UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Alberto Ferreira Neto

Análise e Desenvolvimento de Melhorias para o Software DyNetVis

Uberlândia, Brasil

2023

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Alberto Ferreira Neto

Análise e Desenvolvimento de Melhorias para o Software DyNetVis

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Orientador: Bruno Augusto Nassif Travençolo

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2023

Resumo

Este trabalho tem como objetivo analisar e desenvolver melhorias e funcionalidades adicionais para o software DyNetVis, com o propósito de torná-lo mais acessível e atender às necessidades dos usuários interessados na análise de redes temporais. Utilizando uma metodologia de pesquisa técnica, foram identificados problemas de desempenho na construção visual das redes e no carregamento de redes com tempos extensos e esparsos. Para solucionar esses problemas, foram realizadas melhorias na estrutura de dados utilizada para carregar a rede e na forma como o software reage ao selecionar redes específicas. Essas melhorias resultaram em uma redução relevante no tempo de execução e no consumo de memória. Com essas otimizações, o software DyNetVis se tornou mais eficiente e adequado para a análise de redes temporais.

Palavras-chave: DyNetVis, Redes complexas, Redes temporais.

Lista de ilustrações

Figura 1 – Visualização de redes temporais. (a) Layout matriz. (b) Layout estrutural. (c) Layout temporal. 13
Figura 2 – Janela de abertura de arquivo DyNetVis. 15
Figura 3 – Principais telas de visualização de redes dinamicas no DyNetVis. (a) Estrutural. (b) Temporal. (c) Matriz. 16
Figura 4 – Demonstração uso da ferramenta Profile ao executar algoritmo teste do DyNetVis. 17
Figura 5 – Exemplo uso de ArrayList para armazenar os dados do arquivo. 20
Figura 6 – Exemplo de modificação feita no código após substituir estrutura de dados. 21
Figura 7 – Consumo de memória após carregamento da rede antes e depois da alteração da estrutura. 23
Figura 8 – Exemplo de uso do método equals nas duas estruturas utilizadas. . . . 23
Figura 9 – Solução para problema de esparsidade de tempo. 25
Figura 10 – Resolução definida por padrão ao selecionar arquivo no DyNetVis. . . . 26
Figura 11 – Algoritmo implementado para exibir alerta ao usuário caso necessário escalar tempo da rede temporal. 27

Lista de tabelas

Tabela 1 – Exemplo de formato de arquivo para abrir redes temporais. 14
Tabela 2 – Informação de quantidade de nós, arestas e tempo final das redes analisadas. 19
Tabela 3 – Resultado de consumo de tempo e memória nas duas diferentes estruturas ao executar algoritmo teste. Rede com 1.899 nós e 45.007 arestas. 21
Tabela 4 – Tempo medio de cinco execuções em três redes distintas usando ArrayList e vetor de tamanho fixo. 22

Lista de abreviaturas e siglas

DyNetVis	*Dynamic Network Visualization*
TAM	Mapa de Atividade Temporal
IDE	*Integrated Development Environment*
MOOC	*Massive Open Online Course*
CNO	*Community-based Node Ordering*

Sumário

1 INTRODUÇÃO 7
1.1 Objetivos 8
1.2 Motivação 8
1.3 Metodologia 9
1.4 Organização do Trabalho 9

2 REFERENCIAL TEÓRICO 11
2.1 Redes complexas 11
2.2 Visualização de Redes Temporais 11
2.2.1 Layout Matriz 12
2.2.2 Layout estrutural 12
2.2.3 Layout Temporal 12
2.3 Dynamic Network Visualization (DyNetVis) 13
2.4 Melhorias no software 14
2.5 IDE NetBeans e ferramenta Profile 15

3 DESENVOLVIMENTO 18
3.1 Conjunto de redes analisadas 18
3.2 Desempenho no carregamento de uma rede complexa 19
3.3 Problema ao carregar redes com tempo extenso e esparso 24

4 CONCLUSÃO 28

REFERÊNCIAS 30

APÊNDICES 32

APÊNDICE A – CÓDIGO COMPARADOR 33

1 Introdução

O conceito de redes complexas tem sido importante no estudo de sistemas complexos. Em redes, as arestas conectam pares de nós formando estruturas complexas. Estudos têm mostrado que as redes não só contêm estrutura, mas também podem evoluir com o tempo. Por exemplo, um estudo de caso realizado por (PONCIANO et al., 2020) fez uso da estratégia de visualização de redes temporais para analisar as interações entre alunos e professores em redes educacionais ao longo do tempo. A adição da dimensão temporal agrega complexidade na análise e demanda o desenvolvimento de métodos inovadores para a visualização de redes da vida real.

Com o avanço da tecnologia de informação e a disponibilidade de computadores e redes de comunicação que permitem a análise de dados em grandes quantidades, houve uma mudança significativa na área. As pesquisas, antes focadas nas pequenas redes e nas propriedades de nós individuais ou arestas, passaram a considerar propriedades estatísticas em larga escala. Atualmente, são comuns estudos com redes envolvendo milhões ou bilhões de nós, as quais antes eram compostas por dezenas ou, em casos extremos, centenas de nós (BARABÁSI, 2003).

Com o objetivo de facilitar a análise de redes temporais, foi desenvolvido o *Dynamic Network Visualization System* (DyNetVis) (LINHARES et al., 2020), uma ferramenta de software para visualização de redes temporais. O sistema oferece várias ferramentas de interação do usuário e possui dois layouts visuais coordenados, denominados estrutural e temporal. O layout estrutural representa as formas convencionais de desenho de redes, onde é possível visualizar cada instante de tempo dos nós e das arestas isoladamente ou todos de uma só vez. Já o layout temporal permite a visualização simultânea de vários instantes de tempo da rede temporal. Além disso, foram implementadas algumas abordagens para a visualização do layout temporal, como a estratégia de ordenação de nós Vizinhos Recorrentes, o Mapa de Atividade Temporal (TAM) para identificar períodos de maior atividade e a técnica CNO (*Community-based Node Ordering*) que é utilizada para reduzir a confusão visual de grandes redes explorando a estrutura da comunidade em rede. (LINHARES et al., 2017).

O DyNetVis é uma ferramenta de destaque para análise visual de redes temporais. No entanto, embora ofereça ótimas funcionalidades, o software precisa de atualizações que possam aprimorar sua capacidade de análise de redes temporais e atrair um público mais amplo da comunidade científica.

Este trabalho tem como objetivo principal tornar o software mais acessível à comunidade científica, fornecendo uma análise detalhada de cada etapa do processo de

desenvolvimento. Ele abordará a identificação de áreas de melhoria, a implementação de soluções e a apresentação dos resultados alcançados.

1.1 Objetivos

O objetivo deste trabalho consiste em analisar e desenvolver melhorias e funcionalidades adicionais para o software DyNetVis, com o propósito de torná-lo mais acessível e satisfazer as necessidades dos usuários interessados na análise de redes temporais. Os objetivos específicos a serem alcançados são os seguintes:

- Analisar os requisitos do software;
- Discutir e propor novas funcionalidades e melhorias;
- Identificar e selecionar as melhores técnicas e algoritmos disponíveis;
- Desenvolver as soluções necessárias para atender aos requisitos identificados.

1.2 Motivação

No estudo de (LINHARES et al., 2017), foram identificados softwares voltados para análise quantitativa e qualitativa de redes, como Gephi (BASTIAN; HEYMANN; JACOMY, 2009), Cytoscape (SHANNON et al., 2003), DiffAni (RUFIANGE; MCGUFFIN, 2013), entre outros. No entanto, a maioria dessas ferramentas não suporta a análise e visualização de redes temporais de forma eficaz. Embora Gephi e DiffAni ofereçam recursos relacionados à representação e visualização dinâmica de redes, eles apresentam limitações em termos de métodos de construção de layout e interatividade, o que compromete a experiência de análise visual. Por exemplo, o Gephi permite visualizar a evolução da rede por meio de animação, mas essa visualização fica restrita ao layout estrutural, utilizando técnicas tradicionais de desenho gráfico, e acumula arestas e nós de épocas anteriores.

O DyNetVis visa suprir essas lacunas encontradas nas ferramentas existentes, oferecendo recursos avançados para a análise e visualização de redes temporais, superando as limitações das outras ferramentas mencionadas. É importante destacar, no entanto, que o DyNetVis ainda possui algumas limitações, principalmente no que se refere ao tamanho das redes que podem ser analisadas.

1.3 Metodologia

Este trabalho segue uma metodologia de pesquisa técnico, pois se trata de análise e desenvolvimento de atualizações para um software. Segue abaixo a metodologia usada para o desenvolvimento deste trabalho de conclusão de curso:

1. Determinando problema e levantamento bibliográfico: Inicialmente foi identificado o problema, algo ainda não explorado, que pudesse gerar o tema deste TCC. Após o tema já definido foi realizada uma busca sobre recursos bibliográficos para facilitar e complementar o estudo e desenvolvimento da pesquisa.

2. Leitura e Documentação: Realiza o estudo do material encontrado para melhor entendimento do software e sobre o seu propósito. Com domínio da usabilidade do software e compreensão sobre redes complexas foi realizado um levantamento de requisitos e assim poder decidir a viabilidade de todas as ideias, para assim documentar o que deve ser implementado de melhorias.

3. Projeto e desenvolvimento: A linguagem e ferramentas para desenvolvimento foram definidas de acordo com o que já foi utilizado no software DyNetVis. Identificando quais os melhores algoritmos necessários, foi colocado em prática o desenvolvimento dos requisitos já definidos, e documentada cada etapa deste processo.

4. Conclusão: Após conclusão das etapas anteriores, foi definida nesta etapa a conclusão sobre os resultados obtidos.

1.4 Organização do Trabalho

Este trabalho se inicia com uma introdução que apresenta a importância de aprimorar sistemas de visualização de redes temporais e destaca os benefícios específicos proporcionados pelo DyNetVis.

Em seguida, é apresentado o referencial teórico, abordando conceitos e técnicas relevantes relacionadas à visualização de redes temporais e melhorias de software. São explorados métodos de visualização, funcionalidades específicas do DyNetVis e ferramentas utilizadas no desenvolvimento e manutenção do software.

Na sequência, são detalhadas as possíveis melhorias propostas para o software. São discutidas as abordagens utilizadas para identificar os problemas e encontrar soluções, bem como a implementação das soluções propostas. Durante essa etapa são fornecidos exemplos práticos de imagem de trechos de código que ilustram as alterações realizadas.

Por fim, o trabalho conclui abordando as dificuldades encontradas durante o processo de implementação das melhorias, realizando um resumo dos principais pontos abor-

dados ao longo do trabalho e apresentando os resultados obtidos com as melhorias propostas.

2 Referencial Teórico

Este capítulo apresenta os diversos aspectos teóricos envolvidos no trabalho de pesquisa. Detalhando métodos e ferramentas para desenvolvimento e os conceitos que estão relacionados diretamente ao software já existente.

2.1 Redes complexas

A teoria das redes complexas tem sido amplamente explorada em diversos campos científicos como uma ferramenta poderosa para compreender a estrutura e o funcionamento de sistemas complexos. A abordagem das redes complexas oferece uma nova perspectiva para descrever e analisar sistemas complexos, independentemente de sua natureza específica (BARABÁSI, 2016). Esse campo de estudo tem despertado interesse devido à sua aplicabilidade em áreas como física, biologia, ciência da computação e sociologia.

As redes complexas são caracterizadas pela presença de propriedades emergentes que não podem ser deduzidas apenas ao examinar os nós individualmente, incluindo a formação de hubs, a presença de comunidades densamente interligadas e a robustez diante de falhas ou ataques (NEWMAN, 2003). Além disso, as redes complexas exibem uma distribuição de grau não uniforme, seguindo uma lei de potência, o que influencia sua resiliência e eficiência.

2.2 Visualização de Redes Temporais

A visualização de redes temporais permite a análise de padrões, tendências, anomalias e outros tipos de comportamentos existentes em dados reais que possuem informação temporal (LINHARES et al., 2017). A forma mais comum de representar redes complexas é por meio de grafos, sendo grafo uma estrutura matemática formada por dois conjuntos finitos V e E. Os elementos de V são chamados de vértices ou nós. Os elementos de E são denominadas arestas. Cada aresta é formada por um par de nós (v,w) não necessariamente distintos. Uma rede temporal, também conhecida como rede variável no tempo, é uma rede em que as arestas estão ativas em um determinado instante de tempo.

Existem algumas formas de visualização de redes complexas (por exemplo forma de matriz, estrutural e temporal), onde de acordo com o tipo de pesquisa pode ser utilizado determinada forma para facilitar a análise.

2.2.1 Layout Matriz

Essa forma de visualização apresentada na Figura 1a é feita por uma matriz quadrada e simétrica, onde linhas e colunas representam nós, a conexão entre esses nós é feita pela célula que corresponde ao nó da linha e ao nó da coluna. Nessa forma de visualização a percepção de padrões é dificultada, em especial quando ela possui um grande número de informações. Uma forma de fazer uma visualização temporal de rede complexa representada por uma matriz é utilizando de quadro de animação, em que cada quadro representa um tempo e mostra as conexões naqueles instantes de tempo.

2.2.2 Layout estrutural

O layout estrutural é realizado na forma convencional de um grafo, em que cada conexão é estabelecida por uma aresta que conecta dois nós. O nó pode representar, por exemplo, pessoas, enquanto a aresta representa o relacionamento entre elas. Semelhante a visão em forma de matriz, para que se tenha uma análise temporal de uma forma estrutural é necessário quadros de animação mostrando em cada instante as conexões do grafo.

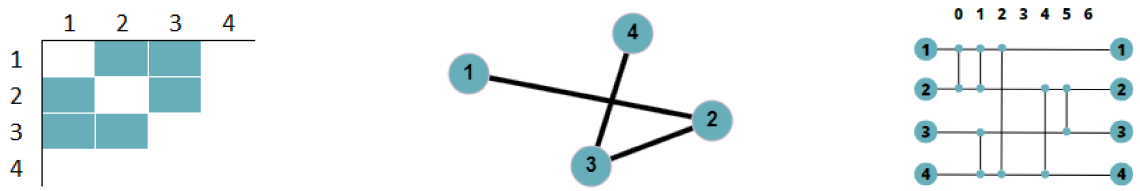
Este tipo de visão é mais utilizado para analisar a rede como um todo, facilitando também a identificação dos nós com um maior número de conexões. Um exemplo de layout estrutural pode ser observado na Figura 1b.

2.2.3 Layout Temporal

O layout temporal baseia-se em um layout que utiliza o eixo das abscisas para representar o tempo e o eixo das ordenadas para representar os nós da rede. As conexões entre nós são feitas por meio de uma reta que traça a ligação de um nó a outro, de acordo com o instante de tempo em que ocorreu a conexão. Com essas características que são demonstradas na Figura 1c é possível analisar a rede e a evolução da rede ao longo do tempo.

Essa forma de visão é muito importante e pode gerar boa percepção dos padrões da rede, mas com o mau posicionamento dos nós pode dificultar a análise. Devido as arestas serem representadas por traços no eixo, pode ocorrer sobreposição entre arestas, principalmente se dois nós estiverem muito distantes entre eles, gerando uma aresta muito grande. Se os nós forem posicionados de formas diferentes, podem gerar visualizações distintas da rede temporal. Nesse sentido é importante a organização dos nós para facilitar a análise, existem vários métodos de organização, vamos citar alguns a seguir.

1. **Nascimento:** Nesta ordenação os nós vão ser ordenados de acordo com ordem de conexões no tempo, ou seja os nós que no primeiro instante de tempo tiverem



(a) (b) (c)

Figura 1 – Visualização de redes temporais. (a) Layout matriz. (b) Layout estrutural. (c) Layout temporal.

conexões serão posicionados no topo já os nós que tiverem conexão por último ficaram no final do eixo.

2. **Lexicográfico:** Essa ordenação segue uma ordem crescente de rótulo, mas depende do tipo do rótulo, se for numérico segue ordem crescente dos números e se for letras segue ordem alfabética.

3. **Vizinhos Recorrentes:** Esta ordenação segue a regra de aproximação entre os nós, ou seja, a ordem é gerada de acordo com os vizinhos de determinado nó. Primeiro coloca o nó com mais conexões no centro do layout, depois são posicionados ao seu redor os nós que são seus vizinhos e, de forma recorrente, vai posicionando os demais nós.

2.3 Dynamic Network Visualization (DyNetVis)

O DyNetVis é um software de análise visual de redes temporais que está disponível gratuitamente como uma ferramenta de código aberto. Ele oferece aos usuários quatro layouts diferentes e uma variedade de técnicas avançadas, como ordenação de nós e arestas, amostragem, entre outras. Além disso, o DyNetVis também inclui a implementação de processos dinâmicos, incluindo modelos epidêmicos comuns (LINHARES et al., 2017; LINHARES et al., 2019; LINHARES et al., 2020; PONCIANO et al., 2021b).

O software gera a rede com base em um arquivo de entrada fornecido. Cada linha do arquivo deve conter a identificação do Nó 1, seguida pela identificação do Nó 2 e o tempo em que ocorreu a conexão entre eles, conforme mostrado na Tabela 1. É importante destacar que o tempo deve ser iniciado a partir de zero ou um. A ordem em que os nós 1 e 2 são especificados em cada linha não afeta a criação da rede no software, uma vez que o sistema DyNetVis representa apenas grafos não direcionados.

É importante destacar que o tempo deve ser iniciado a partir de zero ou um. A ordem em que os nós 1 e 2 são especificados em cada linha não afeta a criação da rede no software, uma vez que o sistema DyNetVis representa apenas grafos não direcionados.

Tabela 1 – Exemplo de formato de arquivo para abrir redes temporais.

Nó 1 ID	Nó 2 ID	Tempo
1	2	0
3	4	32
5	2	104

Ao abrir o software, o usuário tem a opção de selecionar a janela de abertura de arquivo, de acordo com a Figura 2. Nessa janela, ele pode escolher o tipo de layout que deseja visualizar, sendo as opções temporal e estrutural (as duas são carregadas juntas), matriz e comunidade. Após decidir o tipo de layout, o usuário pode inserir um arquivo no formato mencionado anteriormente ou optar por utilizar uma rede aleatória.

O próximo passo é definir um filtro de tempo, no qual o usuário seleciona o tempo mínimo e máximo da rede que deseja visualizar. Por fim, há a opção de reescalar o tempo da rede, oferecendo três algoritmos: "static (Uniform)", "Balanced Visual Complexity" e "Resolução Adaptativa". O algoritmo de resolução adaptativa requer parâmetros de "Window size" e "Fading Factor" (PONCIANO et al., 2021a).

Após definir cada etapa na janela de arquivo, o usuário pode clicar em "OK" e iniciar o carregamento da rede.

O sistema subdivide suas funcionalidades em três telas principais, contendo layout estrutural (Figura 3a), temporal (Figura 3b) e matriz (Figura 3c).

2.4 Melhorias no software

A melhoria de software desempenha um papel fundamental no desenvolvimento e manutenção de sistemas, visando aprimorar a qualidade, desempenho e funcionalidade do software existente. Essa abordagem abrange a identificação de áreas de aprimoramento, a definição de metas e a implementação de mudanças para aperfeiçoar o sistema de software.

A busca por melhorias no software requer o uso de abordagens sistemáticas e técnicas adequadas. Segundo (PRESSMAN, 2015, p. 214), "o processo de melhoria de software envolve a análise detalhada do sistema, identificação de pontos fracos, definição de metas claras e a implementação de mudanças efetivas para alcançar um software de maior qualidade e eficiência". Para isso, é importante que os profissionais envolvidos utilizem metodologias e ferramentas apropriadas para conduzir o processo de melhoria de forma eficaz.

Além disso, as melhorias no software são um processo contínuo e evolutivo. Conforme destaca (SOMMERVILLE, 2016, p. 275), "a melhoria de software vai além das correções imediatas, envolvendo a análise das necessidades dos usuários, a identificação de oportunidades de otimização e a implementação de mudanças que agreguem valor ao

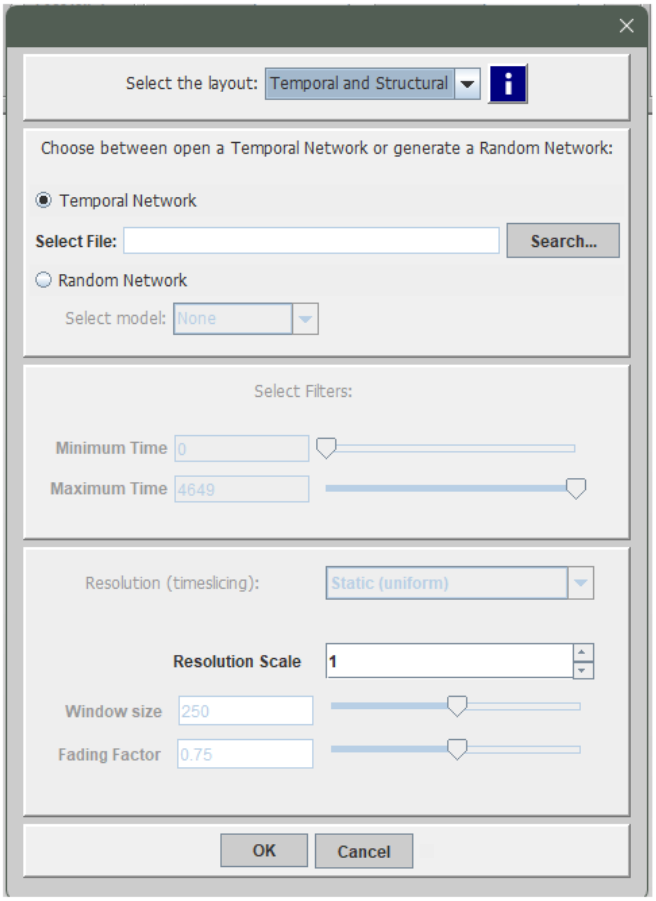


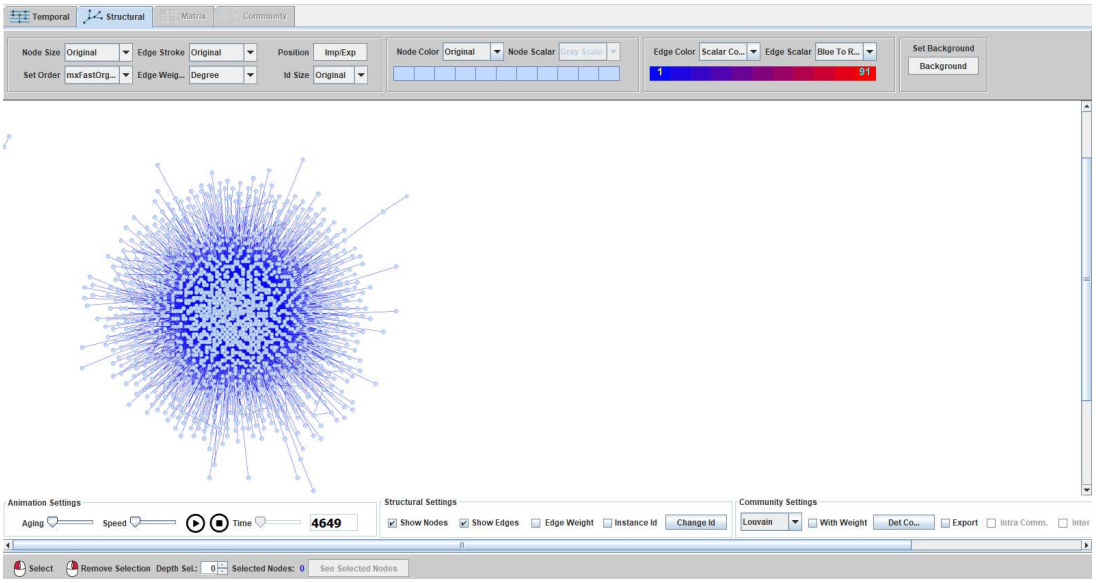
Figura 2 – Janela de abertura de arquivo DyNetVis.

sistema e atendam às expectativas dos usuários". Dessa forma, as melhorias no software desempenham um papel crucial na manutenção do software atualizado, competitivo e alinhado com as demandas do mercado.

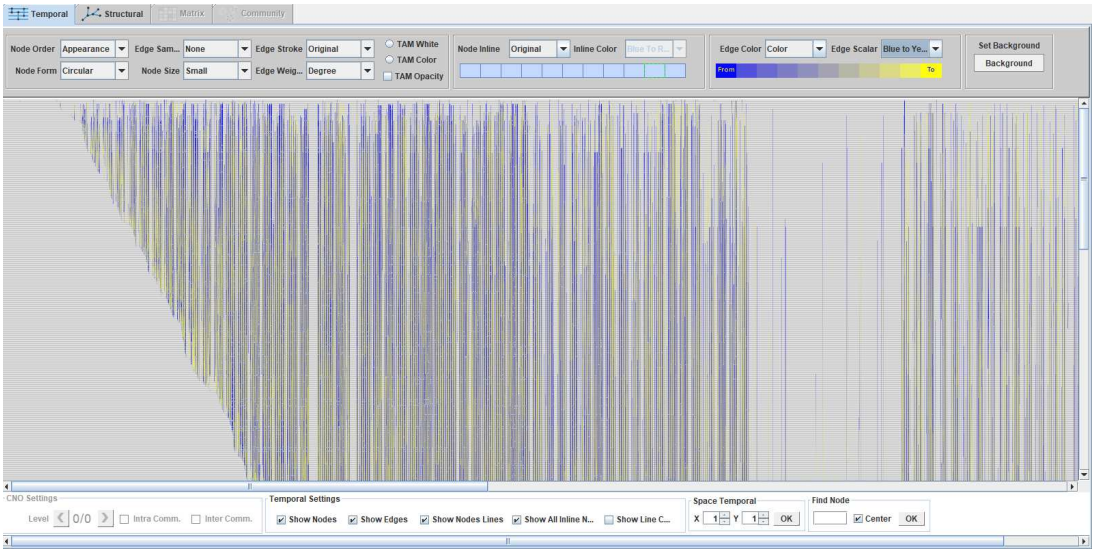
2.5 IDE NetBeans e ferramenta Profile

O desenvolvimento de software em linguagem Java tem sido amplamente adotado e o uso de ambientes de desenvolvimento integrados (IDEs) desempenha um papel fundamental nesse processo. O NetBeans, uma das IDEs mais populares para Java, oferece uma ampla gama de recursos e funcionalidades que facilitam a criação, depuração e manutenção de aplicativos Java.

Segundo (SILVA; FERREIRA; SOUZA, 2018), "o NetBeans é uma IDE de código aberto amplamente utilizada pela comunidade de desenvolvedores Java devido à sua interface intuitiva e recursos avançados, que permitem uma programação eficiente e de alta qualidade". Essa IDE oferece suporte para diversas tecnologias e frameworks Java, como Swing, JavaFX e Servlets, permitindo o desenvolvimento de aplicativos desktop, web e mobile de forma integrada. Além disso, uma das ferramentas mais relevantes do NetBe-



(a)



(b)

(c)

Figura 3 – Principais telas de visualização de redes dinamicas no DyNetVis. (a) Estrutural. (b) Temporal. (c) Matriz.

ans é o Profile, que permite a análise e otimização do desempenho de aplicativos Java. O Profile oferece recursos avançados de monitoramento e análise de uso de memória, tempo de execução, consumo de CPU e outros aspectos críticos para a identificação de gargalos e possíveis melhorias no código.

A utilização do Profile como parte do processo de desenvolvimento de software Java oferece vantagens significativas. De acordo com (GAMA; SANTOS; SILVA, 2019), "a ferramenta Profile do NetBeans permite a identificação de trechos de código ineficientes, alocação excessiva de memória, problemas de concorrência e outras questões que podem impactar negativamente o desempenho do aplicativo".

Figura 4 – Demonstração uso da ferramenta Profile.

Na Figura 4, é possível observar a demonstração da ferramenta Profile em ação. Com ajuda da análise detalhada dos dados coletados pelo Profile, os desenvolvedores podem identificar gargalos de desempenho, como chamadas de métodos demorados ou uso excessivo de recursos, e tomar as medidas necessárias para otimizar o código-fonte, tornando-o mais eficiente e reduzindo possíveis problemas de desempenho.

3 Desenvolvimento

Este capítulo descreve as principais etapas para o desenvolvimento deste trabalho desde a identificação dos possíveis problemas que o software apresenta ou funcionalidades que podem ser melhoradas, até o desenvolvimento das funcionalidades e a interpretação dos resultados obtidos com as melhorias implementadas.

3.1 Conjunto de redes analisadas

Durante os testes realizados no DyNetVis, utilizamos três conjuntos de dados de entrada (Tabela 2) em um ambiente de teste composto por um notebook Samsung com 8 gigas de RAM e processador Intel Celeron 3205U. O objetivo desses testes foi identificar possíveis problemas e áreas de melhoria no software. Para garantir que o software lesse esses arquivos corretamente, foram realizados ajustes e modificações em sua configuração de leitura. As modificações foram necessárias para assegurar a integridade dos dados e a compatibilidade com as funcionalidades do software. Segue abaixo as informações sobre as redes que foram utilizadas:

1. **Bitcoin OTC:** Esta é a rede de "quem-confia-em-quem" de pessoas que realizam negociações usando Bitcoin na plataforma Bitcoin OTC. Devido ao anonimato dos usuários, é essencial manter um registro da reputação de cada indivíduo, a fim de evitar transações com pessoas fraudulentas ou arriscadas. Na plataforma Bitcoin OTC, os membros têm a opção de avaliar outros membros em uma escala que varia de -10 (total desconfiança) a +10 (total confiança), em incrementos de 1 (KUMAR et al., 2016).

2. **CollegeMSG:** Este conjunto de dados é composto por mensagens privadas enviadas em uma rede social online na Universidade da Califórnia. Os usuários podiam pesquisar a rede em busca de outras pessoas e, em seguida, iniciar conversas com base nas informações do perfil. Uma aresta (u, v, t) significa que o usuário u enviou uma mensagem privada para o usuário v no momento t (PANZARASA; OPSAHL; CARLEY, 2009).

3. **MOOC:** O conjunto de dados de ações de usuários de MOOC representa as ações realizadas pelos usuários em uma plataforma MOOC popular. Essas ações são representadas como uma rede direcionada e temporal. No entanto o sistema trata a rede como não direcional. Os nós da rede representam os usuários e as atividades do curso (alvos), enquanto as arestas representam as ações dos usuários nos respectivos

alvos. Cada ação possui atributos específicos e um registro de data/hora (KUMAR; ZHANG; LESKOVEC, 2019).

Tabela 2 – Informação de quantidade de nós, arestas e tempo final das redes analisadas.

Rede	Nós	Arestas	Tempo Final (s)
Bitcoin	5.881	35.592	1.453.684.323
College	1.899	45.007	1.098.777.142
MOOC	5.449	131.161	2.572.086

3.2 Desempenho no carregamento de uma rede complexa

Durante a análise e utilização do software DyNetVis, foi identificado um problema significativo relacionado ao desempenho ao carregar redes complexas com um grande número de conexões. Ao examinar uma rede contendo 1.899 nós e 45.007 arestas, observou-se um tempo médio de carregamento de quase 2 minutos. A demora no carregamento da rede no DynetVis representa uma limitação importante, pois afeta negativamente a eficiência e usabilidade da ferramenta. Essa descoberta foi obtida por meio de testes e monitoramento do tempo de carregamento em diferentes configurações de redes, proporcionando uma compreensão mais aprofundada desse problema específico.

Os estudos para solucionar o problema concentram-se na análise do algoritmo utilizado para a geração da rede. Portanto, não foi necessário realizar uma investigação detalhada na parte gráfica do software. O foco principal é compreender e aprimorar o processo de geração da rede, visando otimizar o desempenho durante o carregamento.

Inicialmente, é identificada uma classe responsável por receber os parâmetros do usuário, como o tipo de rede, a leitura de arquivos ou a utilização de informações previamente salvas no software, o tempo mínimo e máximo para uma conexão entre os nós e, por fim, a resolução da rede. Essa classe é denominada *OpenSetDialog.java* e pertence ao pacote *forms*. Por meio dela, realiza-se a leitura do arquivo, linha por linha. No arquivo, encontra-se um método que é executado ao clicar no botão “OK”. Esse método percorre o arquivo por meio de um laço, armazenando as informações da linha, que são o primeiro nó, o segundo nó e o tempo, em uma variável chamada “coluna” do tipo *ArrayList<Integer>*. Após cada iteração do laço, a variável “coluna” é armazenada em outra variável chamada *MatrizDataInline*, do tipo *ArrayList<ArrayList>*. Essa variável contém as informações necessárias para a construção da rede complexa. Esse procedimento é detalhado de forma simplificada na Figura 5.

Após análise do código, identificou-se uma possível melhoria de desempenho por meio da substituição da estrutura de dados utilizada para armazenar as informações do

```
ArrayList<ArrayList> matrizDataInline = new ArrayList<>();

// Percorrer cada linha do arquivo.
while (in.hasNextLine()) {

    // Array para nós 1, nó 2 e tempo.
    ArrayList<Integer> coluna = new ArrayList<>();


    String line = in.nextLine();
    // Separando valores na linha por tabulação
    String[] tokens = line.split(regex: "[ \\t]");

    // Converte valores para inteiro
    int no1 = Integer.parseInt(tokens[0]);
    int no2 = Integer.parseInt(tokens[1]);
    int tempo = Integer.parseInt(tokens[2]);

    coluna.add(e: no1);
    coluna.add(e: no2);
    coluna.add(e: tempo);

    if(!matrizDataInline.contains(o: coluna)){
        matrizDataInline.add(e: coluna);
    }
}
```

Figura 5 – Exemplo código fonte do software fazendo leitura do arquivo e armazenando dados em um ArrayList.

arquivo. É importante destacar que o algoritmo foi projetado para ler arquivos com exatamente três inteiros em cada linha. Qualquer desvio desta regra resulta em um erro. Com base nessa observação, foi concluído que um vetor de tamanho fixo seria suficiente para armazenar essas informações. Como resultado, sugere-se substituir a variável conhecida como "coluna", que pertence ao tipo *ArrayList<Integer>*, por um vetor do tipo *int[]*, cujo tamanho é previamente definido em três posições.

Devido ao fato de o ArrayList ser uma estrutura de dados dinâmica, capaz de armazenar objetos *Integer*, é natural esperar que seu desempenho e consumo de memória sejam superiores em relação a um vetor de inteiros com tamanho fixo. A fim de analisar essa comparação, um algoritmo de teste foi desenvolvido, ele pode ser encontrado no Apêndice A.

O algoritmo criado tem por objetivo simular a parte do software DyNetVis que lê o arquivo recebido e armazena as informações em uma determinada estrutura de dados. Durante os testes, o arquivo College.txt contendo um total de 1.899 nós e 45.007 arestas foi usado como entrada. A Tabela 3 apresenta uma análise comparativa, revelando que a utilização do vetor de tamanho fixo resultou em uma melhoria de desempenho de aproxi-

Tabela 3 – Resultado de consumo de tempo e memória nas duas diferentes estruturas ao executar algoritmo teste. Rede com 1.899 nós e 45.007 arestas.

	Tempo (s)		Memória (mb)	
Execução	ArrayList<Integer>	int[3]	ArrayList<Integer>	int[3]
1º	27	17	54	48
2º	27	17	54	49
3º	27	17	53	51
4º	26	17	55	51
5º	28	16	55	51
Média	27	17	54	50

madamente 37% no tempo de execução. Esses resultados mostram que a modificação tem um impacto positivo no software.

Com a identificação dos ganhos ao substituir um ArrayList por um vetor de tamanho fixo, foi necessário realizar uma análise das classes afetadas diretamente por essa modificação e determinar se era viável efetuar a mudança sem comprometer outras funcionalidades do sistema. Fazendo uso das ferramentas de buscas disponíveis na IDE NetBeans foi possível localizar o total de cinco classes que recebem como parâmetro a variável *MatrizDataInline*, responsável pelo armazenamento da rede temporal. Mapeando o uso da variável em cada classe foram encontrados os métodos responsáveis por recuperar os dados no Array e definir as correções necessárias para uso da nova estrutura.

ArrayList<Integer>

```
for(ArrayList<Integer> column : netLayoutInlineNew.matrizDataInline)
{
    String[] tokens =  new String[9];
    tokens[0] = column.get(index: 0).toString(); //origem
    tokens[1] = column.get(index: 1).toString(); //destino
    tokens[2] = column.get(index: 2).toString(); //tempo
```

int[3]

```
for(int[] column : netLayoutInlineNew.matrizDataInline)
{
    String[] tokens =  new String[3];

    tokens[0] = Integer.toString(column[0]); //origem
    tokens[1] = Integer.toString(column[1]); //destino
    tokens[2] = Integer.toString(column[2]); //tempo
```

Figura 6 – Exemplo de modificação feita no código após substituir estrutura de dados.

Por se tratar de uma variável que apenas carrega os dados da rede e distribui os dados em outras classes para criar diferentes tipos de visualização de uma rede complexa, foi possível realizar a substituição do tipo de estrutura utilizado sem prejudicar o funcio-

namento do software. Na Figura 6 vemos um exemplo de como os métodos recuperavam os dados da variável *MatrizDataInline* e de como eles ficaram após a substituição da estrutura de dados utilizada.

Após a modificação dos tipos de variáveis na construção da rede, foram obtidos resultados significativos em termos de economia de tempo ao carregar três redes distintas, conforme demonstrado na Tabela 4. Ao utilizar essas três redes complexas, observamos uma média de redução de 48% no tempo de carregamento após a substituição da estrutura de dados utilizada. Esse ganho reforça a importância da modificação implementada, proporcionando um processo mais rápido no carregamento das redes complexas.

Tabela 4 – Tempo medio de cinco execuções em três redes distintas usando ArrayList e vetor de tamanho fixo.

Tempo (s)		
Rede	ArrayList<Integer>	vet[3]
Bitcoin	83	55
College	118	63
MOOC	1079	351

Com a implementação da melhoria foi possível notar com uso da ferramenta Profile do NetBeans um consumo maior de memória do software em comparação a versão anterior, conforme demonstrado na Figura 7. Isso não era esperado, pois de acordo com o que foi visto na Tabela 3 o uso de um vetor de tamanho fixo consome menos memória do que um ArrayList. Para identificar o problema foi feita uma análise focada apenas na parte de carregamento da rede, em que os valores de cada linha do arquivo são atribuídos a um vetor de três posições.

Analisando o tamanho da variável *MatrizDataInline* após o carregamento total do arquivo, foi identificado que a quantidade de arestas armazenada após a alteração da estrutura era maior que na versão anterior. Isso sugere que em algum ponto do processo de filtragem das arestas inseridas, houve um mau funcionamento com a nova estrutura adotada. Como ilustrado na Figura 5 o algoritmo fazia uma verificação se os novos valores a serem inseridos no *MatrizDataInline* já estavam ou não contidos no array.

Ao comparar essa etapa do código da nova versão com a versão anterior, foi identificado que a condicional *if(!matrizdatainline().contains(coluna))* não estava filtrando as linhas repetidas do arquivo, resultando em valores duplicados no array. Esse problema foi causado pelo método *contains* que, por definição, utiliza o método *equals* para fazer comparação de dois objetos. No caso de um *ArrayList*, o método *equals* retorna *true* ao comparar dois *ArrayList* com valores idênticos. Por outro lado, em um vetor de tamanho fixo, o método *equals* compara apenas os endereços de memória dos vetores, não os valores atribuídos a eles. Para resolver o problema, foi desenvolvido um novo método que substitui o uso do *contains* utilizado na versão anterior. Esse novo método possui a mesma

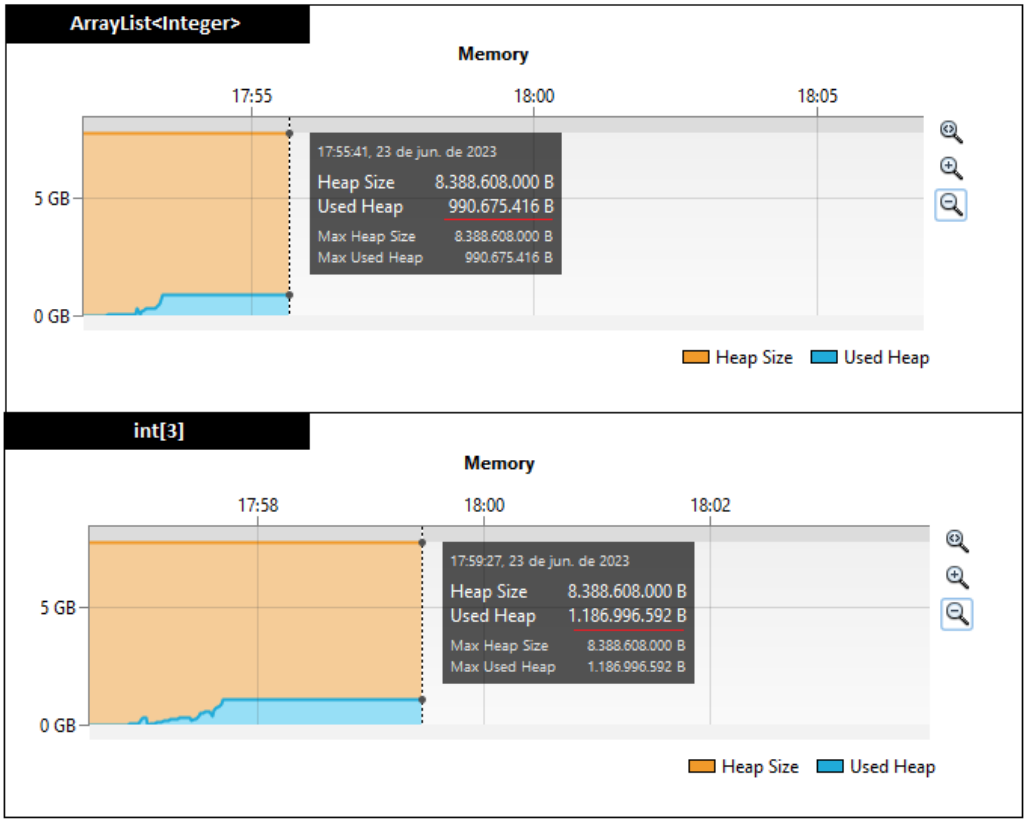


Figura 7 – Consumo de memória após carregamento da rede antes e depois da alteração da estrutura.

finalidade do *contains*, ou seja, verifica se um novo vetor de tamanho fixo a ser adicionado à variável *MatrizDataInline* já existe no array. Ele retorna *true* caso o vetor exista e *false* caso contrário. A diferença é que o novo método realiza a comparação das três posições do vetor de entrada com os vetores existentes no *ArrayList*. O código correspondente está disponível na Figura 8.

```
private boolean containsElement(ArrayList<int[]> matrizdata, int[] element){
    for (int[] i : matrizdata) {
        if (i[0] == element[0] &&
            i[1] == element[1] &&
            i[2] == element[2]){
            return true;
        }
    }
    return false;
}
```

Figura 8 – Exemplo de uso do método equals nas duas estruturas utilizadas.

3.3 Problema ao carregar redes com tempo extenso e esparso

Ao executar o software DyNetVis em algumas redes temporais, foi observado um problema em que, após clicar no botão "OK" para gerar a rede, algumas delas simplesmente não carregavam e não apresentavam nenhum erro de execução. O software continuava executando por horas mostrando apenas a barra de progresso, semelhante ao que acontece quando se entra em um laço infinito. Outro fator importante neste problema é o consumo de memória, ao deixar o algoritmo excutando foi notado um consumo excessivo de memória RAM durante a execução, chegando ao limite máximo em um computador com 8 gigabytes de memória.

Analisando as diferenças entre os arquivos que apresentavam problemas e os demais, foi possível perceber que o problema ocorria nas redes que possuíam um intervalo de tempo muito extenso. Um exemplo é o arquivo bitcoin.txt, cuja rede apresenta um valor máximo de tempo de 1.453.684.323. Durante os testes realizados, observou-se que o desempenho do algoritmo começava a ser afetado quando o tempo máximo ultrapassava 4 dígitos. Em alguns casos, redes com tempo máximo de 5 dígitos eram carregadas com grande lentidão. No entanto, quando o tempo máximo ultrapassava 6 dígitos, o sistema não conseguia realizar o carregamento.

Após identificar uma possível causa do problema no arquivo, foi necessário analisar o código e entender em qual momento o problema era gerado. Utilizando as ferramentas do ambiente de desenvolvimento integrado (IDE), foi possível acompanhar cada etapa da execução e identificar uma estrutura de repetição que iniciava em zero e percorria até o valor máximo de tempo do arquivo. Essa parte do código tem por objetivo inserir a linha de tempo em um modelo de visão temporal, sendo cada instante do tempo adicionado como um objeto. Usando o arquivo bitcoin.txt como exemplo podemos observar que essa estrutura de repetição vai de 0 a 1.453.684.323 e a cada iteração ela cria um objeto para representar o valor do tempo naquele instante, o estouro de memória ocorria antes mesmo do algoritmo chegar ao fim do laço.

Além do tempo máximo extenso, também foi identificado um problema de esparsidade temporal, ou seja, um intervalo muito grande de tempo sem arestas. Esse problema gera uma visão temporal com grandes espaços vazios, sem arestas conectando os nós, atrapalhando a análise do usuário em relação a rede. No código a remoção da esparsidade também poderia contribuir para solução do problema, pois sem a inclusão dos instantes de tempo que não possuem arestas poderíamos evitar o estouro de memória ou até mesmo reduzir o tempo gasto para carregar a rede.

Foi pensado em algo que identificasse os intervalos sem arestas e ao invés de inserir os valores de tempo desse intervalo, fosse inserido apenas um objeto que fosse representado por três pontos que do tempo A ao tempo B não possui arestas a serem mostradas,

um exemplo dessa solução é mostrado na Figura 9. No entanto, a implementação dessa abordagem exigiria um estudo aprofundado e possíveis modificações no uso da biblioteca MxGraph, responsável pela estrutura do grafo e suas visualizações. Dado o potencial de complexidade envolvido, foi decidido usar uma abordagem mais simples para solucionar o problema.

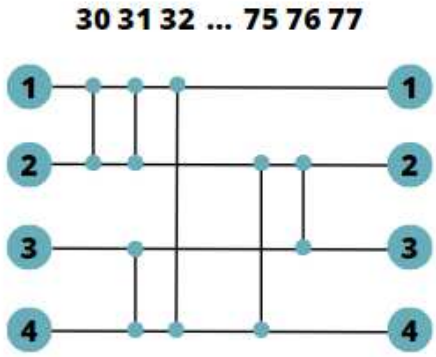


Figura 9 – Solução para problema de esparsidade de tempo.

Uma nova proposta foi reescalar o tempo. Essa abordagem consiste em redefinir os intervalos de tempo de forma a reduzir a diferença entre o menor e o maior valor, permitindo uma distribuição mais uniforme das conexões ao longo do tempo. Foi desenvolvido um código simples para reescalar o tempo nos arquivos que apresentaram o problema. Em seguida, os novos arquivos foram testados no DyNetVis para verificar se o problema era solucionado. Como esperado, o software executou sem apresentar consumo total da memória e não excedeu o tempo médio de execução.

No DyNetVis, o usuário tem à disposição três opções de algoritmos para reescalar o tempo, porém essa funcionalidade é opcional na tela de carregamento do arquivo. Caso o usuário não escolha um dos algoritmos, o software por padrão mantém a opção "Resolution: Static (uniform)" com a escala de resolução igual 1, conforme mostrado na Figura 10. É importante destacar que, em qualquer uma das opções, o software já realiza a reescala do tempo, subtraindo o menor valor dos demais, garantindo que o tempo inicie em zero.

Como o software já disponibiliza os algoritmos para reescalar o tempo, não foi necessário uma nova implementação, mas sim uma abordagem de reescalar antes do carregamento do arquivo ou exibir um alerta orientando esse procedimento. Analisando o algoritmo pode-se perceber que seria inviável a primeira abordagem, pois o algoritmo só carrega o arquivo após o usuário definir os parâmetros de inicialização, ou seja, já definindo qual algoritmo usar para reescalar o tempo. Sendo assim foi decidido o uso da segunda abordagem, em que seria implementado um alerta, avisando que a rede só é carregada

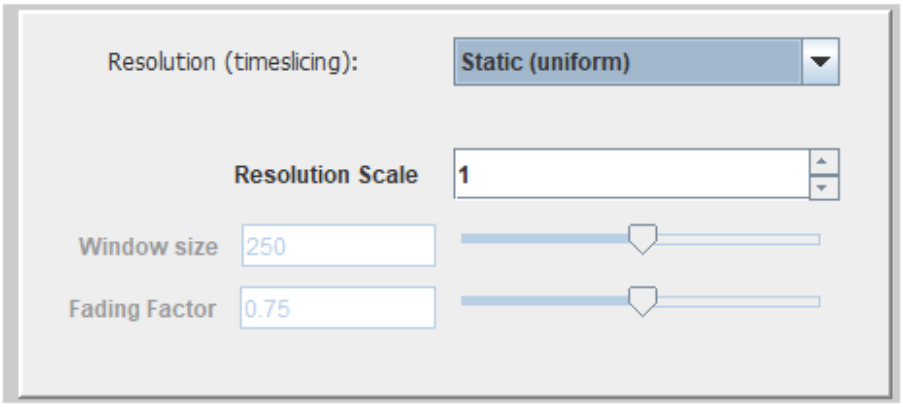


Figura 10 – Resolução definida por padrão ao selecionar arquivo no DyNetVis.

após o usuário reescalar o tempo, evitando assim o problema que impede o carregamento da rede.

A solução do problema foi ler o arquivo após o usuário inserir ele como parâmetro, identificar o tempo máximo da rede temporal e armazenar esse valor em uma variável. Foi utilizada uma condicional para verificar se o tempo máximo era maior que 9.999, o que indicaria um valor com no máximo 4 dígitos. Caso essa condição for verdadeira, um alerta será exibido com a seguinte mensagem "O intervalo de tempo na rede temporal é muito longo. Por favor, reescale o tempo. Foi inserido um parâmetro de escala de resolução como sugestão.". Além disso, o parâmetro "Resolution Scale" será ajustado para um valor que reescale o tempo para até 4 dígitos. É importante relembrar que o desempenho do software é comprometido quando o maior valor de tempo ultrapassa 4 dígitos, e por essa razão, sempre que houver problemas no carregamento, o software irá recomendar reescalar com essa quantidade de dígitos. O algoritmo implementado pode ser visto na Figura 11.

Como resultado do que foi feito, foi possível garantir que qualquer tipo de rede temporal pode ser carregada pelo software, indiferente da forma que o tempo é distribuído nas arestas. Outro fator importante é que o consumo total de memória durante o carregamento das redes foi evitado. Além disso, a reescala do tempo possibilitou uma visualização mais adequada das redes temporais, eliminando a esparsidade temporal e permitindo uma análise mais precisa por parte dos usuários.

```
if(maiorTempo > 9999){

    JOptionPane.showMessageDialog(parentComponent:this,
            "The time span of the network is too long. "
                + "Please change the temporal resolution. "
                + "A resolution scale is suggest",
            title: "Warning",messageType:JOptionPane.WARNING_MESSAGE);

    // Calcula parametro de escala de resolução mais adequada para o arquivo,
    //e inseri no campo.
    double resolution = Math.pow(a: 10, ((maiorTempo+"").length() - 4));

    resolutionSpinner.setValue((int) resolution);
}else{
    this.resolutionSpinner.setValue(value: 1);
}
```

Figura 11 – Algoritmo implementado para exibir alerta ao usuário caso necessário escalar tempo da rede temporal.

4 Conclusão

Neste trabalho, foi possível identificar e abordar duas principais questões relacionadas ao desempenho do software DyNetVis: o desempenho de tempo e memória ao carregar as redes complexas e o carregamento de redes esparsas com um intervalo de tempo muito extenso. Durante o processo de desenvolvimento das melhorias, uma das etapas mais desafiadoras foi analisar e entender o código existente, uma vez que compreender as funcionalidades e o contexto do software não é uma tarefa fácil quando se trabalha em um código desenvolvido por outras pessoas.

Em relação ao carregamento de redes complexas, foi constatado que o tempo de carregamento no DyNetVis era significativamente alto para redes com um grande número de nós e arestas. Isso impactava negativamente a eficiência e usabilidade da ferramenta. Após uma análise aprofundada do algoritmo utilizado para a geração da rede, foi identificada uma oportunidade de melhoria ao substituir a estrutura de dados utilizada para armazenar as informações do arquivo.

Inicialmente, o software utilizava um *ArrayList<Integer>* para armazenar os dados do arquivo, o que resultava em um desempenho mais lento e um maior consumo de memória. Por meio de testes comparativos, foi constatado que a substituição desse *ArrayList* por um vetor *int[3]*, de tamanho fixo, resultou em uma melhoria significativa no tempo de execução, com uma redução de aproximadamente 37% no tempo de carregamento das redes complexas.

Além disso, foi necessário realizar ajustes nas classes afetadas pela mudança na estrutura de dados, garantindo que a substituição não comprometesse outras funcionalidades do software. Após a implementação da melhoria e as correções necessárias dos problemas gerados pela substituição da estrutura de dados, observou-se uma redução média de 48% no tempo de carregamento ao testar em três redes complexas diferentes.

No que se refere ao carregamento de redes esparsas, foi identificado um problema em que algumas redes não eram carregadas corretamente, resultando em um consumo excessivo de memória RAM e uma execução aparentemente infinita. Esse problema estava relacionado a redes com o valor de tempo final representado por um número muito grande.

Após uma análise mais detalhada, verificou-se que o problema ocorria devido à forma como o software tratava essas redes. Ao analisar as diferenças entre os arquivos que apresentavam o problema e os demais, foi possível identificar a causa raiz e propor soluções.

Entre as soluções propostas a mais viável foi a implementação de mecanismos de

controle para evitar que redes com tempo final extenso fossem carregadas, alertando o usuário sobre a necessidade de optar entre alguma das técnicas de normalização disponibilizadas pelo software. Com a aplicação da solução, foi possível resolver o problema de carregamento das redes e melhorar a estabilidade e eficiência do software DyNetVis.

Em resumo, este trabalho de conclusão de curso abordou e resolveu dois problemas fundamentais do software DyNetVis: o desempenho ao carregar redes complexas e o carregamento de redes com tempo final extenso. As melhorias implementadas tiveram um impacto positivo na usabilidade e desempenho do software, aprimorando sua funcionalidade e atendendo às necessidades dos usuários.

Referências

BARABÁSI, A. L. **Linked: How everything is connected to everything else and what it means for business, science and everyday life**. [S.l.]: Plume, 2003. Citado na página 7.

BARABÁSI, A.-L. **Network Science**. [S.l.]: Cambridge University Press, 2016. Citado na página 11.

BASTIAN, M.; HEYMANN, S.; JACOMY, M. Gephi: an open source software for exploring and manipulating networks. In: **Proceedings of the international AAAI conference on web and social media**. [S.l.: s.n.], 2009. v. 3, n. 1, p. 361–362. Citado na página 8.

GAMA, R.; SANTOS, A.; SILVA, M. Análise de desempenho de sistemas java utilizando o netbeans profiler. **Revista Científica do ITPAC**, v. 12, n. 1, p. 72–79, 2019. Citado na página 17.

KUMAR, S.; SPEZZANO, F.; SUBRAHMANIAN, V.; FALOUTSOS, C. Edge weight prediction in weighted signed networks. In: IEEE. **Data Mining (ICDM), 2016 IEEE 16th International Conference on**. [S.l.], 2016. p. 221–230. Citado na página 18.

KUMAR, S.; ZHANG, X.; LESKOVEC, J. Predicting dynamic embedding trajectory in temporal interaction networks. In: ACM. **Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining**. [S.l.], 2019. p. 1269–1278. Citado na página 19.

LINHARES, C. D. G.; PONCIANO, J. R.; PAIVA, J. G. S.; TRAVENÇOLO, B. A. N.; ROCHA, L. E. C. Visualisation of structure and processes on temporal networks. In: HOLME, P.; SARAMÄKI, J. (Ed.). **Computational Social Sciences**. Cham: Springer International Publishing, 2019. p. 83–105. ISBN 978-3-030-23495-9. Citado na página 13.

LINHARES, C. D. G.; PONCIANO, J. R.; PAIVA, J. G. S.; ROCHA, L. E. C.; TRAVENÇOLO, B. A. N. DyNetVis - an interactive software to visualize structure and epidemics on temporal networks. In: **2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)**. [S.l.]: IEEE, 2020. Citado 2 vezes nas páginas 7 e 13.

LINHARES, C. D. G.; TRAVENÇOLO, B. A. N.; PAIVA, J. G. S.; ROCHA, L. E. C. DyNetVis: a system for visualization of dynamic networks. In: **Proceedings of the Symposium on Applied Computing**. New York, NY, USA: ACM, 2017. (SAC'17), p. 187–194. ISBN 978-1-4503-4486-9. Citado 4 vezes nas páginas 7, 8, 11 e 13.

NEWMAN, M. The structure and function of complex networks. **SIAM Review**, v. 45, p. 167–256, 2003. Citado na página 11.

PANZARASA, P.; OPSAHL, T.; CARLEY, K. M. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. **Journal of the**

American Society for Information Science and Technology, Wiley, v. 60, n. 5, p. 911–932, 2009. Citado na página 18.

PONCIANO, J. R.; LINHARES, C. D.; MELO, S. L.; LIMA, L. V.; TRAVENÇOLO, B. A. Visual analysis of contact patterns in school environments. **Informatics in Education**, Vilnius University Institute of Data Science and Digital Technologies, v. 19, n. 3, p. 455–472, 2020. Citado na página 7.

PONCIANO, J. R.; LINHARES, C. D. G.; FARIA, E. R.; TRAVENÇOLO, B. A. N. An online and nonuniform timeslicing method for network visualisation. **Computers & Graphics**, Elsevier BV, v. 97, p. 170–182, jun 2021. Citado na página 14.

PONCIANO, J. R.; LINHARES, C. D. G.; ROCHA, L. E. C.; FARIA, E. R.; TRAVENÇOLO, B. A. N. A streaming edge sampling method for network visualization. **Knowledge and Information Systems**, Springer Science and Business Media LLC, v. 63, n. 7, p. 1717–1743, apr 2021. Citado na página 13.

PRESSMAN, R. S. **Engenharia de Software: Uma abordagem profissional**. 8ª. ed. [S.l.]: McGraw-Hill, 2015. 214 p. Citado na página 14.

RUFIANGE, S.; MCGUFFIN, M. J. Diffani: Visualizing dynamic graphs with a hybrid of difference maps and animation. **IEEE Transactions on Visualization and Computer Graphics**, IEEE, v. 19, n. 12, p. 2556–2565, 2013. Citado na página 8.

SHANNON, P.; MARKIEL, A.; OZIER, O.; BALIGA, N. S.; WANG, J. T.; RAMAGE, D.; AMIN, N.; SCHWIKOWSKI, B.; IDEKER, T. Cytoscape: a software environment for integrated models of biomolecular interaction networks. **Genome research**, Cold Spring Harbor Lab, v. 13, n. 11, p. 2498–2504, 2003. Citado na página 8.

SILVA, J.; FERREIRA, V.; SOUZA, L. Avaliação do uso da ide netbeans em um curso introdutório de programação. In: **Anais do Simpósio Brasileiro de Informática na Educação**. [S.l.: s.n.], 2018. p. 45–54. Citado na página 15.

SOMMERVILLE, I. **Engenharia de Software**. 10ª. ed. [S.l.]: Pearson Education, 2016. 275 p. Citado na página 14.

Apêndices

APÊNDICE A – Código Comparador

Código utilizado para testar e comparar o desempenho ao carregar a rede usando duas diferentes estruturas de dados.

```
1  public  class CarregaRede {
2
3      final private Scanner in;
4
5      public CarregaRede(Scanner in){
6          this.in = in;
7      }
8
9
10     // Simulando execucao da leitura do arquivo
11     public void TesteArray(){
12
13         ArrayList<ArrayList> matrizDataInline = new ArrayList<>()
               ;
14
15
16         // Percorrer cada linha do arquivo
17         while (in.hasNextLine()) {
18
19             // Array para nos e tempo da conexao
20             ArrayList<Integer> coluna = new ArrayList<>();
21             String line = in.nextLine();
22
23             System.out.println(line);
24             String[] tokens = line.split("[ \\t]");
25
26             //Verifica se linha nao e vazia, se e numero
27             //e se aresta nao liga o proprio no
28             if(!line.equals("")){
29                 if(!Character.isDigit(line.charAt(0))){
30                     continue;
31                 }else if(tokens[0].equals(tokens[1])){
32                     continue;
33                 }
34             }else{
```

```
35                  continue;
36              }
37
38              int no1 = Integer.parseInt(tokens[0]);
39              int no2 = Integer.parseInt(tokens[1]);
40              int tempo = Integer.parseInt(tokens[2]);
41
42              coluna.add(no1);
43              coluna.add(no2);
44              coluna.add(tempo);
45
46              if(!matrizDataInline.contains(coluna)){
47                  matrizDataInline.add(coluna);
48              }
49          }
50          for (ArrayList<Integer> x : matrizDataInline) {
51              System.out.println( x.get(0) + " " + x.get(1) + " " +
                    x.get(2));
52          }
53      }
54
55
56      public void TesteVector() {
57
58          ArrayList<int[]> matrizDataInline = new ArrayList<>();
59
60          int coluna[];
61
62          // Percorrer cada linha do arquivo
63          while (in.hasNextLine()) {
64
65              coluna = new int[3];
66              String line = in.nextLine();
67
68              String[] tokens = line.split("[ \\t]");
69
70              //Verifica se linha nq  aao e vazia, se e numero
71              //e se aresta nao liga o proprio no
72              if(!line.equals("")){
73                  if(!Character.isDigit(line.charAt(0))){
74                      continue;
75                  }else if(tokens[0].equals(tokens[1])){
```

```
76                      continue;
77                  }
78              }else{
79                  continue;
80              }
81
82              // Converte nos para inteiro
83              int no1 = Integer.parseInt(tokens[0]);
84              int no2 = Integer.parseInt(tokens[1]);
85              int tempo = Integer.parseInt(tokens[2]);
86
87
88              coluna[0] = no1;
89              coluna[1] = no2;
90              coluna[2] = tempo;
91
92              if(!containsElement(matrizDataInline, coluna)){
93                  matrizDataInline.add(coluna);
94              }
95          }
96
97          for (int[] x : matrizDataInline) {
98              System.out.println( x[0] + " " + x[1] + " " + x[2]);
99          }
100     }
101
102
103     private boolean containsElement(ArrayList<int[]> matrizdata,
            int[] element){
104          for (int[] i : matrizdata) {
105              if (i[0] == element[0] &&
106                  i[1] == element[1] &&
107                  i[2] == element[2]){
108                  return true;
109              }
110          }
111          return false;
112     }
113 }
114
115
116 class Main {
```

```
117
118       static public void main(String args[]) throws
              FileNotFoundException{
119
120           String file = "bitcoin.txt";
121           Scanner in;
122
123           in = new Scanner(new FileReader("C:\\Users\\bf__\\
                  Documents\\dadosTCC\\"+file));
124           CarregaRede cr = new CarregaRede(in);
125           cr.TesteArray();
126           cr.TesteVector();
127
128       }
129   }
```