9:    vetorGrauVertices[i]←grauVertice
10: end for
```

Já o método *calculaGrauMaximo* percorre o vetor *grausVertices* para encontrar o maior valor de grau e armazena na variável *grauMaximo*. Esse algoritmo é composto por 2

loops linhas 8 e 12, onde no primeiro é obtido a quantidade de vértice que possuem o mesmo grau do índice i que está sendo iterado no primeiro *loop* que se faz necessário para o cálculo do deslocamento angular de um vértice para o outro, linha 10, muito semelhante ao algoritmo Circular. Quando o algoritmo passa para o segundo *loop*, é executada a função *verificaRaioMaximoAtual()* na linha 13, que recebe como parâmetro o grau atual e executa uma operação para que o raio máximo seja diminuindo a medida que o grau vai aumentando, com isso garante que as circunferências não sejam sobrepostas e se aproximem para o centro da tela e retorna para a variável *raioMaximo* o valor calculado. Por ultimo, o algoritmo testa na linha 13, se o grau do vértice analisado é o mesmo do grau atual, caso positivo, nas linhas 14 a 16 realiza o mesmo procedimento de atualização de posição do algoritmo Circular 5.2. A versão do algoritmo em GuaráScript pelo DrawNet pode ser conferido no Apêndica E. O resultado do algoritmo é visto na Figura 5.3.

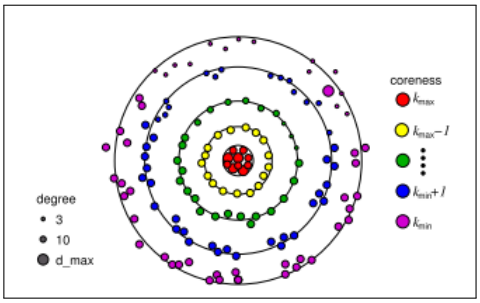

Figura 5.3: Figura retirada do artigo K-core decomposition: a tool for the visualization of large scale networks. Fonte: Alvarez-Hamelin et al. (2005)

Na Figura 5.3 é possível verificar os vértices respeitando uma hierarquia de localização, onde os vértices com menos grau tendem a ficarem mais próximos das bordas e os vértices de maior grau mais próximo ao centro da tela.

Essa visualização é muito interessante para se identificar vértices que possuem maiores importâncias nas rede. Caso se trate de uma rede social, esse grupo de pessoas podem representar potenciais disseminadores de informação.

5.4 Random Layout

O algoritmo Random Layout 7 é um algoritmo que por definição, preza pela simplicidade na geração de exibição de redes e geralmente é utilizado em diversos outros algoritmos para posicionar inicialmente os vértices na tela. A grande maioria das linguagens de programação implementam como função interna a geração de números aleatórios. No caso da linguagem GuaraScript a função *random* também faz parte do interpretador e no algoritmo ela é chamada na linha 4 dentro de um laço que só é encerrado quando a condição do número aleatório gerado é atendida que no caso o número precisa ser menor

do que o eixo X meno o ajuste de borda, caso a condição seja aplicada, o algoritmo passa para o segundo laço que executa o mesmo procedimento alterando apenas a condição que precisa atender ao eixo Y. Esse algoritmo é muito utilizado para ter uma visão geral da rede, sem se preocupar como os vértice serão dispostos em tela conforme é possível visualizar na Figura 5.4. A seguir é possível ver o pseudo-código do algoritmo:

Algorithm 7 RANDOM LAYOUT

```
1:  for i ← 0 to quantidadeVertices do
2:      condicao ← 0
3:      while condicao = false do
4:          valorAleatorioX ← posicaoAleatoria()
5:          if valorAleatorioX < (limiteEixoX - ajuste) then
6:              posicaoVerticesEixoX[i] ← valorAleatorio
7:              condicao ← true
8:          end if
9:      end while
10:     condicao ← false
11:     while condicao = false do
12:         valorAleatorioY ← posicaoAleatoria()
13:         if valorAleatorioY < (limiteEixoY - ajuste) then
14:             posicaoVerticesEixoY[i] ← valorAleatorio
15:             condicao ← true
16:         end if
17:     end while
18: end for
```

O algoritmo implementado em GuaráScript pelo DrawNet pode ser analisado no Apêncide F.

5.5 *Expansion e Retraction*

O algoritmo Expansion tem como objetivo afastar os vértices do centro do plano em diagonal para as bordas, provendo a perspectiva de que a rede está expandindo. O algoritmo Retraction tem objetivo inverso; os vértices são deslocados diagonalmente em direção ao centro do plano.

Para a execução desses algoritmos, é necessário que o cálculo de deslocamento seja feito considerando qual quadrante do plano cartesiano o vértice se encontra no momento, isso é necessário para saber se os valores de posição serão incrementados ou decrementados.

Na prática, esses algoritmos têm como objetivo o auxílio a outros, onde após a execução do algoritmo o usuário sente a necessidade de expandir ou retrair a rede para melhor visualização. Apesar desses dois algoritmos serem utilizados em sua maioria como auxiliar

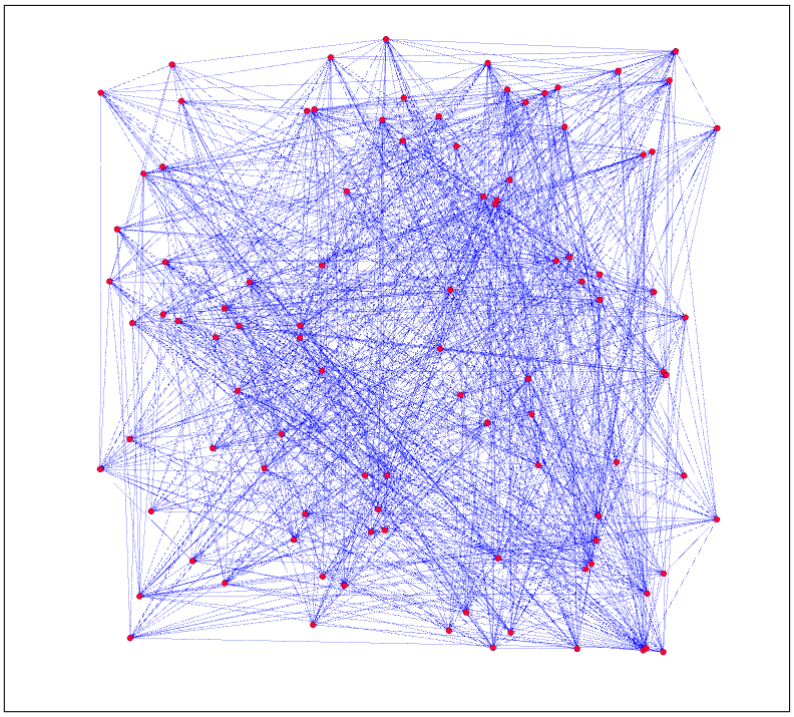

Figura 5.4: Representação da execução do algoritmo Random com rede de 100 vértices executado no Gephi. Fonte: Próprio autor

na execução de outro algoritmo, os mesmos podem ser utilizados sozinhos, basta apenas que os vértices estejam plotados no plano para usá-los.

A posição inicial dos vértices pode ser definida manualmente ou usando um algoritmo aleatório como mostrado no algoritmo Random Layout 7 ou um algoritmo com modificações para que os vértices sejam posicionados aleatoriamente próximos ao centro e para o Retraction estariam posicionados aleatoriamente afastados do centro para que o funcionamento do algoritmo seja satisfatório.

Se observarmos a Figura 5.6 o funcionamento dos algoritmos seguem o afastamento ou aproximação do centro do plano diagonalmente e isso faz com que alguns problemas sejam notados na execução dos mesmos. Se executarmos o algoritmo Expansion indefinidamente, o que irá acontecer é que os vértices chegaram na borda da tela e comecem a se sobrepor uns aos outros caso não seja implementando nenhuma condição para contornar tal problema. Já se executarmos o algoritmo Retraction indefinidamente, os vértices se acumularam ao centro da tela também sobrepondo os vértices.

Na Figura 5.5 temos a representação do plano cartesiano para melhor entendimento do funcionamento do algoritmo e é possível perceber que o eixo X e Y tem valores que variam de -1 e 1, onde no 1º quadrante os valores de X e Y são positivos 2º quadrantes valores positivos para Y e negativos para X, no 3º quadrantes valores negativos para X e Y e no 4º valores positivos para X e negativos para Y. Essas definições são a base para a

execução dos algoritmos, pois para saber que operação matemática será necessária aplicar, vai depender em que quadrante o vértice se encontra. Se um vértice tiver valor de posição para o eixo X=0.5 e Y= 0.7 e formos aplicar o algoritmo Expansion, os valores de X e Y serão incrementados proporcionalmente por uma constante o que fará com que o vértice seja deslocado diagonalmente para mais próximo a borda, e caso o algoritmo aplicado seja o Retraction os valores serão decrementados proporcionalmente pela mesma constante se aproximando assim ao centro do plano cartesiano.

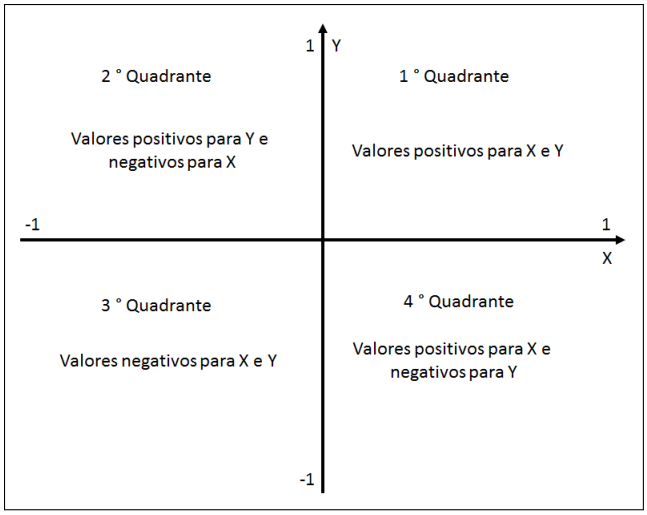

Figura 5.5: Representação do plano cartesiano com valores variando de -1 a 1 nos dois eixos. Fonte: Próprio autor

Como a maioria de softwares de visualização de redes não trabalha como valores negativos, o pseudo código dos algoritmos irá considerar a imagem 5.6 como referência os os valores de X e Y variam de 0 a 1 e o algoritmo Expansion tem o efeito mostrado pela seta na cor azul de se deslocar diagonalmente em relação aos eixos em direção as bordas e no sentido inverso para o algoritmo Retraction.

A seguir é possível ver o pseudo código do algoritmo:

O algoritmo 8 Expansion é composto por um *loop* principal que irá iterar até a quantidade de vértices e irá testar as 4 possibilidades possíveis para identificação em que quadrante se encontra o vértice. Esse 4 testes se encontram nas linhas 2, 5, 8 e 11 onde irá verificar o posicionamento dos vértices tanto para o eixo X quanto para o Y e identificar em que quadrando o mesmo se encontra. Após esses testes serão aplicados o incremente o decremento de acordo a posição do vértice. O algoritmo Retraction segue o mesmo principio apresentado com a diferença que onde no algoritmo Expansion a posição do vértice é incrementado, no algoritmo Retraction a posição do vértice é decrementado e vice versa. Conforme observado no pseudo código é verificado que o algoritmo possui complexidade $\theta(n)$. O algoritmo em GuaráScript implementado pelo DrawNet pode ser visto no Apêndice C e G. É possível ver o funcionamento dos algoritmos através das

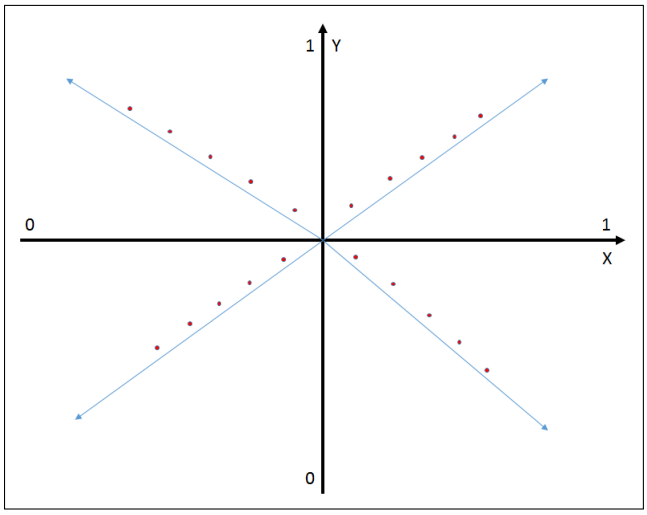

Figura 5.6: Representação do plano cartesiano com valores variando de 0 a 1 nos dois eixos. Fonte: Próprio autor

Figuras 5.7 e 5.8 onde a numeração indica a equivalência das etapas de evolução do posicionamento dos vértices à medida que algumas execuções são realizadas.

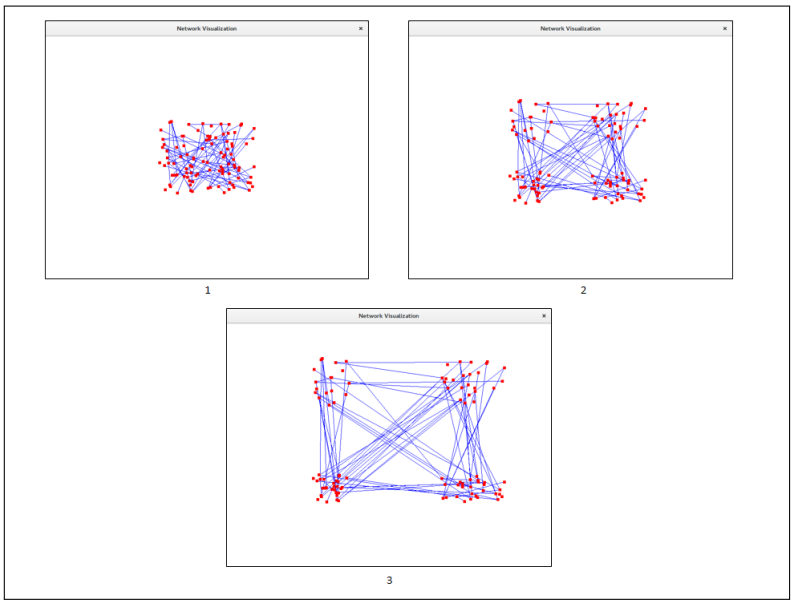

Figura 5.7: Figura representando a execução do algoritmo Expansion. Fonte: Próprio autor

Neste capítulo foi apresentado os algoritmos com suas específicas características com exemplos em pseudo código. No próximo capítulo serão analisados os resultados das visualizações obtidas através da aplicação desenvolvida em GuaráScript com os artigos científicos e os softwares Pajek e Gephi.

Algorithm 8 EXPANSION LAYOUT

```
1: for i ← 0 to quantidadeVertices do
2:     if posVerticeEixoX[i] > 0.5 and posVerticeEixoY[i] > 0.5 then
3:         posVerticeEixoX[i] = posVerticeEixoX[i] * constante
4:         posVerticeEixoY[i] = posVerticeEixoY[i] * constante
5:     else if posVerticeEixoX[i] > 0.5 and posVerticeEixoY[i] < 0.5 then
6:         posVerticeEixoX[i] = posVerticeEixoX[i] * constante
7:         posVerticeEixoY[i] = posVerticeEixoY[i] / constante
8:     else if posVerticeEixoX[i] < 0.5 and posVerticeEixoY[i] < 0.5 then
9:         posVerticeEixoX[i] = posVerticeEixoX[i] / constante
10:        posVerticeEixoY[i] = posVerticeEixoY[i] / constante
11:    else if posVerticeEixoX[i] < 0.5 and posVerticeEixoY[i] > 0.5 then
12:        posVerticeEixoX[i] = posVerticeEixoX[i] / constante
13:        posVerticeEixoY[i] = posVerticeEixoY[i] * constante
14:    end if
15: end for
```

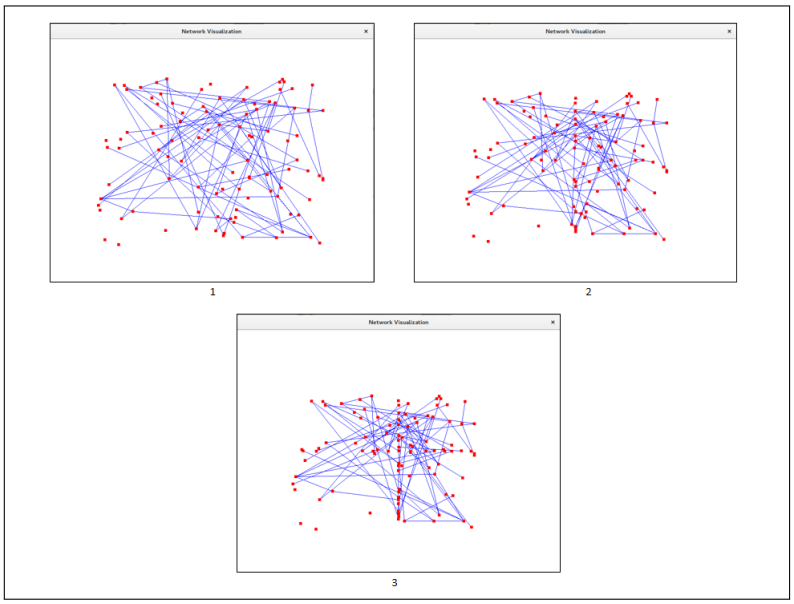

Figura 5.8: Figura representando a execução do algoritmo Expansion. Fonte: Próprio autor

Capítulo Seis

Resultado e Discussões

Neste capítulo serão abordados os resultados obtidos nas visualizações das redes com os algoritmos implementados em GuaráScript pelo DrawNet onde essas visualizações geradas serão comparadas com os resultados encontrados na literatura pelos autores dos algoritmos e também por mais 2 software de visualização de redes o Gephi versão 0.9.1 e Pajek versão 4.09.

O intuito dessas comparações é validar se as visualizações geradas pelo DrawNet estão condizentes com o que a comunidade cientifica espera e o que já é encontrada implementado em outros softwares.

A ideia é que os algoritmos tenham o comportamento esperado pelos autores ou pelos softwares que também os implementa com intuito de validá-lo. Os resultados serão exibidos com auxílio de Figuras comparativas.

6.1 Funcionamento do software Network Visualization

Nesta seção, será explicado como realizar as instalações ou configurações necessárias para o funcionamento do software DrawNet. Apesar da linguagem GuaráScript ser multi plataforma, esta seção utilizará como base o sistema operacional Linux Ubuntu. O primeiro passo é a instalação do Git que poderá ser feito via console usando o comando $apt-get install git e tem como objetivo o versionamento do código e neste caso específico, será utilizado para realizar o clone projeto do GuaráScript e do código fonte do software de visualização. O código fonte dos dois projetos, estão hospedados no GitHub um repositório público onde é possível fazer diversas operações inclusive realizado modificações e solicitação de alteração no repositório original. O GitHub é muito importante para o desenvolvimento de aplicações, pois com ele é possível o controle de versão dos arquivo para melhor gerenciamento do código fonte.

Após a instalação do Git, o próximo passo é o clone do projeto do GuaráScript pelo comando no console de $git clone https://github.com/guarascript/guash.git dentro da pasta que deseje que seja armazenado. Informações para realização da compilação e instalação do GuaráScript estão disponíveis no Readme dentro da pasta do projeto, ou na página do repositório no GitHub. Após a instalação padrão do GuaráScript, é necessário instalar o interpretador glwmguash via comando $sudo make install glwmguash para o perfeito funcionamento do software.

Após a realização da instalação do GuaráScript, será necessário o clone do projeto do DrawNet que também se encontra no repositório do GitHub utilizando o comando no console "$git clone https://github.com/rodrigopvb/mestrado.networkvisualizat ion.git"dentro da pasta que deseje armazenar o código fonte. Este projeto não precisa ser compilado, basta realizar a execução do arquivo "DrawNet.gua"dessa forma "./DrawNet.gua"mas é preciso informar os parâmetros para execução correta do software. Qualquer mudança na forma de executar o software será atualizada no arquivo Readme disponível no repositório do GitHub.

Para saber quais os parâmetros necessários para execução do algoritmo desejado, basta adicionar o parâmetro -help"para obter informações de quais os parâmetros necessários e quais os algoritmos suportados para execução vide Tabela 6.1.

O primeiro parâmetro é referente a escolha do algoritmo ao qual se deseja gerar a rede, e o segundo parâmetro é referente ao arquivo em formato .net - arquivo no formato do Pajek - que contem as informações dos vértices e arestas. No primeiro momento, o padrão de arquivo suportado pela aplicação é mostrado na Figura 6.1 que contém a quantidade de vértices e o par de vértices representando as arestas.

Tabela 6.1: Algoritmos Disponíveis na Aplicação

Nome	Descrição
fruchterman	Algoritmo de Força direta por Fruchterman e Reingold
circular	Circular Layout algorithm
random	Random Layout algorithm
expansion	Expansion Layout algorithm
retraction	Retraction Layout algorithm

6.2 Algoritmo de Force Direct - Fruchterman Reingold

O algoritmo Fruchterman Reingold implementa as forças de atração e repulsão seguindo a lei de Hooke's e os algoritmos implementos pelo DrawNet, no Pajek e no Gephi, também seguem o mesmo princípio, mas o comportamento e/ou disposição da rede pode variar para cada aplicação. Mesmo que a mesma rede seja aplicada, não é possível afirmar que as visualizações geradas serão iguais. O que podemos afirmar é que as visualizações terão características e comportamentos similares. Essas variações ocorrem por motivos como a disposição inicial do vértices que variam em cada software, tamanho do vértice em tela, aplicação ou não de condições que previnam a sobreposição de vértices, entre outros.

```
*Vertices 100
1 "Rótulo 1"
2 "Rótulo 2"
3 "Rótulo 3"
4 "Rótulo 4"
5 "Rótulo 5"
...
99 "Rótulo 99"
100 "Rótulo 100"
*Edges
4 16
18 19
4 20
1 22
9 23
8 24
8 27
```

Figura 6.1: Excerto de um arquivo .net do Pajek. Fonte: Próprio autor

O que será analisado é se o comportamento da execução do algoritmo atende as expectativas esperadas no artigo de (Fruchterman; Reingold, 1991). Para isso faremos a comparação de uma rede não direcionada de 100 vértices, que será a mesma aplicada em todos os algoritmos, e também serão geradas visualizações de 3 grafos completos K5, K6 e K8 os mesmo utilizados por Fruchterman e Reingold (1991) para ser analisado com os resultados obtidos no artigo. Grafo completo é um tipo de grafo onde todos os vértices estão conectados entre si.

Com exceção de (Fruchterman; Reingold, 1991) que especifica o tamanho do grafo completo em seu artigo, os demais autores não especificam o tamanho da rede ao qual seus algoritmos foram submetidos, com isso a definição do tamanho da rede foi realizada de forma empírica com intuito de facilitar a análise dos resultados.

Para realizar a primeira comparação foi utilizado uma rede com 100 vértices, 73 arestas e um grau máximo de 5. A rede será executada no software DrawNet, Pajek e Gephi e depois comparada com o artigo de (Fruchterman; Reingold, 1991) intitulado *Graph Drawing by Force-directed Placement*.

Apesar de pontuarmos para as possíveis variações que poderiam ocorrer para a geração da visualização e possível observar que a Figura 6.2 quadro 1 gerada pelo software DrawNet em relação a visualização gerada no Gephi quadro 2 existe grande semelhança em relação a disposição final do vértices que possuem conexões. Como dito por Fruchterman e Reingold (1991) um dos princípios esperados são que vértices conectados sejam exibidos próximos um dos outros, e isso á possível encontrar nas duas visualizações. Com relação aos vértices desconectados, os softwares podem aplicar tratamento para passar uma impressão mais linear como é possível observar na Figura 6.2 quadro 2 que dispões os vértices desconectados próximos aos vértices conectados seguindo um formato circular. Esse tipo

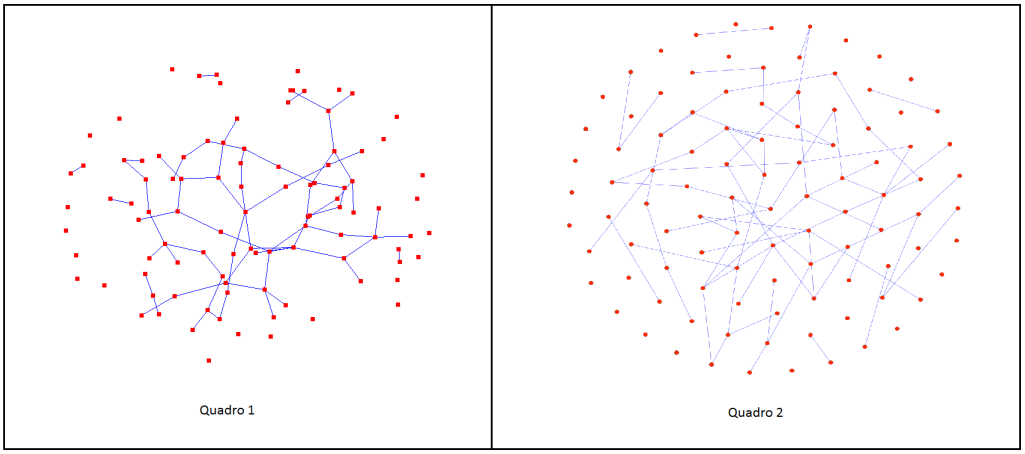

Figura 6.2: Figura comparando a imagem gerada pela aplicação DrawNet com a imagem gerada pelo Gephi. Rede com 100 vértices Fonte: Próprio autor

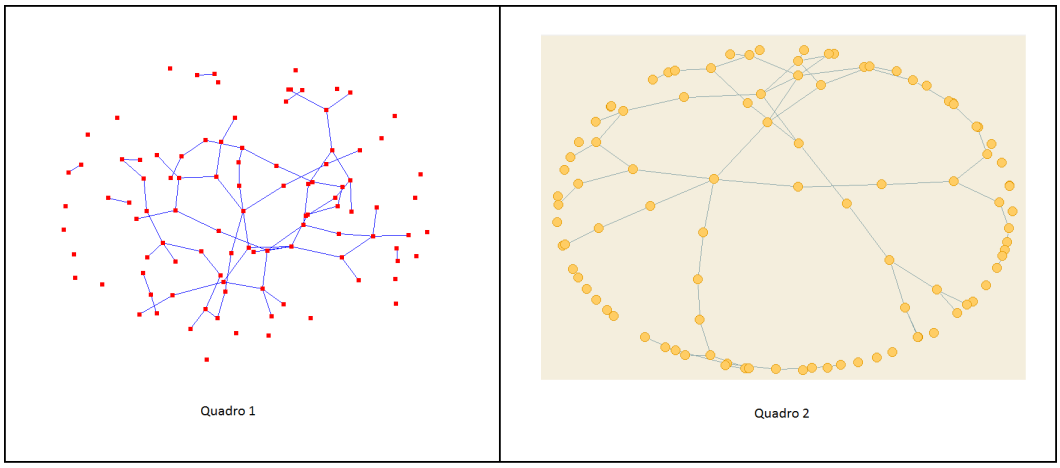

Figura 6.3: Figura comparando a imagem gerada pela aplicação DrawNet com a imagem gerada pelo Pajek. Rede com 100 vértices Fonte: Próprio autor

de tratamento não foi implementado no software desenvolvido em GuaráScript, tendo em vista que não era uma das premissas especificadas por Fruchterman e Reingold (1991) e com isso é possível perceber a diferença da disposição desses vértices entres as visualizações geradas. Fazendo uma análise da Figura 6.3 quadro 1 gerado pelo DrawNet com a imagem do quadro 2 gerada pelo Pajek percebemos que a disposição dos vértices conectados, segue similar ao apresentado como resultado da visualização do software DrawNet, nos dois caso os vértices conectados são dispostos próximos uns aos outros, e os vértices desconectados na borda da visualização.

Se analisarmos as imagens obtidas pelos softwares e compararmos com o quadro 4 da Figura 6.4 obtida por Fruchterman e Reingold (1991) em seu artigo, nota-se que todas as visualizações seguem o mesmo padrão, vértices conectados próximos uns dos outros e tamanho de aresta similar.

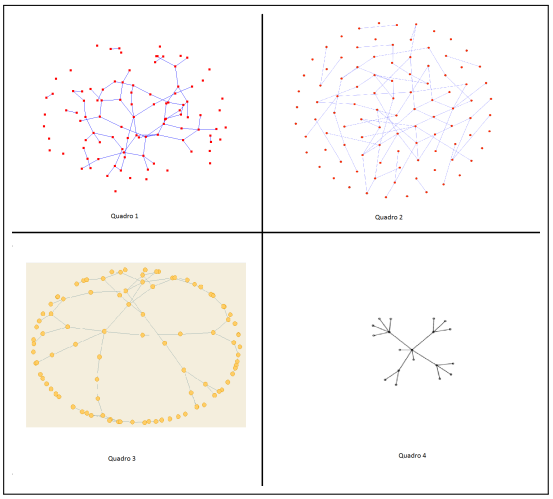

Figura 6.4: Figura comparando a imagem gerada pela aplicação DrawNet, Gephi, Pajek e imagem retirada do artigo *Graph Drawing by Force-directed Placement* Fonte: (Fruchterman; Reingold, 1991)

Como essa primeira comparação deixa margem para interpretações diferentes das que foram tiradas nesta pesquisa, faremos uma comparação com grafos completos no software DrawNet, Gephi, Pajek e pegaremos as imagens de referência do artigo de Fruchterman e Reingold (1991).

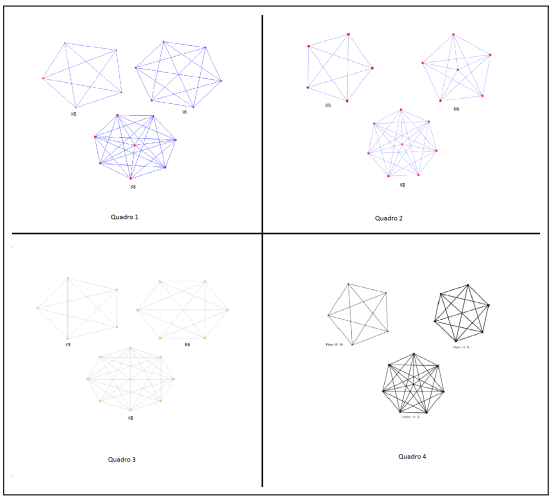

Figura 6.5: Figura comparando a imagem gerada pela aplicação DrawNet, Gephi, Pajek e imagem retirada do artigo *Graph Drawing by Force-directed Placement*. Grafos completos K5, K6 e K8. Fonte: Próprio autor

Analisando a Figura 6.5 é possível perceber a semelhança na visualização do grafo completo gerado por todos os softwares (quadro 1, 2 e 3) comparando com o modelo de referência do artigo (quadro 4). Com isso, podemos afirmar que o algoritmo implementado no DrawNet atende todas as características especificadas no artigo.

6.3 Algoritmo Circular

O algoritmo circular pode ser implementado de diversas maneiras, com grandes variações de projeção, entre elas estão o layout clássico e variações como a Radial. Para ser implementado no DrawNet, o algoritmo escolhido foi o clássico onde apenas uma única circunferência é gerada mais próxima possível as bordas da janela de projeção.

Para realizar a comparação do resultado gerado pelo software DrawNet, foi utilizado uma rede com 100 vértices, 73 arestas e um grau máximo de 5, a mesma rede usada para comparação do algoritmo Force Direct Fruchterman and Reingold 6.2. Nesse caso, o algoritmo implementado no DrawNet será comparado com o artigo de Six e Tollis (2013) intitulado *Circular Drawing Algorithms* e através da geração através do Pajek que possui implementado esse algoritmo. Nesse caso o Gephi não possui implementado o algoritmo.

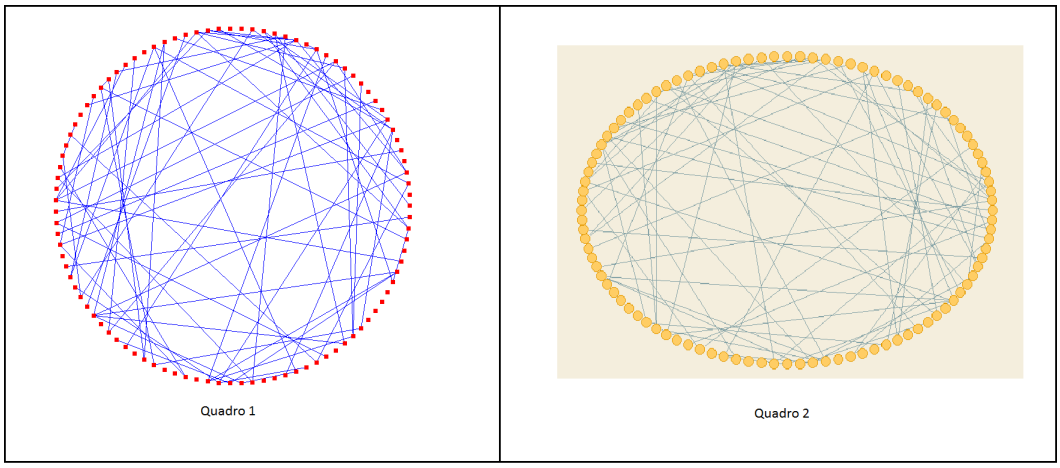

Figura 6.6: Figura comparando a imagem gerada pela aplicação DrawNet com a imagem gerada pelo Pajek. Rede com 100 vértices. Fonte: Próprio autor

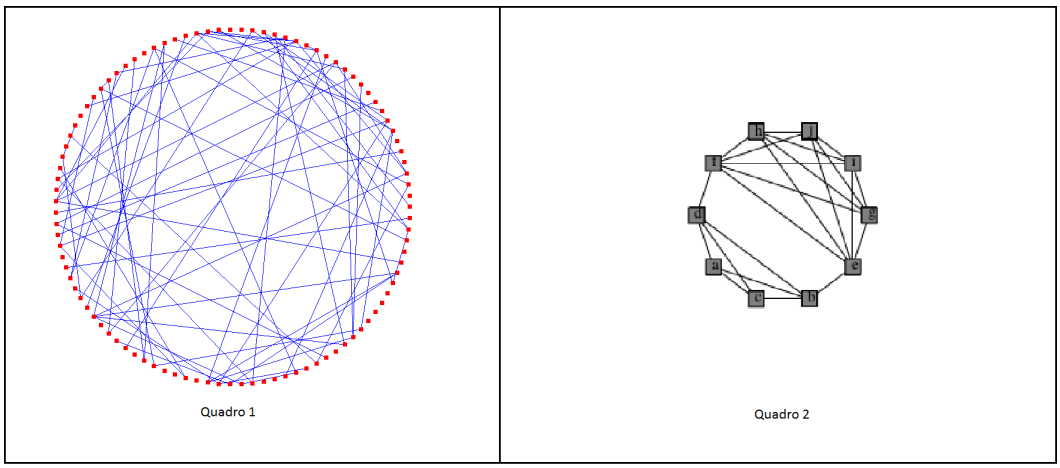

Figura 6.7: Figura comparando a imagem gerada pela aplicação DrawNet com a imagem retirado do artigo *Circular Drawing Algorithms* Fonte: Próprio autor

Como é possível observar na Figura 6.6 nos quadros 1 e 2 que foram geradas respectivamente pelo software DrawNet desenvolvido em GuaráScript e Pajek, vemos uma semelhança muito alta entre a disposição dos vértices sendo que a diferença é que no Pajek os vértices no eixo Y não estão tão próximos a borda quanto estão na visualização gerada pelo DrawNet. E se compararmos as imagens obtidas dos softwares com a imagem de referência no quadro 2 da Figura 6.7, percebemos que nas duas visualizações as características da geração seguem o padrão definido no artigo de Six e Tollis (2013). Com isso é possível afirmar que o algoritmo implementado pelo DrawNet atende todas as características do apresentadas na literatura.

6.4 Algoritmo K-core

Para realizar a comparação do resultado gerado pelo software DrawNet, foi utilizado uma rede com 100 vértices, 73 arestas e um grau máximo de 5, a mesma rede usada para comparação do algoritmo Force Direct Fruchterman and Reingold 6.2 e no algoritmo Circular Layout 6.3 e uma rede com 500 vértices e 960 arestas e grau máximo 11 que será usada com intuito de comparar com uma figura retirada do artigo de Alvarez-Hamelin et al. (2005). Como esse algoritmo não foi implementado nem pelo Gephi e nem pelo Pajek, o mesmo será comparado com a literatura. O artigo em questão é o de Alvarez-Hamelin et al. (2005) intitulado *K-core decomposition: a tool for the visualization of large scale networks*. Nele o autor realiza a exibição de redes aplicando o algoritmo tamanhos diversos de redes. Como os autores não deixam claro o tamanho exato da rede, quantidade de vértices e arestas, a comparação será por similaridade.

Na primeira geração, foi utilizada uma rede com 100 vértices para geração da visualização. A rede não está totalmente conectada. Circunferências foram inseridas no quadro 1 da Figura 6.8 para facilitar a comparação com o quadro 2 da mesma figura.

Como é possível observar, apesar da geração da visualização serem realizadas com redes diferentes, é possível verificar que o comportamento das duas visualizações são bem similares. Os vértices com menor grau na circunferência mais externa e os vértices com maior grau, na circunferência mais interna. Com isso, é possível afirmar que o algoritmo implementado pelo DrawNet atende com similaridade, a proposta do algoritmo para esse caso.

Agora faremos o comparativo utilizando a rede com 500 vértices. O quadro 1 da Figura 6.9 representa a visualização gerada pelo software DrawNet e a imagem do quadro 2 foi retirada do artigo. É possível observar também a similaridade do funcionamento do algoritmo implementado com a literatura. Apesar da imagem da literatura haver mais vértices no centro da visualização, é possível notar o mesmo comportamento com a rede

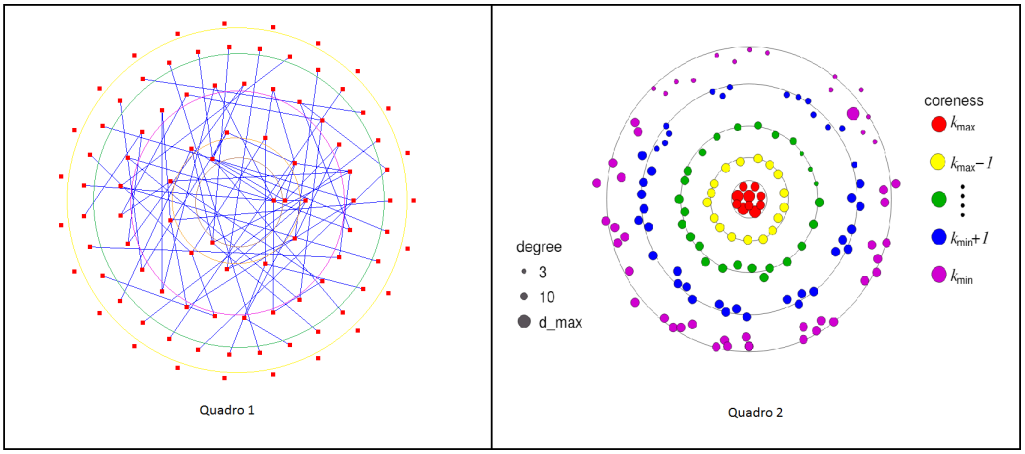

Figura 6.8: Figura comparando a imagem gerada pela aplicação DrawNet do algoritmo K-core com os círculos coloridos adicionado posteriormente para comparação a imagem retirada do artigo *K-core decomposition: a tool for the visualization of large scale networks*. Rede com 100 vértices. Fonte: Próprio autor

com 500 vértices, onde a maior parte dos vértices possuem um grau elevado na rede. Com isso, é possível afirmar que o algoritmo implementado pelo DrawNet atende com similaridade, a proposta do algoritmo para esse caso.

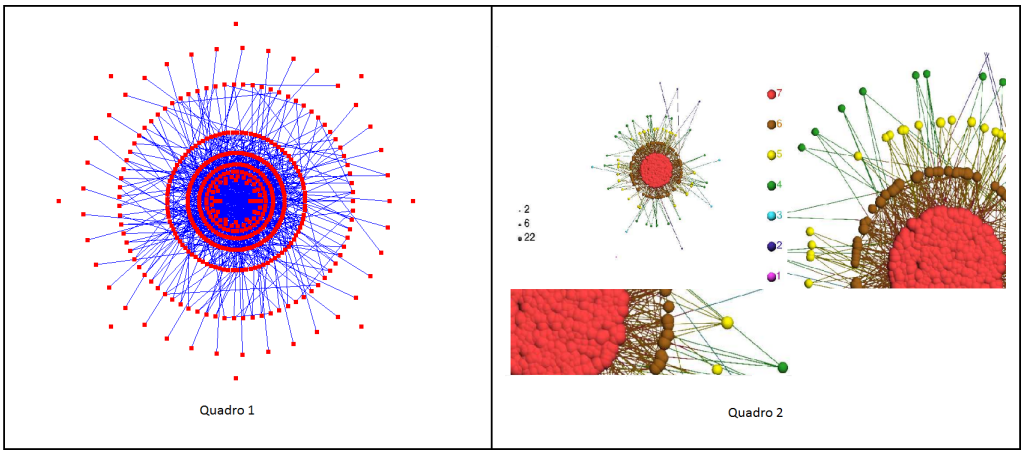

Figura 6.9: Figura comparando a imagem gerada pela aplicação DrawNet do algoritmo K-core com os círculos coloridos adicionado posteriormente para comparação a imagem retirada do artigo *K-core decomposition: a tool for the visualization of large scale networks*. Rede com 500 vértices. Fonte: Próprio autor

Esse capítulo apresentou em formato comparativo a visualização de redes utilizando os algoritmos Force-Direct, Circular Layout e K-core entre os softwares DrawNet, Pajek e Gephi com os devidos artigos, com intuito de validar o resultado obtido pelo software DrawNet desenvolvido a partir da modelo computacional especificado nesta pesquisa. O próximo capítulo irá apresentar as considerações finais da pesquisa.

Capítulo Sete

Considerações finais

A análise comportamental de uma rede ou a obtenção de características dos vértices, geralmente é realizada através de cálculos matemáticos que a partir de propriedades estatísticas como o grau ou coeficiente de aglomeração que classificam a rede em tipos. Ao realizar a classificação das redes, comportamentos padrões podem ser observados para inferir características, e com isso realizar comparações e análises sobre sua dinâmica de interações.

Apesar dos recursos estatísticos disponíveis para análise das redes, o advento da visualização de redes pode revelar comportamentos, até então, mais difíceis de serem compreendidos apenas pelas suas características matemáticas. A visualização das redes em ambiente bidimensional e tridimensional traz uma nova perspectiva de análise de rede que poderá ajudar em pesquisas sobre seu comportamento. A diversidade de algoritmos de visualização de redes existentes demonstra que existe um esforço em utilizar a representação gráfica das redes, para obter vantagens na obtenção de informações. Com isso, a importância do estudo e da aplicação dos conceitos de visualização de informação em redes sociais e complexas na forma de software aumenta, e a necessidade de cada vez mais ferramentas que auxiliem na extração de informação da rede através da visualização se faz necessária.

Com isso foi apresentado um modelo computacional contendo a macro arquitetura, estrutura de dados, linguagem de programação e API que auxiliasse no desenvolvimento de um software de visualização de rede capaz de plotar redes sociais e complexas em ambiente bidimensional.Para validação desse modelo computacional apresentado, o desenvolvimento de um software que pudesse agregar uma outra perspectiva na análise visual da rede, provendo recursos de computação gráfica que podem ser evoluídos para disponibilizar uma melhor interação com a rede o suporte a multi plataforma e de código aberto foi realizado e através das análises comparativas realizadas no capítulo 6 percebemos que esses objetivos foram alcançados.

Além do inerente problema em desenvolver uma solução de software de visualização de redes, a escolha de uma linguagem de programação traz consigo pontos positivos e negativos independente de qual linguagem seja escolhida. No caso desta pesquisa a escolha da linguagem GuaráScript trouxe grandes vantagens no desenvolvimento do software como a integração nativa com a API do OpenGL sem a necessidade de adição de plugins ou configuração adicional, a gama de recursos matemáticos disponíveis na linguagem já que a mesma foi desenvolvida com cunho científico e uma biblioteca de recursos para traba-

lhar com redes complexas já disponível no software SCNTools desenvolvido também em GuaráScript. Como contraponto as vantagens obtidas, alguns problemas foram percebidos ao longo do desenvolvimento da aplicação DrawNet como o fato da mesma não ser multi-thread o que impossibilitou o desenvolvimento de algoritmos como o force atlas 2 que tem como princípio o uso de threads para otimizar a convergência da rede ao seu estado de equilíbrio.

7.1 Contribuições

As contribuições que esta pesquisa traz são (1) um modelo computacional capaz de realizar a execução dos algoritmos bem conhecidos no campo de pesquisa de visualização de redes sociais e complexas, atendendo todos os requisitos e características especificadas na literatura após comparações realizadas com os artigos e os softwares já disponíveis, (2) um conjunto de diversas funções implementadas para geração e manipulação básica da rede possibilitando assim a evolução do software DrawNet desenvolvido em GuaráScript agregando novas funcionalidades, (3) um módulo de serviços com recursos já testados e funcionais como integração com mouse e teclado, re-posicionamento do vértices através do arrasto do mesmo na tela, entre outros, (4) uma biblioteca de algoritmos já implementados em linguagem GuaráScript para visualização de redes.

Outra contribuição importante é a disponibilização do código fonte em um repositório público no Github no endereço https://github.com/rodrigopvb/mestrado.networkvisualization para que a comunidade científica possa realizar pesquisas com o software desenvolvido em GuaráScript bem como que possa contribuir com a inserção de novos recursos.

7.2 Atividades Futuras de Pesquisa

Como atividades futuras da pesquisa podemos citar:

- A adoção de uma estrutura de dados de vetores bi-dimensionais para otimizar o uso da memória;
- Estudos sobre otimização dos algoritmos implementados;
- Implementação da visualização de redes em ambiente tri-dimensional;

Apêndice A

Documentos

A.1 Instruções de Instalação

Realizar download, compilar e instalar a linguagem GuaraScript.

Link para download: https://github.com/guarascript/guash

Após a extração dos arquivos em uma pasta, abra o terminal e digite os comando a seguir em ambiente Linux:

$make
$make install
$make install_guash
$make install_glwmguash

Após a instalação do GuaráScript, realizar download do repositório DrawNet no link https://github.com/rodrigopvb/mestrado.networkvisualization.git e extrair o conteúdo em uma pasta.

Abra o terminal e digite os comando a seguir

$ glwmguash drawnet.gua -i [nomeArquivo] –[algoritmo]

Tabela A.1: Algoritmos implementados no DrawNet

Nome	**Descrição**
fruchterman	Algoritmo de Força Direta por Fruchterman e Reignold
circular	Algoritmo Circular
kcore	Algoritmo K-core
random	Algoritmo de posicionamento aleatório
expansion	Algoritmo de expansão de rede
retraction	Algoritmo de retração de rede

Arquivos de exemplos de rede para teste:

Network20Vertex.net
Network100Vertex.net
Network500Vertex.net

Funcionalidade:

Movimentação de Vértice:
Pressionar o botão direito do mouse e mover o vértice para a posição desejada.

Criar arestas dinamicamente (Tempo de Execução):
Pressionar o botão esquerdo do mouse no vértice inicial e pressionar novamente o botão esquerdo do mouse no vértice de destino.

Pausar algoritmo baseado em força direta:
Pressionar barra de espaço.

Expandir a rede:
Pressionar a tecla "e"

Contrair a rede:
Pressionar a tecla "c"

Mostrar o nome do vértice:
Pressionar a tecla "l"

Informação sobre a rede no terminal:
Algoritmo Escolhido
Nome do Arquivo
Número de Vértices

Algoritmo Circular

GuaSh IDE 2016-11-16

```
1  source("DrawNetController.gua")
2
3  startAngle = 0.0;
4  endAngle = PI * 2;
5  radiusXaxis = ((getWidthArea() / 2)* 0.1)*10;
6  radiusYaxis = ((getHeightArea() / 2) * 0.1)*10;
7  adjustMaxBorder = 5.0;
8
9  posX = {0};
10 posY = {0};
11
12 function calculatePosition(){
13     amountVertices = getLengthXv();
14
15     offset = ((endAngle - startAngle)/(amountVertices - 1));
16     if($radiusXaxis < $radiusYaxis){
17         smallestRadius = $radiusXaxis;
18     }else{
19         smallestRadius = $radiusYaxis;
20     }
21
22     println("Quantidade de Vértices: "+ amountVertices);
23     println("Valor do offset em radiano: "+ offset);
24     println("Valor do menor raio: " + smallestRadius);
25
26     for (i = 0; i < getLengthXv(); i = i + 1){
27         angle = (i * offset);
28
29         $posX[i] = convertValueNormalized(($radiusXaxis + ((smallestRadius -
   adjustMaxBorder) *cos($startAngle + angle))) / getWidthArea());
30         $posY[i] = convertValueNormalized(($radiusYaxis + ((smallestRadius -
   adjustMaxBorder) *sin($startAngle + angle))) / getHeightArea());
31     }
32     setPositionVerticesAxisX($posX);
33     setPositionVerticesAxisY($posY);
34     printf(arrayToString($posX)+"\n\n")
35     printf(arrayToString($posY)+"\n\n")
36 }
```

14:55:23 CircularLayout.gua Page 1/1

B.1 Linha de Comando

$ glwmguash drawnet.gua -i [nomeArquivo] –circular

Algoritmo Expansion

GuaSh IDE 2016-11-16

```
1  posXv = {0};
2  posYv = {0};
3
4  function initExpansion(){
5      amountVertices =  getLengthXv();
6      println("Quantidade de Vértices: "+ amountVertices);
7      for (i = 0; i < amountVertices; i = i + 1){
8      condition = 0;
9      randomValueX =0;
10     randomValueY =0;
11         while(condition == 0){
12             randomValueX = randomPosition();
13             if((randomValueX < 0.650) && (randomValueX > 0.350)){
14                 condition = 1;
15             }
16         }
17         condition = 0;
18         while(condition == 0){
19             randomValueY = randomPosition();
20             if((randomValueY < 0.650) && (randomValueY > 0.350)){
21                 condition = 1;
22             }
23         }
24
25         $posXv[i] = randomValueX;
26         $posYv[i] = randomValueY;
27     }
28 setPositionVerticesAxisX($posXv);
29 setPositionVerticesAxisY($posYv);
30
31 }
32
33 function expansion(){
34     $posXv = getPositionsVertexAxisX();
35     $posYv = getPositionsVertexAxisY();
36     for(i = 0; i< length($posXv); i=i+1){
37         if(($posXv[i] > 0.5) && ($posYv[i]<0.5)){
38             parcialValueX = $posXv[i] * 1.001;
39             if((parcialValueX < 0.995) && (parcialValueX > 0.005)){
40                 $posXv[i] = parcialValueX;
41             }elseif(parcialValueX < 0.005){
42                 $posXv[i] = 0.005;
43             }elseif(parcialValueX > 0.995){
44                 $posXv[i] = 0.995;
45             }
46             parcialValueY = $posYv[i] / 1.001;
47             if((parcialValueY < 0.995) && (parcialValueY > 0.005)){
48                 $posYv[i] = parcialValueY;
49             }elseif(parcialValueY < 0.005){
50                 $posYv[i] = 0.005;
51             }elseif(parcialValueY > 0.995){
52                 $posYv[i] = 0.995;
53             }
54         }elseif(($posXv[i] > 0.5) && ($posYv[i] > 0.5)){
55             parcialValueX = $posXv[i] * 1.001;
56             if((parcialValueX < 0.995) && (parcialValueX > 0.005)){
57                 $posXv[i] = parcialValueX;
```

```
58              }elseif(parcialValueX < 0.005){
59                  $posXv[i] = 0.005;
60              }elseif(parcialValueX > 0.995){
61                  $posXv[i] = 0.995;
62              }
63              parcialValueY = $posYv[i] * 1.001;
64              if((parcialValueY < 0.995) && (parcialValueY > 0.005)){
65                  $posYv[i] = parcialValueY;
66              }elseif(parcialValueY < 0.005){
67                  $posYv[i] = 0.005;
68              }elseif(parcialValueY > 0.995){
69                  $posYv[i] = 0.995;
70              }
71          }elseif(($posXv[i] < 0.5) && ($posYv[i] > 0.5)){
72              parcialValueX = $posXv[i] / 1.001;
73              if((parcialValueX < 0.995) && (parcialValueX > 0.005)){
74                  $posXv[i] = parcialValueX;
75              }elseif(parcialValueX < 0.005){
76                  $posXv[i] = 0.005;
77              }elseif(parcialValueX > 0.995){
78                  $posXv[i] = 0.995;
79              }
80              parcialValueY = $posYv[i] * 1.001;
81              if((parcialValueY < 0.995) && (parcialValueY > 0.005)){
82                  $posYv[i] = parcialValueY;
83              }elseif(parcialValueY < 0.005){
84                  $posYv[i] = 0.005;
85              }elseif(parcialValueY > 0.995){
86                  $posYv[i] = 0.995;
87              }
88          }elseif(($posXv[i] < 0.5) && ($posYv[i] < 0.5)){
89              parcialValueX = $posXv[i] / 1.001;
90              if((parcialValueX < 0.995) && (parcialValueX > 0.005)){
91                  $posXv[i] = parcialValueX;
92              }elseif(parcialValueX < 0.005){
93                  $posXv[i] = 0.005;
94              }elseif(parcialValueX > 0.995){
95                  $posXv[i] = 0.995;
96              }
97              parcialValueY = $posYv[i] / 1.001;
98              if((parcialValueY < 0.995) && (parcialValueY > 0.005)){
99                  $posYv[i] = parcialValueY;
100             }elseif(parcialValueY < 0.005){
101                 $posYv[i] = 0.005;
102             }elseif(parcialValueY > 0.995){
103                 $posYv[i] = 0.995;
104             }
105         }
106         parcialValueX = 0;
107         parcialValueY = 0;
108     }
109 setPositionVerticesAxisX($posXv);
110 setPositionVerticesAxisY($posYv);
111 }
```

C.1 Linha de Comando

$ glwmguash drawnet.gua -i [nomeArquivo] –expansion

Algoritmo Fruchterman

GuaSh IDE 2016-11-16

```
1  AREA_MULTIPLICATOR    = 0.001;
2  speed                 = 1.0;
3  amountVertex          = 0;
4  k                     = 0;
5  gravity               = 10;
6  speed_divisor         = 2.0;
7  inverse_temperature   = 0;
8  temperature_increment = 0.2;
9
10
11 posXv = {0};
12 posYv = {0};
13
14 forceDirectionY = {0};
15 forceDirectionX = {0};
16
17 edgesXe = {0};
18
19 function getFruchtermanReingoldAttractionForce(distance){
20     quadDistance = distance * distance;
21     result = quadDistance / $k;
22     return(result);
23 }
24
25 function getFruchtermanReingoldRepulsionForce(distance){
26     quadK = $k * $k;
27     result = quadK / distance;
28     return(result);
29 }
30
31 function initFruchtermanReingoldForceVector(){
32     for (i = 0; i < $amountVertex; i = i + 1){
33         $forceDirectionX[i] = 0;
34         $forceDirectionY[i] = 0;
35     }
36 }
37
38 function initialPositions(){
39
40     for (i = 0; i < getLengthXv(); i = i + 1){
41         condition    = 0;
42         randomValueX = 0;
43         randomValueY = 0;
44             while(condition == 0){
45                 randomValueX = randomPosition();
46                 if((randomValueX < 0.550) && (randomValueX > 0.450)){
47                     condition = 1;
48                 }
49             }
50             condition = 0;
51             while(condition == 0){
52                 randomValueY = randomPosition();
53                 if((randomValueY < 0.550) && (randomValueY > 0.450)){
54                     condition = 1;
55                 }
56             }
57             $posXv[i] = convertValueNormalized(randomValueX);
```

```
58              $posYv[i] = convertValueNormalized(randomValueY);
59      }
60
61      println("Valores do array posXv: " + arrayToString($posXv));
62      println("Valores do array posYv: " + arrayToString($posYv));
63      setPositionVerticesAxisX($posXv);
64      setPositionVerticesAxisY($posYv);
65  }
66
67  function updateFruchtermanReingoldPosition(){
68      #updatePosition
69      for(i = 0; i < amountVertex; i = i + 1){
70          displacementX = $forceDirectionX[i];
71          displacementY = $forceDirectionY[i];
72          d = sqrt(displacementX * displacementX + displacementY * displacementY);
73
74          if(d > 0){
75              maxDisplace = getFruchtermanReingoldMaxDisplace();
76              limitedDist =  maxDisplace * ($speed / $speed_divisor);
77              if(limitedDist < d){
78                  $posXv[i] = convertValueNormalized($posXv[i] + displacementX / d *
    limitedDist);
79                  $posYv[i] = convertValueNormalized($posYv[i] + displacementY / d *
    limitedDist);
80
81              }else{
82                  limitedDist = d;
83                  $posXv[i] = convertValueNormalized($posXv[i] + displacementX / d *
    limitedDist);
84                  $posYv[i] = convertValueNormalized($posYv[i] + displacementY / d *
    limitedDist);
85              }
86          }
87      }
88      setPositionVerticesAxisX($posXv);
89      setPositionVerticesAxisY($posYv);
90  }
91
92  function calculateFruchtermanReingoldRepulsionForce(){
93      #repulsionForce
94      for (i = 0; i < $amountVertex; i = i + 1){
95          dxCurrentVertex = $posXv[i];
96          dyCurrentVertex = $posYv[i];
97          for (j = 0; j < $amountVertex; j = j + 1){
98              if(i != j){
99                  displacementDifferenceX = dxCurrentVertex - $posXv[j];
100                 displacementDifferenceY = dyCurrentVertex - $posYv[j];
101                 value = displacementDifferenceX * displacementDifferenceX +
    displacementDifferenceY * displacementDifferenceY;
102
103                 if(value!=0) {
104                     d = sqrt(value);
105                 }
106                 else {
107                     d = 0.001;
108                 }
109
```

```
110                 if(d > 0){
111                     repulseF = getFruchtermanReingoldRepulsionForce(d);
112                     $forceDirectionX[i] = $forceDirectionX[i] + (displacementDifferenceX/
    d) * repulseF;
113                     $forceDirectionY[i] = $forceDirectionY[i] + (displacementDifferenceY/
    d) * repulseF;
114                 }
115             }
116         }
117     }
118 }
119
120 function calculateFruchtermanReingoldAttractionForce(){
121     #attractionForce
122     for (i = 0; i < getLengthXe(); i = i + 2){
123         displacementDifferenceX = $posXv[$edgesXe[i]] - $posXv[$edgesXe[i+1]];
124         displacementDifferenceY = $posYv[$edgesXe[i]] - $posYv[$edgesXe[i+1]];
125
126         value = displacementDifferenceX * displacementDifferenceX +
    displacementDifferenceY * displacementDifferenceY;
127
128         if(value != 0) {
129             d = sqrt(value);
130         }
131         else {
132             d = 0.001;
133         }
134
135         attractiveF = getFruchtermanReingoldAttractionForce(d);
136         if(d > 0){
137             $forceDirectionX[$edgesXe[i]]   = $forceDirectionX[$edgesXe[i]]   -
    (displacementDifferenceX/d) * attractiveF;
138             $forceDirectionY[$edgesXe[i]]   = $forceDirectionY[$edgesXe[i]]   -
    (displacementDifferenceY/d) * attractiveF;
139             $forceDirectionX[$edgesXe[i+1]] = $forceDirectionX[$edgesXe[i+1]] +
    (displacementDifferenceX/d) * attractiveF;
140             $forceDirectionY[$edgesXe[i+1]] = $forceDirectionY[$edgesXe[i+1]] +
    (displacementDifferenceY/d) * attractiveF;
141         }
142     }
143 }
144
145 function calculateFruchtermanReingoldGravity(){
146     #gravity
147     for(i = 0; i < $amountVertex; i = i + 1){
148         d = sqrt($posXv[i] * $posXv[i] + $posYv[i] * $posYv[i]);
149         gf = 0.01 * $k * $gravity * d;
150         $forceDirectionX[i] = $forceDirectionX[i] - gf * $posXv[i] / d;
151         $forceDirectionY[i] = $forceDirectionY[i] - gf * $posYv[i] / d;
152     }
153
154 }
155 function calculateFruchtermanReingoldSpeed(){
156     #speed
157     for(i = 0; i < $amountVertex; i = i + 1){
158         $forceDirectionX[i] = $forceDirectionX[i] * $speed / $speed_divisor;
159         $forceDirectionY[i] = $forceDirectionY[i] * $speed / $speed_divisor;
```

```
160     }
161
162 }
163 function getFruchtermanReingoldArea (){
164     area = getWidthArea() * getHeightArea()/100.0;
165     return(area);
166 }
167
168 function getFruchtermanReingoldMaxDisplace(){
169     area = getFruchtermanReingoldArea();
170     maxDisplace =  sqrt($AREA_MULTIPLICATOR *area)/($inverse_temperature + 25.0);
171     $inverse_temperature = $inverse_temperature + $temperature_increment;
172     return(maxDisplace);
173 }
174
175 function getFruchtermanReingoldK(){
176     area           = getFruchtermanReingoldArea();
177     l_amountVertex = getLengthXv();
178     k              = sqrt($AREA_MULTIPLICATOR *area)/(1.0 + l_amountVertex);
179     return(k);
180 }
181
182 function calculateForces(){
183     calculateFruchtermanReingoldRepulsionForce();
184     calculateFruchtermanReingoldAttractionForce();
185     calculateFruchtermanReingoldGravity();
186     calculateFruchtermanReingoldSpeed();
187     updateFruchtermanReingoldPosition();
188 }
189
190 function initFruchterman(){
191     $edgesXe      = getPositionsEdgesAxisX();
192     $amountVertex = getLengthXv();
193     $k = getFruchtermanReingoldK();
194
195     initialPositions();
196     initFruchtermanReingoldForceVector();
197     calculateForces();
198 }
```

D.1 Linha de Comando

$ glwmguash drawnet.gua -i [nomeArquivo] –fruchterman

Algoritmo K-core


```
1  source("DrawNetController.gua")
2
3  startAngle = 0.0;
4  endAngle = PI * 2;
5  radiusXaxisMax = ((getWidthArea() / 2)* 0.1)*10;
6  radiusYaxisMax = ((getHeightArea() / 2) * 0.1)*10;
7  adjustMaxBorder = 5.0;
8  degreeMax = 0;
9
10 posX = {0};
11 posY = {0};
12
13
14 function calculateDegree(vertex){
15     degree = 0;
16     xe = getPositionsEdgesAxisX();
17     for (i = 0; i < length(xe); i = i + 2){
18         if(xe[i] == vertex){
19             degree = degree + 1;
20         }elseif(xe[i+1] == vertex){
21             degree = degree + 1;
22         }
23     }
24     if($degreeMax < degree){
25         $degreeMax = degree;
26     }
27     return(degree);
28 }
29
30 function getSmallestRadius(){
31     if(currentDegree == 0){
32         currentRadiusXaxis = $radiusXaxisMax / (1);
33         currentRadiusYaxis = $radiusYaxisMax / (1);
34         if(currentRadiusXaxis < currentRadiusYaxis){
35             smallestRadius = currentRadiusXaxis;
36         }else{
37             smallestRadius = currentRadiusYaxis;
38         }
39     }elseif(currentDegree == 1){
40         currentRadiusXaxis = $radiusXaxisMax / (currentDegree + 0.150);
41         currentRadiusYaxis = $radiusYaxisMax / (currentDegree + 0.150);
42         if(currentRadiusXaxis < currentRadiusYaxis){
43             smallestRadius = currentRadiusXaxis;
44         }else{
45             smallestRadius = currentRadiusYaxis;
46         }
47     }else{
48         currentRadiusXaxis = $radiusXaxisMax / (currentDegree - 0.500);
49         currentRadiusYaxis = $radiusYaxisMax / (currentDegree - 0.500);
50         if(currentRadiusXaxis < currentRadiusYaxis){
51             smallestRadius = currentRadiusXaxis;
52         }else{
53             smallestRadius = currentRadiusYaxis;
54         }
55
56     }
57     return(smallestRadius)
```

```
58  }
59
60
61  function calculeAmountDegreeVertex(arrayDegreVertex, degree){
62      amountVertex = 0;
63      for(i = 0; i < length(arrayDegreVertex); i=i+1){
64          if(arrayDegreVertex[i] == degree){
65              amountVertex = amountVertex + 1;
66          }
67      }
68      return(amountVertex);
69  }
70
71  function calculatePositionKCore(){
72      amountVertices = getLengthXv();
73
74      for(i = 0; i < amountVertices; i=i+1){
75          degreeVertex[i] = calculateDegree(i);
76      }
77
78      println("Quantidade de Vértices: "+ amountVertices);
79      println("Grau máximo: "+ $degreeMax+"\n");
80
81      for(i = 0; i <= $degreeMax; i=i+1){
82          currentDegree = i;
83          amountDegreeVertex = calculeAmountDegreeVertex(degreeVertex, currentDegree);
84
85          if(amountDegreeVertex == 1){
86              offset = ((endAngle - startAngle)/(amountDegreeVertex));
87          }elseif(amountDegreeVertex > 1){
88              offset = ((endAngle - startAngle)/(amountDegreeVertex - 1));
89          }else{
90              offset = ((endAngle - startAngle)/(1));
91          }
92          diferenceAmountDegreeVertex = 0;
93
94          println("Grau atual: " + currentDegree);
95          println("Quantidade de vertices com grau "+ currentDegree + " Total: "+
    amountDegreeVertex);
96          println("Valor de offset em radiano: "+ offset+ "\n");
97
98      for (j = 0; j < getLengthXv(); j = j + 1){
99          smallestRadius = getSmallestRadius();
100
101             if((degreeVertex[j] == currentDegree)){
102                 diferenceAmountDegreeVertex = diferenceAmountDegreeVertex + 1;
103                 angle = (diferenceAmountDegreeVertex * offset);
104                 $posX[j] = ($radiusXaxisMax + ((smallestRadius - adjustMaxBorder) *cos
    ($startAngle + angle))) / getWidthArea();
105                 $posY[j] = ($radiusYaxisMax + ((smallestRadius - adjustMaxBorder) *sin
    ($startAngle + angle))) / getHeightArea();
106
107             }
108         }
109     }
110     setPositionVerticesAxisX($posX);
111     setPositionVerticesAxisY($posY);
```

```
112 }
113
```

E.1 Linha de Comando

$ glwmguash drawnet.gua -i [nomeArquivo] –kcore

Algoritmo Random

GuaSh IDE 2016-11-16

```
1  source("DrawNetController.gua")
2
3  posXv = {0};
4  posYv = {0};
5
6  function initRandom(){
7      amountVertices =  getLengthXv();
8      println("Quantidade de Vértices: "+ amountVertices);
9      for (i = 0; i < amountVertices; i = i + 1){
10     condition = 0;
11     randomValueX =0;
12     randomValueY =0;
13         while(condition == 0){
14             randomValueX = randomPosition();
15             if((randomValueX < 0.850) && (randomValueX > 0.150)){
16                 condition = 1;
17             }
18         }
19         condition = 0;
20         while(condition == 0){
21             randomValueY = randomPosition();
22             if((randomValueY < 0.850) && (randomValueY > 0.150)){
23                 condition = 1;
24             }
25         }
26         randomValueNormalizedX = convertValueNormalized(randomValueX);
27         randomValueNormalizedY = convertValueNormalized(randomValueY);
28         $posXv[i] = randomValueNormalizedX;
29         $posYv[i] = randomValueNormalizedY;
30     }
31 setPositionVerticesAxisX($posXv);
32 setPositionVerticesAxisY($posYv);
33 }
```

14:57:35 RandomLayout.gua Page 1/1

F.1 Linha de Comando

$ glwmguash drawnet.gua -i [nomeArquivo] –random

Algoritmo Retraction

GuaSh IDE 2016-11-16

```
1  posXv = {0};
2  posYv = {0};
3
4  function initRetraction(){
5      amountVertices =  getLengthXv();
6      println("Quantidade de Vértices: "+ amountVertices);
7      for (i = 0; i < getLengthXv(); i = i + 1){
8      condition = 0;
9      randomValueX =0;
10     randomValueY =0;
11         while(condition == 0){
12             randomValueX = randomPosition();
13             if((randomValueX < 0.850) && (randomValueX > 0.150)){
14                 condition = 1;
15             }
16         }
17         condition = 0;
18         while(condition == 0){
19             randomValueY = randomPosition();
20             if((randomValueY < 0.850) && (randomValueY > 0.150)){
21                 condition = 1;
22             }
23         }
24
25         $posXv[i] = randomValueX;
26         $posYv[i] = randomValueY;
27     }
28 setPositionVerticesAxisX($posXv);
29 setPositionVerticesAxisY($posYv);
30
31 }
32
33 function retraction(){
34     $posXv = getPositionsVertexAxisX();
35     $posYv = getPositionsVertexAxisY();
36     for(i = 0; i< length($posXv); i=i+1){
37         if(($posXv[i] > 0.5) && ($posYv[i]<0.5)){
38             parcialValueX = $posXv[i] / 1.001;
39             if((parcialValueX < 0.995) && (parcialValueX > 0.005)){
40                 $posXv[i] = parcialValueX;
41             }elseif(parcialValueX < 0.005){
42                 $posXv[i] = 0.005;
43             }elseif(parcialValueX > 0.995){
44                 $posXv[i] = 0.995;
45             }
46             parcialValueY = $posYv[i] * 1.001;
47             if((parcialValueY < 0.995) && (parcialValueY > 0.005)){
48                 $posYv[i] = parcialValueY;
49             }elseif(parcialValueY < 0.005){
50                 $posYv[i] = 0.005;
51             }elseif(parcialValueY > 0.995){
52                 $posYv[i] = 0.995;
53             }
54         }elseif(($posXv[i] > 0.5) && ($posYv[i] > 0.5)){
55             parcialValueX = $posXv[i] / 1.001;
56             if((parcialValueX < 0.995) && (parcialValueX > 0.005)){
57                 $posXv[i] = parcialValueX;
```

```
58              }elseif(parcialValueX < 0.005){
59                  $posXv[i] = 0.005;
60              }elseif(parcialValueX > 0.995){
61                  $posXv[i] = 0.995;
62              }
63              parcialValueY = $posYv[i] / 1.001;
64              if((parcialValueY < 0.995) && (parcialValueY > 0.005)){
65                  $posYv[i] = parcialValueY;
66              }elseif(parcialValueY < 0.005){
67                  $posYv[i] = 0.005;
68              }elseif(parcialValueY > 0.995){
69                  $posYv[i] = 0.995;
70              }
71          }elseif(($posXv[i] < 0.5) && ($posYv[i] > 0.5)){
72              parcialValueX = $posXv[i] * 1.001;
73              if((parcialValueX < 0.995) && (parcialValueX > 0.005)){
74                  $posXv[i] = parcialValueX;
75              }elseif(parcialValueX < 0.005){
76                  $posXv[i] = 0.005;
77              }elseif(parcialValueX > 0.995){
78                  $posXv[i] = 0.995;
79              }
80              parcialValueY = $posYv[i] / 1.001;
81              if((parcialValueY < 0.995) && (parcialValueY > 0.005)){
82                  $posYv[i] = parcialValueY;
83              }elseif(parcialValueY < 0.005){
84                  $posYv[i] = 0.005;
85              }elseif(parcialValueY > 0.995){
86                  $posYv[i] = 0.995;
87              }
88          }elseif(($posXv[i] < 0.5) && ($posYv[i] < 0.5)){
89              parcialValueX = $posXv[i] * 1.001;
90              if((parcialValueX < 0.995) && (parcialValueX > 0.005)){
91                  $posXv[i] = parcialValueX;
92              }elseif(parcialValueX < 0.005){
93                  $posXv[i] = 0.005;
94              }elseif(parcialValueX > 0.995){
95                  $posXv[i] = 0.995;
96              }
97              parcialValueY = $posYv[i] * 1.001;
98              if((parcialValueY < 0.995) && (parcialValueY > 0.005)){
99                  $posYv[i] = parcialValueY;
100             }elseif(parcialValueY < 0.005){
101                 $posYv[i] = 0.005;
102             }elseif(parcialValueY > 0.995){
103                 $posYv[i] = 0.995;
104             }
105         }
106         parcialValueX = 0;
107         parcialValueY = 0;
108     }
109 setPositionVerticesAxisX($posXv);
110 setPositionVerticesAxisY($posYv);
111 }
```

G.1 Linha de Comando

$ glwmguash drawnet.gua -i [nomeArquivo] –retraction

Referências Bibliográficas

Albert, R.; Barabási, A.-L. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, American Physical Society, v. 74, p. 47–97, 2002.

Alvarez-hamelin, J. I.; Asta, L. D.; Barrat, A.; Vespignani, A. k-core decomposition: a tool for the visualization of large scale networks. *CoRR*, abs/cs/0504107, 2005.

Belviranli, M. E.; Dilek, A.; Dogrusoz, U. Cise: A circular spring embedder layout algorithm. *IEEE Transactions on Visualization & Computer Graphics*, v. 19, p. 953–966, 2013.

Brandes, U.; Wagner, D. Using graph layout to visualize train interconnection data. *Journal of Graph Algorithms and Applications*, v. 4, n. 3, p. 135–155, 2000.

Chen, C. *Information Visualization*. 2002.

Chen, C. *Information Visualization Beyond the Horizon*. [S.l.]: Springer-Verlag New York, Inc., 2006.

Chen, C. Information visualization. *Wiley Interdisciplinary Reviews: Computational Statistics*, John Wiley & Sons, Inc., v. 2, n. 4, p. 387–403, 2010.

Cheng, J.; Ke, Y.; Chu, S.; Ozsu, M. T. Efficient core decomposition in massive networks. In: *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 2011. p. 51–62.

Chernobelskiy, R.; Cunningham, K. I.; Goodrich, M. T.; Kobourov, S. G.; Trott, L. Force-directed lombardi-style graph drawing. In: _____. *Graph Drawing: 19th International Symposium, GD 2011, Eindhoven, The Netherlands, September 21-23, 2011, Revised Selected Papers*. [S.l.]: Springer Berlin Heidelberg, 2012. p. 320–331.

Didimo, W.; Montecchiani, F. Fast layout computation of hierarchically clustered networks: Algorithmic advances and experimental analysis. In: *IV*. [S.l.]: IEEE Computer Society, 2012. p. 18–23.

Erdős, P.; Rényi, A. On random graphs. I. *Publ. Math. Debrecen*, v. 6, p. 290–297, 1959.

Freeman, L. C. Graphic techniques for exploring social network data. In: *in Models and Methods in Social Network Analysis*. [S.l.]: Univ Press, 2004.

Fruchterman, T. M. J.; Reingold, E. M. Graph drawing by force-directed placement. *Softw. Pract. Exper.*, John Wiley & Sons, Inc., New York, NY, USA, nov. 1991.

Giacomo, E. D.; Didimo, W.; Liotta, G.; Montecchiani, F.; Tollis, I. G. Techniques for edge stratification of complex graph drawings. *J. Vis. Lang. Comput.*, v. 25, n. 4, p. 533–543, 2014.

Guo, L.; Zuo, W.; Peng, T.; Adhikari, B. K. Attribute-based edge bundling for visualizing social networks. *Physica A: Statistical Mechanics and its Applications*, v. 438, n. C, p. 48–55, 2015.

Hu, Y. Algorithms for visualizing large networks. *Combinatorial Scientific Computing*, v. 5, n. 3, p. 180–186, 2011.

Jacomy, M.; Venturini, T.; Heymann, S.; Bastian, M. Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software. *PLoS ONE*, Public Library of Science, v. 9, n. 6, 2014.

Kitchenham, B. Procedures for performing systematic reviews. *Keele, UK, Keele University*, v. 33, n. 2004, p. 1–26, 2004.

Mazza, R. *Introduction to information visualization*. [S.l.]: Springer Science & Business Media, 2009.

Monteiro, R. L. *GuaráScript: Projeto e Implementação de uma Linguagem de Programação "Open Source" Orientada à Computação Científica*. Dissertação (Mestrado) — Fundação Viscode de Cairu, 2005.

Monteiro, R. L. S. *Um Modelo Evolutivo para Simulação de Redes de Afnidade*. Tese (Doutorado) — Ufba, 2012.

Monteiro, R. L. S.; Fadigas, I. d. S.; Moret, M. A.; Pereira, H. B. B. *SCNTools 1.0.0*. [S.l.]: Instituição de registro:INPI - Instituto Nacional da Propriedade Industrial, 2010.

Moreno, J. L.; Jennings, H. H. et al. Who shall survive? *Nervous and mental disease publishing co.*, 1934.

Moreno, J. L.; Whitin, E. S.; Jennings, H. H. *Application of the group method to classification*. [S.l.]: National committee on prisons and prison labor, 1932.

Nawaz, S.; Jha, S. A graph drawing approach to sensor network localization. In: *Mobile Adhoc and Sensor Systems, 2007. MASS 2007. IEEE International Conference on*. [S.l.]: IEEE, 2007. p. 1–12.

Newman, M. E. J. The structure and function of complex networks. *SIAM REVIEW*, v. 45, p. 167–256, 2003.

Olmeda-gómez, C. Visualización de información. *Profesional de la Informacion*, v. 23, n. 3, p. 213–220, 2014.

Segal, M.; Akeley, K. *The OpenGL Graphics Systema: A Specification (Version 1.0)*. [S.l.], 1994.

Shelley, D. S.; Gunes, M. H. Gerbilsphere: Inner sphere network visualization. *Computer Networks*, Elsevier North-Holland, Inc., v. 56, n. 3, p. 1016 – 1028, 2012.

Six, J. M.; Tollis, I. G. Circular drawing algorithms. In: *Handbook on Graph Drawing and Visualization*. [S.l.: s.n.], 2013. p. 285–315.

Solomonoff, R.; Rapoport, A. Connectivity of random nets. *Bulletin of Mathematical Biology*, Kluwer Academic Publishers, v. 13, n. 2, p. 107–117, 1951.

Watts, D.; Strogatz, S. Collective dynamics of small-world networks. *Nature*, n. 393, p. 440–442, 1998.

Zhu, B.; Chen, H. Information visualization. *ARIST*, v. 39, n. 1, p. 139–177, 2005.

Zhu, B.; Watts, S.; Chen, H. Visualizing social network concepts. *Decis. Support Syst.*, v. 49, n. 2, p. 151–161, 2010.