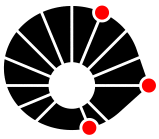
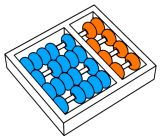




Visualização Interativa da Evolução de Grafos de Conhecimento

Eduardo Moreira Freitas de Souza *Julio C. dos Reis*

Relatório Técnico - IC-PFG-21-34
Projeto Final de Graduação
2021 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Visualização Interativa da Evolução de Grafos de Conhecimento

Eduardo Moreira Freitas de Souza, Julio Cesar dos Reis *

Resumo

Este trabalho estuda uma ferramenta para a visualização interativa da evolução de grafos de conhecimento. Implementamos o software *TKGEvolViewer*, uma ferramenta que possibilita a exploração gráfica de *Temporal Knowledge Graphs* (TKGs) a partir de valores de métricas codificadas nas estruturas dos TKGs. Nosso resultados permitem usuários conduzirem análises visuais de *TKGs* através de um modal gráfico que filtra informações disponíveis por meio de analises predefinidas. Nossa solução possibilita adicionalmente uma exploração avançada e livre da estrutura dos grafos de conhecimento.

1 Introdução

É possível analisar um conjunto de dados de diversas maneiras: vendo cada um de seus elementos em sua forma bruta, como células em uma planilha ou através de uma representação visual que sintetize alguns de seus aspectos mais importantes, como um histograma que mostre os componentes mais comuns ou a variação de algum valor em relação ao tempo. No entanto, como escolher tal representação de forma que satisfaça a necessidade do usuário enquanto aproveita ao máximo as propriedades da estrutura do material em questão?

Em cenários mais simples — como descobrir a tendência do custo de um produto — a representação gráfica do preço em relação ao tempo pode aparecer de maneira natural, enquanto que um cenário de visualização dos contatos de um usuário em uma rede social em relação ao tempo abre maior espaço para discussão acerca do objetivo da análise e qual a melhor representação da base de dados afim de que a necessidade em questão seja cumprida.

Para o exemplo supracitado, uma forma de estruturar os dados de um determinado momento seria com um grafo: usuários seriam representados como vértices e as suas conexões, como arestas; Ademais, haveriam várias versões deste grafo, representando essas conexões em diferentes momentos no tempo. Por conseguinte, não existiria apenas a questão de como representar os grafos em sua unidade, haveria também o caso de representar as suas mudanças a cada iteração destes, aumentando a complexidade do problema da visualização. Dada a natureza recente desse campo, é de se esperar que existam diferentes abordagens que analisam toda sorte de características dos grafos dinâmicos, porém não há uma visualização amigável consolidada disponível.

*Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP.

Este estudo visa propor e implementar meios de representar e explorar graficamente um grupo específico de grafos dinâmicos, conhecidos como *Temporal Knowledge Graphs*. Tratando-se de grafos dinâmicos — isto é, várias iterações de um conjunto de dados que possuem relações entre si, classificados, respectivamente, como vértices e arestas. Enquanto Grafos de Conhecimentos (*KG*) são compostos por triplas *RDF* (*Resource Description Framework*), na forma de sujeito—predicado—objeto (i.e. conceitos são ligados com relações, como em Fernando—visitou—Espanha), os Grafos de Conhecimento Temporais (*TKG*) são compostos por mais de uma iteração de um *KG*. Dessa forma, um *TKG* contém representações das triplas de um determinado domínio em diversos momentos.

Há diversas aplicações da visualização destes, como relatado por Beck *et al.* [1]: em mapeamento de interações entre proteínas na biologia, transações econômicas no mundo financeiro ou até mesmo performance de jogadores nos esportes. Rossanez *et al.* [2] aplicaram o conhecimento de visualização de grafos em diversas etapas da literatura biomédica sobre doenças degenerativas, através da geração de diversas ontologias — modelo de dados que representa um conjunto de conceitos e os relacionamentos entre eles.

Visamos melhorar a compreensão sobre a análise de TKGs através do desenvolvimento de uma ferramenta de software que, com o uso de técnicas conhecidas para representação de grafos, satisfaça as carências tanto de um usuário iniciante, utilizando-se de uma interface simples com componentes que guiam o fluxo de uso; quanto as de um usuário avançado, habilitando componentes mais complexos, que possibilitam tanto a exploração de métricas medidas anteriormente quanto a própria estrutura gráfica dos vértices.

O resultado obtido foi o *TKGEvolViewer*, uma ferramenta que permite a visualização interativa de um *TKG*, com seus *KGs* separados espacialmente em ordem cronológica — em formato de linha do tempo, representando seus nós (sujeitos e objetos) com círculos e suas arestas (predicados) com segmentos de retas e setas devidamente coloridos de acordo com qual iteração ele representa. Além disso, conta com um assistente de exploração que contém questões pré-preparadas que agem como guia na extração de informações para usuários que estão entrando em um primeiro contato tanto com a ferramenta quanto o conjunto de dados que será estudado; E, para usuários que queiram explorar as estruturas de maneira mais avançada, há suporte para exibição das métricas medidas pela ferramenta TKGAnalyser [3], desenvolvida por Rossanez, e possibilidade de livre exploração do grafo, visualizando a estrutura dos conceitos com o uso do mouse, através de *zoom-in*, *zoom-out* e translação na representação, além do etiquetamento e realce dos vértices, bem como de suas arestas.

A ferramenta se destaca na sua fácil capacidade de representação dos dados desejados, em comparação à baixa complexidade de implementá-la. A plataforma permite uma fácil exportação para os principais sistemas operacionais conhecidos: Windows, Linux, MacOS, iOS e Android, incluindo a possibilidade de ser acessada remotamente por um navegador, através da sua exportação para HTML5, caso seja hospedada em um servidor.

O restante deste documento está organizado da seguinte maneira: A seção 2, Revisão da Literatura, condensa casos de estudo importantes acerca do uso de Grafos de Conhecimento e da análise e técnicas de visualização de Grafos Dinâmicos. A seção 3, TKGEvolViewer, detalha a motivação, casos de uso, o funcionamento dos módulos internos da ferramenta e a base de dados utilizada. Por fim, a seção 4, Conclusão, apresenta a conclusão acerca da visualização de grafos dinâmicos na ferramenta proposta.

2 Revisão da Literatura

2.1 Uso de Grafos de Conhecimento

Iniciamos a revisão da literatura pelo estudo da criação do *dataset* utilizado nesse estudo. Rossanez e Dos Reis [4] propõem a implementação de uma ferramenta semiautomática de geração de *KGs* a partir da literatura científica de doenças degenerativas, como Alzheimer. Ele também aborda formalmente o processo de criação de um *KG*, assim como a definição de uma Ontologia.

A ferramenta proposta consiste de quatro módulos: um preprocessador, responsável por simplificar as orações do texto original e resolver abreviações e pronomes que serão lidas pelo extrator de triplas. Este extrai candidatos a sujeitos, predicados e objetos, além de ser responsável por reconhecer orações na voz passiva e identificar agentes ou pacientes possivelmente fora da oração, de forma que a tripla contenha o elemento original, e não uma referência a este. A partir do texto preprocessado, é gerada uma *parse tree* de cada oração, identificando os elementos desta com uma etiqueta de uso semântico e os ligando a um elemento de uma Ontologia. Por último, é gerado um arquivo *turtle* que contém um recurso para cada constituinte da tripla — bem como a sua ligação com a respectiva Ontologia — e o relacionamento entre cada elemento desta: o resultado é a descrição de um *KG*.

Aprimorando a ferramenta do artigo anterior, Rossanez et al. [5] comparam o resultado semiautomático com especialistas do campo extraindo as triplas *RDF* já supracitadas, e há um grande espaço para se discutir o que foi obtido. O KGen [6] se mostrou capaz de extrair pelo menos três vezes mais triplas que os pesquisadores da área, e isso se deve à adição de um resolutor de dependências, capaz de extrair triplas secundárias de um tripla primária, englobando recursos como subclasses de outros conceitos ou ligações de subtemas entre os elementos de um mesmo sujeito ou objeto. Ademais, apesar dele não ser capaz de extrair informações em uma frase cuja dependência gramatical entre os objetos de uma mesma frase não esteja clara e de acordo com o esperado, é possível inserir manualmente triplas após o processo automático, de forma que o *KG* fique mais completo.

Já Pomp *et al.* [7], preocupado com a manutenção de um grande armazenamento de dados, propõe um uso diferente para os Grafos de Conhecimentos, de forma que sugiram conceitos semânticos a atributos de dados inseridos em um sistema. Tal abordagem tem como objetivo identificar de forma automática e eficiente o tipo do dado inserido para que consiga manter um bom nível de organização em um sistema que mantém um grande número de dados armazenados, visando impedir o surgimento de um 'pântano de dados'.

Dessa forma, através de um classificador treinado para reconhecer e classificar a qual classe de dado a nova entrada pertence, faz-se o uso de um *KG* para que se armazene a escolha mais provável entre texto, identificador, *bag of words* e valor numérico. Assim, o tipo de dado é um nó que está ligado a cada classificação por uma aresta ponderada, que indica o resultado do modelo de treinamento para a base de dados proposta. Vale ressaltar que o problema dessa abordagem é a necessidade de treinar novamente o modelo caso um tipo de dado ou de classificação fosse inserido ou retirado do grafo, sendo inviável o seu uso para sistemas cujo os tipos de dados estão em constante mudança.

2.2 Análise de Grafos Dinâmicos

Para melhor entender a necessidade do usuário e saber quais propriedades dos grafos dinâmicos são oportunas de serem usadas no *software* desejado, é fundamental estudar os rumos das pesquisas nessa área.

Rossanez et al. [2], avançando com o o trabalho realizado em [5], propõe uma análise inovativa de estudo de centralidades acerca dos *TKGs* obtidos pelo KGen. Essas análises de centralidade são: *degree centrality*, *eigenvector centrality* e *betweenness centrality*. Através delas, é possível representar a mudança de conhecimento acerca de diferentes iterações de corpos textuais de um mesmo tema ou contexto, porém em tempos diferentes.

Enquanto isso, Tosi e dos Reis [8] aumentam o escopo da análise e propõem um estudo acerca da evolução de um campo científico a nível de conceito, descrevendo subárea e as suas relações, assim como os conceitos pertencentes a estas, bem como as relações entre si, e apresentam a ferramenta que os auxiliam, o SciKGraph. Esta ferramenta é responsável pela construção do *KG*, que conecta conceitos que se apresentam próximos no texto original com arestas ponderadas, que têm seu peso ajustado de acordo com o número de ocorrência dessas conexões. Assim, avaliando nós vizinhos que apresentam arestas de maior peso, é possível separar o grafo em diferentes núcleos (áreas).

Dessa forma, com as áreas bem definidas, é possível comparar duas iterações de um texto e avaliar a mudança do uso e de conexões entre os conceitos ao comparar os dois *KGs* gerados. Para isso, é criado o conceito de similaridade, que visa medir a porcentagem de nós iguais entre dois núcleos de diferentes grafos, tornando possível reconhecer quais áreas permaneceram sem grandes mudanças ou se houve alguma aglutinação ou separação destas. Reconhecendo as mudanças dos núcleos, torna-se analisável reconhecer quais áreas perderam ou ganharam uma conexão entre elas entre as duas iterações.

Através da organização do SciKGraph, é possível observar que conceitos mais gerais se encontram no centro do grafo, enquanto conceitos mais específicos se alinham mais às bordas — comportamento que se manteve na ferramenta obtida como resultado do estudo.

Também foram estudadas abordagens com um menor foco na utilização prática das alterações sofridas nos *KGs*: Liu et al. [9] propõe um modelo de Grafo derivado e uma ferramenta que analisa a evolução de um *TKG*, o EvolveKG. O trabalho adiciona uma *timestamp* à então tripla *RDF* (sujeito, predicado, objeto), de forma que distingua as relações de cada outra iteração, integrando cada elemento deste em um único grafo. Assim, torna possível realizar predições baseadas no comportamento que cada elemento tem apresentado à medida que evolui — sendo uma das medidas a aceleração da velocidade de crescimento do grafo: acelerada, constante ou desacelerada.

Já Pernischova et al. [10] propõe uma solução para analisar o impacto de uma operação em um *TKG*, de forma a poupar recursos para realizar uma atualização. Avaliando a somatória e a média entre as ocorrências de nós vizinhos entre duas iterações de um grafo, um modelo de aprendizado de máquina é gerado, analisando o impacto de uma operação de modificação no grafo, tal qual: `move`, `merge`, `split`, `add` e `delete`, em uma medida de centralidade, como *degree*, *degree centrality*, *closeness centrality* e *betweenness centrality*.

Ao transformar o *TKG* para um Grafo Derivado, criam-se arestas ponderadas também etiquetadas em relação ao tempo, para cada conexão no grafo original. Essas arestas repre-

sentam a importância da conexão naquele instante de tempo. Dessa forma, torna-se possível não só analisar a tendência de crescimento do Grafo Derivado e do original, como se torna possível prever também o comportamento do *TKG*, através de modelos de aprendizado de máquina — que apresentaram uma melhora em relação aos outros modelos de predição comparados no artigo.

2.3 Técnicas de Visualização de Grafos Dinâmicos

Buscando representações visuais para mostrar informações relevantes de um grafo, Rodrigues et al. [11] propõe uma abordagem de codificação gráfica para a visualização da evolução de diferentes métricas em um grafo. Na pesquisa, foi utilizado o exemplo de uma análise de um jogo de futebol. Dessa forma, para a criação do grafo, a posição dos jogadores no campo se tornaram os nós e a possibilidade de passo da bola para um jogador do mesmo time se tornaram as arestas. Isso feito para cada instante relevante do jogo, é calculada a entropia de cada nó, que tem uma codificação equivalente em cores — quanto maior a entropia, mais saturada é a cor, e esta é colocada em um elemento de uma nova coluna de gráfico, de forma que represente ordenadamente a entropia de cada nó do grafo em um instante de tempo.

No final do processo, repetindo para cada grafo gerado, obtém-se uma representação gráfica que codifica o valor de entropia para cada nó a cada instante do tempo, sendo possível, dentro do exemplo proposto, fazer uma análise de quanto um jogador foi pressionado por um oponente durante o jogo, tornando possível criar estratégias que aliviem o estresse da marcação em cima do jogador supracitado, por exemplo.

Beck *et al.* [1] realizaram uma meta-análise do estado da arte de representação de grafos dinâmicos estudando 162 publicações — com datas de 1992 a 2015, afim de sintetizar e categorizar as formas mais utilizadas, além de mostrar a visão de especialistas na área sobre elas. As maneiras de visualização se encontram dividas em duas categorias principais: as que utilizam algum recurso de animação para mostrar a mudança de cada passo do grafo de maneira fluida e as que utilizam um mapeamento de tempo para espaço para representar tais alterações, resultando em uma estrutura que se assemelha a uma linha do tempo.

É citado que há concordância entre os especialistas de que utilizar a abordagem animada facilita a assimilação do usuário acerca das mudanças aplicadas no grafo quando maior parte da estrutura deste se mantém estática (isto é, grande parte dos vértices não se movimenta). Essa interpelação se mostra uma barreira quando várias mudanças são aplicadas no grafo em um mesmo passo, visto que, além de grande parte da sua estrutura se movimentar, tudo é realizado ao mesmo tempo. Contudo, também foram aplicadas estratégias de forma a diminuir o impacto dessa dificuldade, como o agrupamento dos nós mais frequentemente vistos juntos em *clusters* e aplicando um guia visual extra de forma a identificar esse novo grupo, como um círculo ao redor dos vértices integrantes. Enquanto isso, para passos mais complexos ou mais numerosos, é preferido a abordagem de linha do tempo, como a integrada, na qual o tempo é mapeado diretamente do grafo, como adicionando a informação do tempo em uma codificação por cores nas arestas dos vértices, indicando em quais iterações essa tripla esteve presente, bem como a ordem das suas ocorrências.

Após serem apresentados algumas questões sobre o tema, os 16 especialistas que res-

ponderam apresentaram alguma convergência de opiniões, como o melhor proveito no uso de estratégias animadas para apresentações e para um primeiro contato com usuários que não conhecem a área; o melhor proveito na utilização de estrategias de linha do tempo para explorações interativas de um *dataset*; e a importância da estratégia de visualização ser escalável para um grande número de dados, enquanto mantém o seu aspecto interativo.

3 TKGEvolViewer

A ferramenta foi desenvolvida com performance em mente: como o número de vértices em diversas iterações de um grafo pode alcançar facilmente a casa de dezenas de milhares, foi escolhida uma plataforma enxuta e de fácil utilização (a *Godot Game Engine*[12]) como base para a ferramenta proposta, bem como bibliotecas adicionais otimizadas para lidar com uma grande quantidade de dados e operações complexas, como a *igraph* [13], a *boost* [14] e a *annoy* [15].

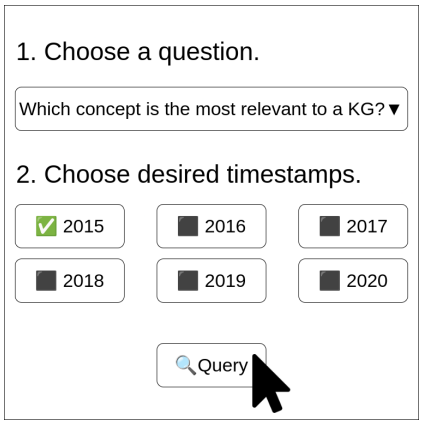
Além disso, foi pensado em um fluxo simples e objetivo de obtenção de informações, como descrito no *Storyboard* da figura 1. Ao carregar o conjunto de dados desejado, o usuário seria apresentado a uma tela (figura 1a) com um conjunto pré-selecionado de perguntas-guia — de forma a direcionar o usuário novato acerca das informações que podem ser obtidas de forma mais direta — e de um conjunto de anos disponíveis nos dados de entrada. Em seguida, ele seria levado para um ambiente que destaca os conceitos que respondem a pergunta-guia, bem como outros conceitos que o interessem, através da exploração interativa dos vértices apresentados com o mouse (figura 1b). Ademais, após se sentir satisfeito explorando a resposta dada, bem como os conceitos que o usuário considerou relevante através do travamento deles na interface (figura 1c), ele acessaria o menu lateral, sempre disponível, que apresentaria a mesma interface da primeira interação, disponibilizando ao usuário a possibilidade de uma nova busca, possibilitando alterar tanto a questão-guia como a janela de tempo desejada (figura 1d), repetindo o fluxo.

Com o objetivo de atender à necessidade de exploração de *TKGs* de usuários de diversos níveis de experiência de uma maneira intuitiva, foi-se obtido o *software* nomeado de TKGEvolViewer [16]: uma plataforma para a representação gráfica e interativa dos resultados obtidos pelo TKGAnalyser [3]. Através de elementos gráficos conhecidos, apresenta um visual minimalista e módulos de auxílio para a obtenção de informações, busca ser uma alternativa amigável para desde uma pesquisa simples a uma exploração mais completa dos conjuntos de dados, utilizando estruturas formadas de triplas na configuração de conceito — relação — conceito para a sua representação gráfica, podendo possuir diversas iterações destes dados, com diferentes *timestamps* em cada *dataset*.

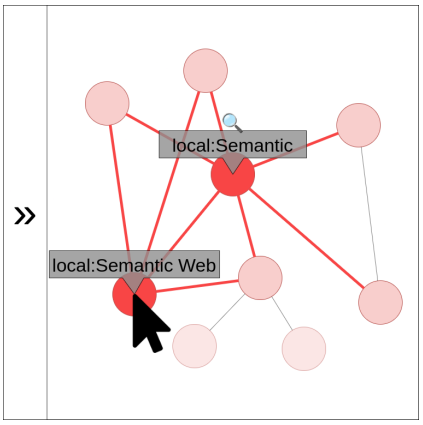
O fluxo para início da exploração dos grafos é simples e intuitiva, moldado a partir do *Storyboard* da figura 1 e melhor detalhado no Apêndice A.

3.1 Funcionalidades principais

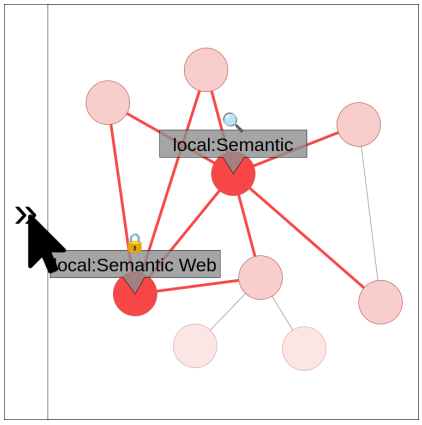
Através da representação da figura 2, é possível apresentar três elementos-chave na representação gráfica do *dataset* usado: a barra lateral, separada em dois elementos, nomeados — de cima para baixo — de **Wizard** e **Metrics Tree**. O primeiro possui as mesmas



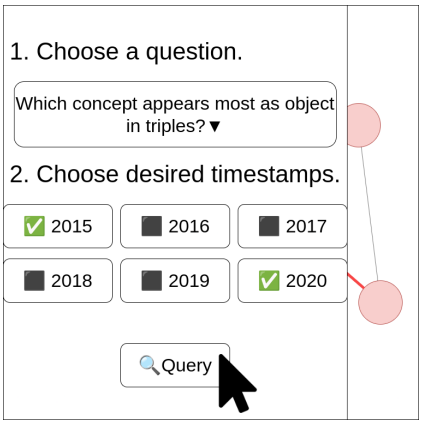
(a) Primeiro conjunto de ações do usuário no fluxo, selecionando a questão-guia 'Which concept is the most relevant to a KG?' dentre as disponíveis, bem como a janela de tempo do ano de 2015.



(b) Segundo conjunto de ações do usuário no fluxo, explorando a resposta dada à pergunta escolhida (no caso, o conceito 'local: Semantic'), com o destaque de um elemento de interesse através da aproximação do mouse, o conceito 'local: Semantic Web'.



(c) Terceiro conjunto de ações do usuário no fluxo, selecionando o menu lateral após realizar o travamento do conceito 'local: Semantic Web' na visualização do grafo.



(d) Quarto conjunto de ações do usuário no fluxo, redefinindo a pergunta-guia para 'Which concept appears most as object in triples?' e selecionando os *timestamps* de 2015 e 2020, repetindo o fluxo descrito.

Figura 1: *Storyboard* do fluxo de uso básico da ferramenta.

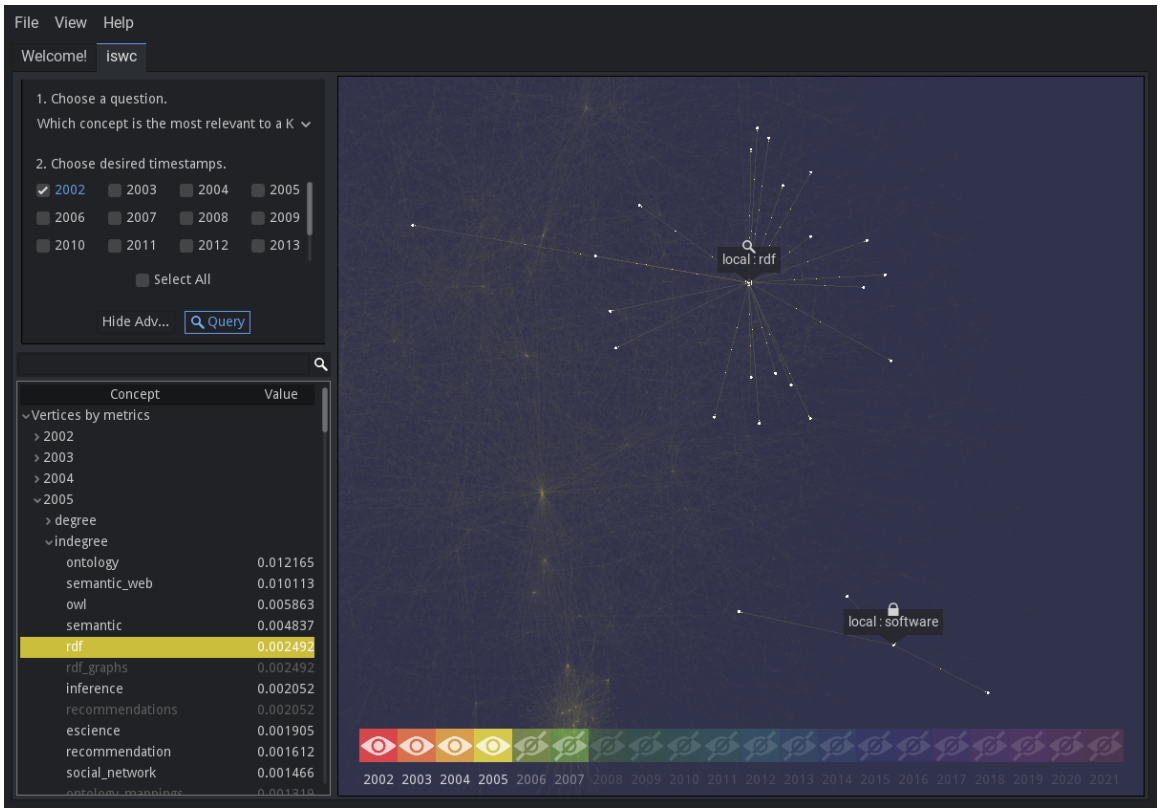


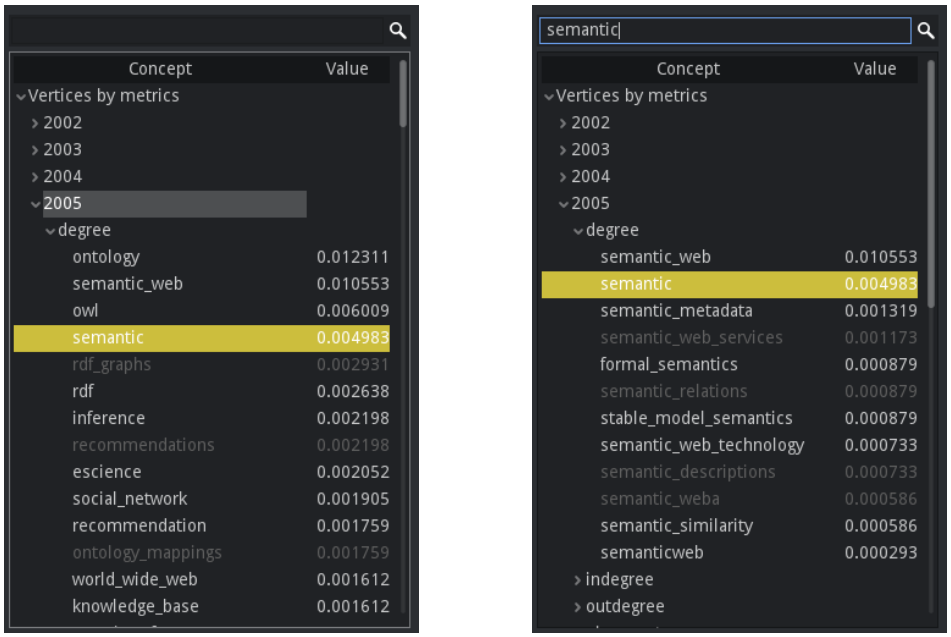
Figura 2: Exemplo de execução do TKGEvolViewer, com um enfoque na representação do Grafo de Conhecimento do conjunto de dados 'iswc' na iteração do ano de 2005, com os conceitos 'rdf' e 'software' em destaque. À esquerda observa-se um modal com uma seleção de questões, uma tabela com os anos disponíveis naquele conjunto de dados e dois botões: 'Hide Adv', e 'Query'. Abaixo, há uma lista em formato de árvore que contém os dados de centralidades medidos previamente para cada um dos vértices relevantes, ordenados por *timestamp* e por métricas. À direita, abaixo do grafo, uma linha do tempo serve como referência para relacionar os *timestamps* com as cores empregadas em cada iteração deste, bem como ícones indicando a visibilidade atual de cada uma das iterações.

funcionalidades que foram apresentadas no fluxo de uso básico do programa (figura 14), possibilitando a busca mesmo após alguma outra ser concluída e, o segundo, apresentado com maior foco nas figuras 3a e 3b será explicado abaixo em maiores detalhes; Por fim, há a **janela de visualização do grafo**, exibida anteriormente também na figura 15, com a linha do tempo indicando quais iterações estão sendo apresentadas, bem como a cor de referência usada para colorir os seus elementos da iteração em questão.

O **Wizard** é composto por dois seletores: o primeiro apresenta para o usuário as opções disponíveis de perguntas pré-fabricadas em um modelo de *dropdown* que o guiará para os conceitos que as satisfaçam; E, o segundo, composto por botões selecionáveis, etiquetados a partir do conjunto dos *timestamps* disponíveis no *TKG* fornecido, ordenados em várias linhas e colunas em um modal com rolagem vertical — de forma que exista um limite de espaço para ser ocupado por este, caso sejam fornecidas muitas iterações de um *dataset* —

seguidos pelo seletor 'Select All', facilitando a seleção/limpeza de todos os botões.

Já na sua parte de baixo, há um conjunto com três botões: 'Explore', disponível somente durante o processo da figura 14, visto que a sua função é de ignorar a seleção da pergunta e dos *timestamps* desejados, carregando todas as iterações do Grafo de Conhecimento Temporal, tornando tudo disponível para o usuário assim que todas as estruturas necessárias forem calculadas. O segundo botão, disponível somente após a transição para a página principal, alterna entre 'Advanced' (figura 15) e 'Hide Adv' (figura 2), controlando a visibilidade da **Metrics Tree** de acordo com a necessidade do usuário. E o terceiro e último botão disponível, 'Query', executa a requisição do usuário para a pergunta e *timestamps* selecionados, tornando visível apenas as iterações desejadas e realizando o realce dos vértices que satisfazem a questão.



(a) **Metrics Tree** apresentando a barra de busca sem qualquer entrada, não realizando a filtragem de seus elementos.

(b) **Metrics Tree**, apresenta a barra de busca com o conteúdo 'semantic', filtrando os conceitos para que apenas os que contenham tal palavra em seu nome sejam exibidos.

Figura 3: **Metrics Tree** com o conceito 'semantic' na métrica 'degree' no ano de 2005 selecionado, indicado pela colorização do fundo de acordo com a cor correspondente à iteração a qual pertence. Os itens que estão representados com uma cor mais escurecida significam conceitos que não podem ser selecionados, em oposição aos que têm uma cor mais próxima ao branco, que podem.

A **Metrics Tree** tem três principais funcionalidades: armazenar e organizar os principais conceitos de acordo com a iteração do grafo (ano do artigo, identificado pela sua *timestamp*) e as diferentes métricas aplicadas no TKGAnalyser, fornecidas pelo arquivo de entrada. Quando um ano é selecionado, é apresentado logo abaixo — em uma estrutura de listas aninhadas — as métricas disponíveis para aquele ano. Após o usuário selecionar uma

das métricas, é apresentada, então a lista dos conceitos com maior valor na métrica selecionada, ordenados em ordem decrescente de valor. Tais conceitos podem estar representados de três formas: A primeira, com um fundo colorido, significa que é um vértice que está em destaque (seja pela posição do cursor do mouse ou por ter sido um conceito 'travado' ou resultado da pesquisa pelo **Wizard** ou seleção pela própria **Metrics Tree**); A segunda, com o fundo escuro e uma fonte com coloração próxima do branco representa um conceito que não está selecionado atualmente; E a terceira, cujo conceito é representado com uma cor mais escurecido em relação ao anterior, indicando que, apesar do conceito estar na lista de vértices medidos pelo arquivo de entrada, não foi encontrado no grafo carregado através do *link* neste fornecido, e, portanto, não se mostra disponível para seleção por quaisquer um dos métodos.

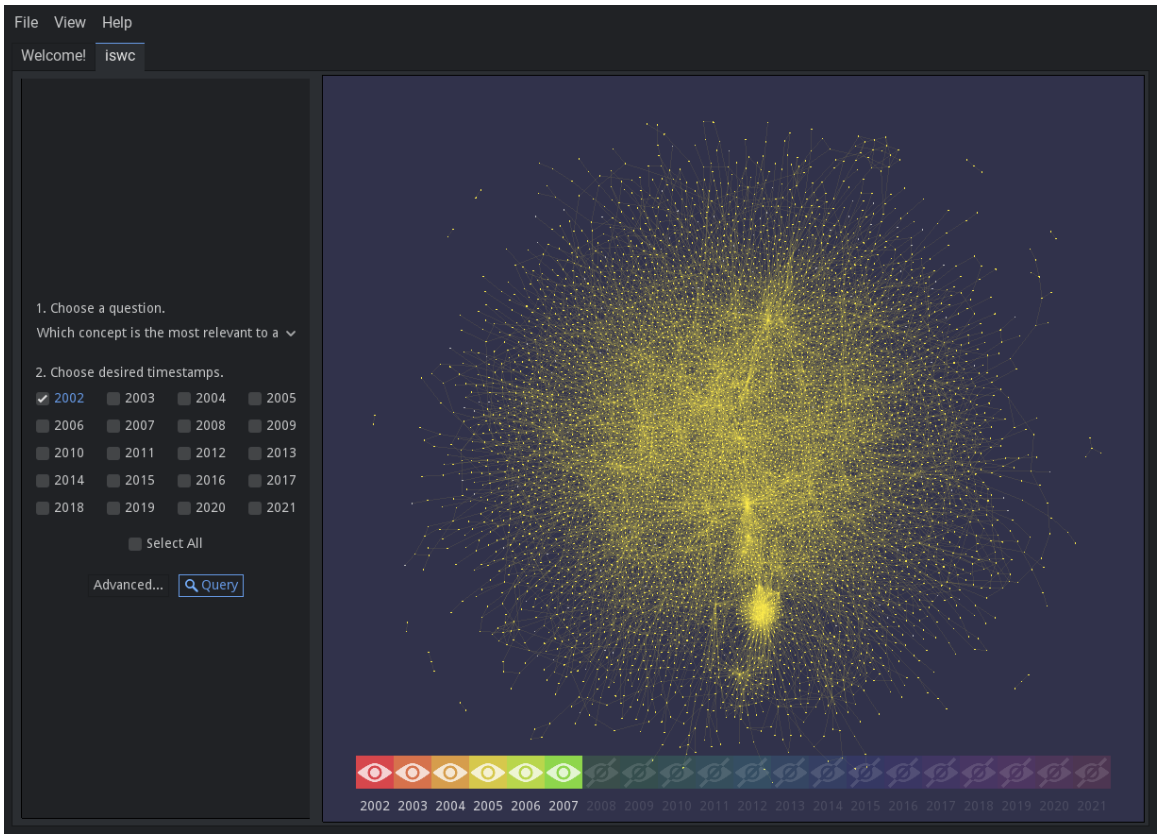


Figura 4: Representação gráfica completa do *KG* de '2005', sem foco em quaisquer vértices, mantendo opacidade de todos os elementos.

A segunda funcionalidade é a seleção de vértices: ao clicar em um conceito, a respectiva linha é destacada, alterando a sua cor de fundo para a cor designada ao *KG* no qual o vértice pertence; A visão presente na **janela de visualização do grafo** se desloca para mostrar o conceito selecionado no centro da tela e as suas relações com outros vértices ganham um destaque, ganhando mais tonalidade e alternando entre branco e sua cor original, além de ser apresentada uma etiqueta com seu nome, indicando a sua localização, como é possível

observar na figura 15. Já a sua terceira funcionalidade é a possibilidade de filtragem dos conceitos em sua apresentação, de forma que seja mais fácil para o usuário verificar a existência e pontuação de um conceito desejado, processo representado pelas figuras 3a e 3b, através da busca por conceitos que contém a palavra 'semantic' em seu nome.

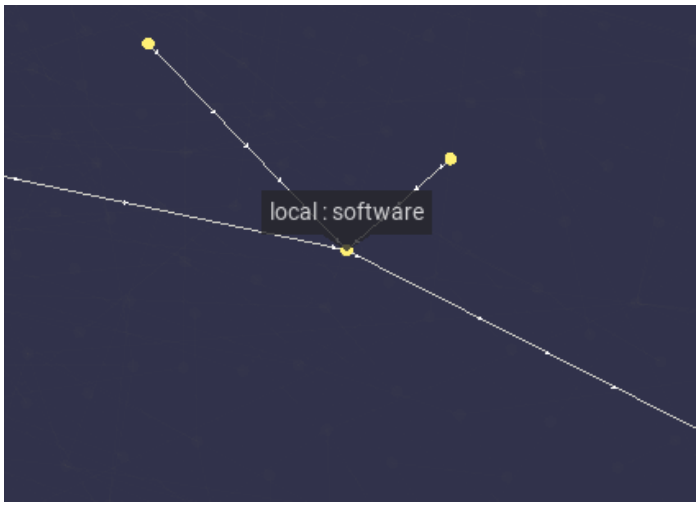


Figura 5: Representação gráfica do elemento 'software' do *KG* de '2005' quando o cursor do mouse se encontra próximo ao centro da sua representação, colocando ele e suas conexões em destaque, etiquetando a sua localização com o seu nome e tornando transparente o restante dos elementos do grafo.

Já a **janela de visualização do grafo** apresenta a representação interativa dos dados fornecidos aos programa na forma das várias iterações do *TKG* alinhadas em ordem cronológica, em uma reta. Ademais, é possível se movimentar no ambiente virtual dos grafos através do mouse, movendo-se no sentido do plano ao clicar e arrastar o botão esquerdo, assim como fazer *zoom-in* e *zoom-out* usando a roda do mouse para cima e para baixo, respectivamente.

Utilizando-se dessa movimentação, quando a visualização fica próxima o suficiente, é habilitada a etiquetagem interativa dos vértices do grafo focado, passando a opacidade dos outros conceitos de total para transparente, como é possível ver nas figuras 4 e 5. Nesse estado é habilitado o realce dinâmico das conexões de um vértice, alternando as cores das arestas e dos vértices alternadamente para branco, além de mostrar o conceito referente como selecionado na **Metrics Tree**, representado na figura 3a.

Com o vértice próximo ao mouse, é possível realizar a ação de 'travar' para torná-lo persistente, isto é, sua etiqueta não será removida caso o mouse saia dos seus arredores. Esse estado mantém o realce das conexões e da sua entrada na **Metrics Tree** e é representado pelo ícone de cadeado acima do nome do conceito, representado na figura 6. Há um outro meio de tornar a etiqueta persistente: a seleção automática do conceito pelo **Wizard** ou a seleção manual direto na **Metrics Tree**, representado na figura 7, com o símbolo de uma lupa no topo da etiqueta.

Por fim, há na parte inferior da **janela de visualização do grafo** uma indicação

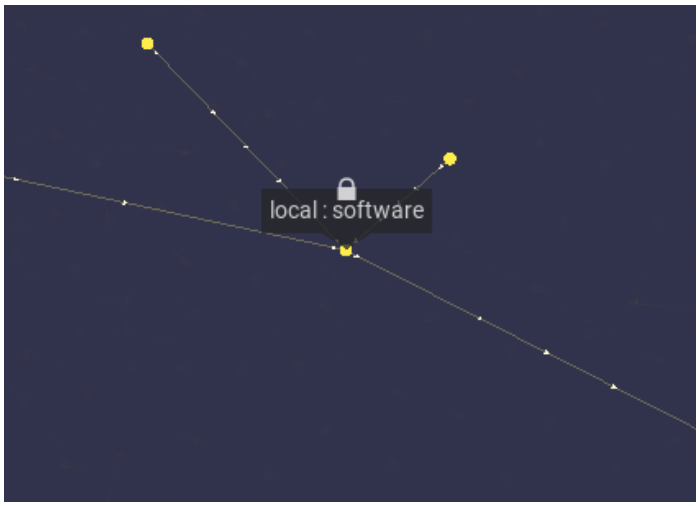


Figura 6: Representação gráfica do elemento 'software' do *KG* de '2005' após o cursor do mouse executar a ação de 'travar' próximo ao centro da sua representação, apresentando um símbolo de cadeado, tornando persistente ele e suas conexões em destaque, assim como sua etiqueta e a transparência do restante dos elementos do grafo.

acerca da relação entre a iteração e a cor que a corresponde, conectadas pela mesma posição horizontal, como a cor vermelha acima da *timestamp* '2002' na figura 8. Há também um ícone indicando a visibilidade atual da iteração correspondente: visível, caso apresente o ícone do olho ou invisível, caso apresente o ícone do olho cortado por uma semirreta.

Relacionado com a representação da visibilidade do *KG*, é possível observar também que na figura 8 há três níveis de transparência dos seus elementos: De 2002 a 2005 os elementos se apresentam completamente opacos e com o ícone do olho, indicando que os grafos estão disponíveis e atualmente visíveis; De 2006 a 2007 os elementos se mostram levemente transparentes, dessa vez com o ícone do olho cortado, indicando que os grafos correspondentes não estão visíveis — porém podem ser rapidamente exibidos; E, por fim, de 2008 a 2021, que apresentam também o ícone do olho cortado, entretanto, com uma transparência maior que o conjunto anterior, indicando que os grafos correspondentes não estão visíveis e precisarão ter a posição dos seus elementos calculadas para serem exibidos.

É possível alternar esses estados interagindo com o *timestamp* desejado ao clicar com o botão esquerdo do mouse neles: caso o grafo esteja visível, ele será escondido, juntamente com as suas etiquetas persistentes. Caso ele esteja invisível e disponível, ele será prontamente apresentado, acompanhado de suas etiquetas, caso possua alguma; Caso contrário, será necessário esperar o cálculo sob demanda destes finalizar. A última funcionalidade deste modal é a centralização da **janela de visualização do grafo** para um grafo, caso o ponteiro do mouse permaneça meio segundo pairando acima de um ícone de olho — ou seja, represente um *KG* visível e disponível.

Figura 7: Representação gráfica do elemento 'software' do *KG* de '2005' quando selecionado diretamente da sua entrada na **Metrics Tree** ou automaticamente através do **Wizard** (nesse caso, a sua entrada na **Metrics Tree** é automaticamente colorida também). Apresenta um símbolo de lupa e mantém persistente a sua etiqueta, assim como o destaque do elemento e das suas conexões, mantendo também a transparência do restante dos elementos do grafo.

Figura 8: Modal posicionado na parte inferior da **janela de visualização do grafo**, que relaciona uma iteração do *TKG* com a cor correspondente, utilizada para a coloração dos seus elementos no espaço. Há também a indicação da visibilidade e disponibilidade do grafo, indicado pelo ícone do olho, caso esteja visível. O ícone do olho cortado indica que o grafo não está sendo exibido e a alta intensidade da sua transparência aponta a necessidade de cálculo das posições dos elementos de um *KG*.

3.2 Módulos principais

O TKGEvolViewer é composto por quatro módulos principais, ambientados de forma gráfica dentre os outros módulos na figura 9:

1. **GUI (Graphical User Interface):** responsável pela interação entre o usuário e o programa e exibição dos resultados, assim como a ativação dos módulos responsáveis pelas outras partes do programa, como a exibição da **Metrics Tree** e o posicionamento das etiquetas na tela na posição correta.

2. **TKGFactory:** responsável por ler o arquivo de saída do TKGAnalyser, recuperar todos os elementos das iterações do *TKG* fornecido, assim como iniciar o módulo de integração com as bibliotecas externas.

3. **GDNTKG (Godot Native TKG):** ao receber as informações dos grafos preparados

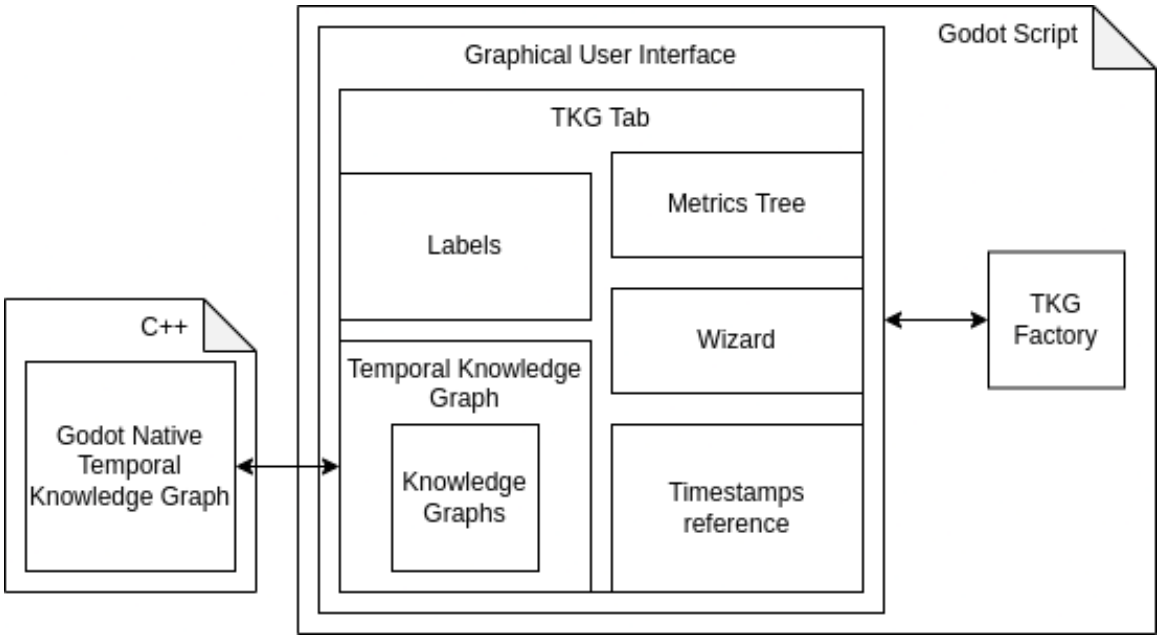


Figura 9: Diagrama arquitetural dos módulos presentes no projeto. À esquerda, está presente o módulo feito na linguagem C++, o **Godot Native Temporal Knowledge Graph**, responsável por integrar a funcionalidade das bibliotecas *igraph*, *boost* e *annoy* e se comunicar com o **Temporal Knowledge Graph**, presente no ambiente do Godot. À direita, se apresentam os módulos criados dentro do ambiente do Godot Script: o **TKG Factory** cria e fornece as novas estruturas de dados para a interface (**Graphical User Interface**), que, por sua vez, detém os módulos **Wizard**, **TKG Tab**, **Metrics Tree**, **Timestamps reference**, **Labels**, **Timestamps reference**, assim como a própria estrutura do **Temporal Knowledge Graph**, com seus respectivos **Knowledge Graphs**.

pelo TKGFactory, executa os algoritmos de posicionamento de vértices em grafos do *igraph* e os adapta para a visualização na GUI, espaçando-os de acordo com o tamanho disponível.

4. **TKG e KG:** após a criação dos grafos pelo GDNTKG, as informações do *TKG* e dos *KGs*, respectivamente, são armazenados neles, servindo de intermédio para a maioria das operações futuras que envolvam a interação do usuário com o grafo.

De forma a complementar o diagrama da figura 9, é possível notar que há vários níveis de contenção entre os módulos, representado em um esquema de nós em uma árvore: **TKG Tab** é o nó pai dos módulos contido dentro do seu retângulo, que, por sua vez, está contido — ou seja, é filho — da **Graphical User Interface**. Dessa forma, é possível encapsular todos os objetos necessários dentro de um mesmo módulo, de forma que fique reaproveitável e permita que sejam exibidos diversos *datasets* de forma simultânea, com diversos **TKG Tabs** organizado por abas dentro da **GUI**.

Essa estrutura também permite o salvamento de um determinado ramo da árvore (por exemplo, uma aba específica) para uso posterior em um arquivo, não necessitando que o usuário repita os passos realizados até o cenário que já havia chegado, além de tornar sua experiência com a ferramenta algo compartilhável entre usuários desta, facilitando a troca de informações e incentivando discussões.

3.3 Tecnologias utilizadas

A base escolhida para o desenvolvimento da ferramenta foi uma *game engine* chamada *Godot* [12] graças à sua alta capacidade de gerar elementos gráficos com pouca complexidade de implementação: elementos gráficos podem ser facilmente criados através da interface gráfica e possui uma linguagem de *script* própria chamada GDScript, que possui fortes semelhanças com Python, devido à sua simplicidade e diversos níveis de abstração. Outra funcionalidade do *Godot* que facilita a criação de elementos é a sua estruturação em árvore, que permite agrupar de forma hierárquica e organizada módulos necessários para determinada função da aplicação, muitas vezes especificadas pelo seu tipo de nó, gerando elementos reaproveitáveis em diversos cenários.

Outro ponto decisivo na adoção do *Godot* é a fácil integração de bibliotecas externas através da criação de classes especiais (nativas) em código C ou C++, como foi feita com a **GDNTKG**. Dessa maneira, foi possível integrar os algoritmos de posicionamento de vértices da biblioteca em C do *igraph* [13] expondo certos métodos para a camada que utiliza o GDScript, de forma que os valores calculados pudessem não só ser traduzidos para o ambiente de visualização de grafos do programa através da classe nativa supracitada como também pudessem ser facilmente acessados pela camada de maior abstração.

Adicionalmente, para otimizar o tempo de cálculo de diversas iterações de um *TKG*, foi utilizado a biblioteca de concorrência do *boost* [14] em conjunto com a detecção do sistema atual do Godot. Assim, foi possível configurar de forma simples uma execução em paralelo dos cálculos de cada um dos grafos, melhor aproveitando a capacidade dos sistemas atuais enquanto respeita o uso do usuário com a sua máquina, alocando no máximo metade dos seus núcleos para os cálculos necessários e, consequentemente gerando um resultado

em um tempo menor na maior parte das vezes, em comparação caso fosse completamente sequencial.

Entretanto, o tempo de cálculo das posições dos vértices não foi o único obstáculo na performance: o número de conceitos e relações se mostrou um limitante na hora da exibição do resultado de maneira convencional, apresentando uma taxa de quadros que tornava qualquer interação com a ferramenta inviável. Foi necessário utilizar os recursos do *Godot* como *Game Engine* e usar instruções específicas que utilizam a placa gráfica do computador — especializada em várias tarefas simultâneas — para apresentar as dezenas de milhares de conceitos e relações entre eles da maneira leve como o TKGEvolViewer se mostra no seu estado atual.

Ademais, para a detecção da posição do cursor do mouse em relação aos grafos, a biblioteca *annoy* [15] foi utilizada: ela é capaz de indexar pontos no espaço (nesse caso, os vértices) com um baixo uso de memória, de forma a acelerar a busca pelo elemento que mais se aproxima do ponto dado (a posição do mouse), tornando incrivelmente responsiva a criação das etiquetas para a identificação dos em tempo real.

3.4 Base de Dados Utilizada

Os dados utilizados para o desenvolvimento da ferramenta advêm do processamento de textos da literatura científica biomédica através de duas ferramentas: o KGen[6] e o TKGAnalyser[3].

A primeira é responsável por fornecer as informações de forma que seja possível construir um Grafo de Conhecimento a partir de um texto em linguagem natural, realizando o preprocessamento, extração de triplas e ligação a elementos de ontologias deste, produzindo os seguintes tipos de elementos:

- *Statements*: Afirmações enumeradas, compartimentalizando as triplas do texto, representadas como:

```
local:s2002.82.1 a rdf:Statement ;
    rdfs:label "the two areas meet today" .
```

- *Classes*: Definição dos sujeitos e objetos da triplas, representadas como:

```
local:two_area a rdf:Class ;
    rdfs:label "two area" .
```

- *Mapped Relations*: Ligação de classes locais a conceitos já existentes em ontologias, representados como:

```
local:world_wide_web owl:sameAs cso:world_wide_web .
```

- *Relations*: Ligação de classes e afirmações através de mapeamentos locais e já existentes em ontologias, representados como:

```
local:two_area rdfs:subClassOf local:area .
local:area_meet_today rdfs:subClassOf local:today .
local:area_meet_today rdfs:member local:area_meet .
local:s2002.82.1 rdf:subject local:two_area .
local:s2002.82.1 local:AM-TMP local:today .
local:s2002.82.1 rdf:predicate local:meet .
```

A segunda é responsável por analisar as métricas dos vértices criados pela ferramenta anterior, ordenando os conceitos locais de interesse em várias listas, gerando um arquivo de acordo com a seguinte estrutura JSON:

```
[
  {
    "year": "2002",
    "tkg": "https://raw.githubusercontent.com/.../2002/merged_kg.ttl",
    "metrics": [
     {
        "metric": "degree",
        "concepts": [
          { "http://local/local.owl#semantic_web": 0.02578361981799798 },
          { "http://local/local.owl#ontology": 0.008088978766430738 }, ... ]
    },
    {
        "metric": "indegree",
        "concepts": [ ... ]
    }, ...
  },
  {
    "year": "2003",
    "tkg": "https://raw.githubusercontent.com/.../2003/merged_kg.ttl",
    "metrics": [ ... ]
  }, ...
]
```

Dessa forma, é necessária somente a saída do TKGAnalyser, visto que, para cada iteração do processamento dos textos da literatura biomédica, há uma referência online para o arquivo gerado pelo KGen.

Entre os anos de 2002 e 2021, a base usada contém mais de 26 mil *Statements*, cerca de 89 mil *Classes*, aproximadamente 2 mil *Mapped Relations* e mais de 230 mil *Relations*, além de contar com 12 métricas de centralidade disponíveis para análise.

4 Conclusão

A visualização de métricas de grafos dinâmicos ainda é um problema em aberto, com algumas abordagens estudadas durante o processo de aprendizado do assunto, como a representação do tempo em um mapeamento espacial ou em forma de animação; Ou a codificação visual em um esquema de mapa de calor. Por isso, é importante ter uma base sólida que permita armazenar o grafo de forma completa enquanto mantém a flexibilidade da representação, não necessitando realizar grandes mudanças como utilizar diversas linguagens de programação ou incorporar diferentes bibliotecas de auxílio visual para mudar a sua representação.

Portanto, a ferramenta proposta apresentou uma base sólida que supre as carências do usuário iniciante através do modal **Wizard**, apresentando um caminho claro e guiado acerca das capacidades do TKGEvolViewer, disponibilizando perguntas selecionadas especificamente para o tema de evolução de *Knowledge Graphs*. Ademais, também satisfaz o usuário mais experiente, possibilitando que ele interaja com os grafos de uma maneira exploratória, podendo relacionar através da **Metrics Tree** a pontuação dos conceitos nas métricas medidas com a posição e suas conexões nas iterações desejadas, controladas pelo modal de referência e visibilidade na **janela de visualização do grafo**.

E, finalmente, dada a maleabilidade e modularidade da plataforma utilizada, é simples a adição de módulos que auxiliem a experimentação de novos métodos de obtenção de informações, assim como representações adicionais, graças ao seu foco de natureza gráfica.

Referências

[1] BECK, Fabian; BURCH, Michael; DIEHL, Stephan; WEISKOPF, Daniel. A Taxonomy and Survey of Dynamic Graph Visualization. **COMPUTER GRAPHICS Forum**. Volume 36. 2017. p. 133-159.

[2] ROSSANEZ, Anderson; DOS REIS, Julio; TORRES, Ricardo Da Silva. Representing Scientific Literature Evolution via Temporal Knowledge Graphs. In proceedings of the 6th Workshop on Managing the Evolution Preservation of the Data Web (MEPDaW), co-located with the 19th International SemanticWeb Conference (ISWC).**MEPDaW**. 2020. p. 33-42.

[3] ROSSANEZ, Anderson. **TKGAnalyser**. Gitlab, 2021. Python. Disponível em: <https://gitlab.ic.unicamp.br/ra124136/TKGAnalyser>. Acesso em: Julho de 2021.

[4] ROSSANEZ, Anderson; DOS REIS, Julio Cesar. Generating Knowledge Graphs from Scientific Literature of Degenerative Diseases. In Proceedings of the 4th International Workshop on Semantics-Powered Data Mining and Analytics (SEPDA 2019), co-located with the 18th International Semantic Web Conference (ISWC 2019). **Aachen: CEUR Workshop Proceedings**. 2019. v. 2427. p. 12-23.

[5] ROSSANEZ, Anderson; DOS REIS, Julio Cesar; TORRES, Ricardo da Silva; DE RIBAUPIERRE, Hélène. KGen: a knowledge graph generator from biomedical scientific literature. **BMC Medical Informatics and Decision Making**, v. 20, p. 314, 2020.

[6] ROSSANEZ, Anderson. **Kgen**. Github, 2021. Python. Disponível em: <https://github.com/rossanez/KGen>. Acesso em: Setembro de 2020.

[7] POMP, André; KRAUS, Vadim; POTH, Lucian; MEISEN, Tobias. Semantic Concept Recommendation for Continuously Evolving Knowledge Graphs. In ICEIS (Revised Selected Papers). Lecture Notes in Business Information Processing. **International Conference on Enterprise Information Systems**. Springer, Cham, 2019. vol. 378. p. 361-385.

[8] DALLE LUCCA TOSI, Mauro; DOS REIS, Julio Cesar. Understanding the evolution of a scientific field by clustering and visualizing knowledge graphs. **Journal of Information Science**, p. 0165551520937915, 2020.

[9] LIU, Jiaqi; ZHANG, Qin; FU, Luoyi; WANG, Xinbing. Evolving Knowledge Graphs. **IEEE INFOCOM 2019-IEEE Conference on Computer Communications**. IEEE, 2019. p. 2260-2268.

[10] PERNISCHOVÁ, Romana; DEEL'AGLIO, Daniele; HORRIDGE, Matthiew; BAUMGARTNER, Matthias. Toward Predicting Impact of Changes in Evolving Knowledge Graphs. In International Semantic Web Conference, Auckland, New Zealand. **ISWC Satellites 2019**. CEUR Workshop Proceedings, vol. 2456, p. 137-140.

[11] RODRIGUES, Daniele C. Uchoa Maia; MOURA, Felipe A.; CUNHA, Sergio Augusto; TORRES, Ricardo da Silva. Graph visual rhythms in temporal network analyses. **Graphical Models**, v. 103, p. 101021, 2019.

[12] Godot Engine. **Godot**. Versão 3.4. Página inicial. C++. Disponível em: <https://godotengine.org/>. Acesso em: Dez de 2020.

[13] Igraph - The network analysis package. **igraph**. Versão 0.9. Página inicial. C. Disponível em: <https://igraph.org/>. Acesso em: Jul de 2021.

[14] Boost C++ libraries. **boost**. Versão 1.77. Página inicial. C++. Disponível em: <http://boost.org/>. Acesso em: Jul de 2021.

[15] Annoy (Approximate Nearest Neighbors Oh Yeah). **annoy**. Versão 1.17. Github, 2021. C++, Python, Lua, Go, C. Disponível em: <https://github.com/spotify/annoy>. Acesso em: Nov de 2021.

[16] TKGEvolViewer. **TKGEvolViewer**. Gitlab, 2021. C++, GDScript. Disponível em: <https://gitlab.ic.unicamp.br/ra124136/TKGEvolViewer>. Acesso em: Nov de 2021.

Apêndice A Fluxo de uso real da interface implementada

Ao abrir a aplicação, o usuário vê uma aba padrão que, através de um texto predefinido o contextualiza com os outros projetos da área, disponibilizando links para tais, bem como as funcionalidades atuais do projeto, como é possível ver na figura 10.

Logo em seguida, para abrir o arquivo desejado, basta selecionar o botão *File* da barra superior (figura 11) e o usuário será apresentado ao seletor de arquivos (figura 12) para escolher a sua base de dados local desejada.

Após selecionar um arquivo compatível, a janela de seleção de arquivos se fecha e coloca em destaque uma janela persistente de carregamento (figura 13), que mantém o usuário informado acerca da proporção de dados que foram carregados, necessários para iniciar o processo de análise.

Uma vez concluída, a janela da figura 14 se apresenta, com o objetivo de guiar o usuário sobre quais informações podem ser extraídas através do fornecimento de perguntas previamente selecionadas.

Ao escolher a pergunta desejada e selecionar em quais iterações o TKGEvolViewer deverá apresentar os resultados, haverá mais um período de carregamento — proporcional à quantidade de *timestamps* escolhidos — até que, por fim, uma visão parecida com a figura 15 será apresentada ao usuário, mostrando conceitos etiquetados em destaque, assim como as suas conexões ao longo da estrutura espacial do grafo, para todas as iterações selecionadas.

Após esse processo, a representação gráfica da base de dados estará pronta para ser explorada livremente pelo usuário.

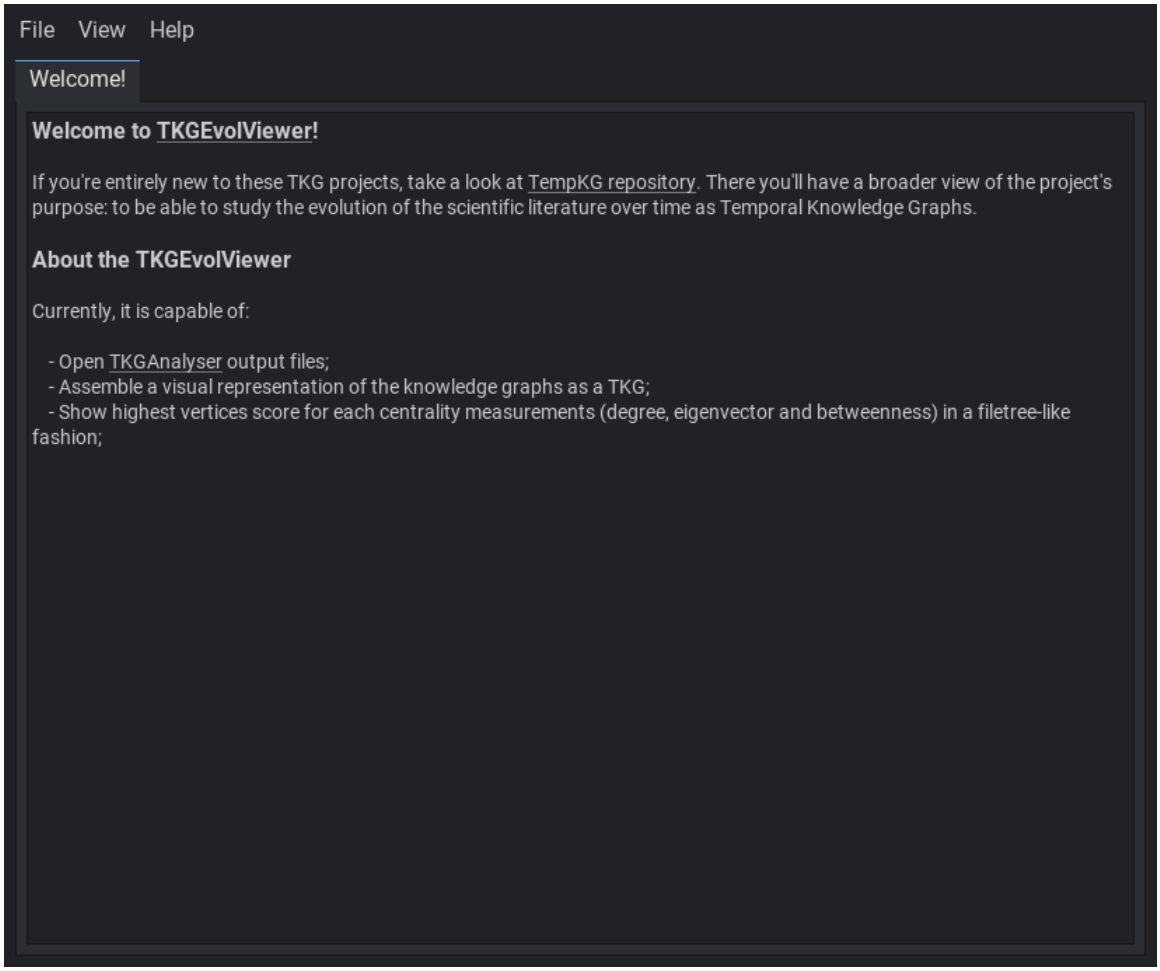


Figura 10: Tela de boas-vindas. É a primeira visão que o usuário tem da ferramenta, exibindo um texto de contextualização do projeto e dos programas relacionados na aba 'Welcome!'. Também está visível uma barra de opções, com os botões *File*, *View* e *Help*.

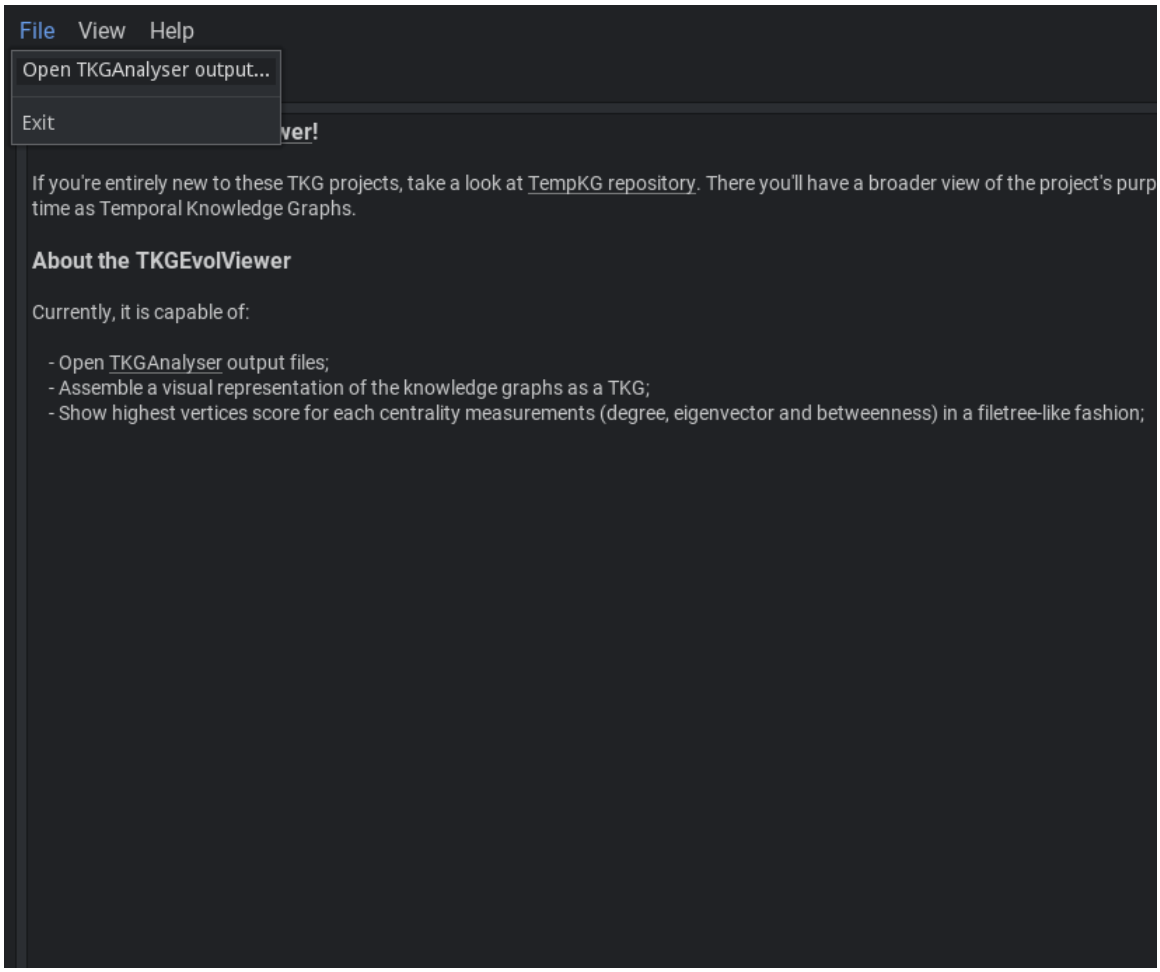


Figura 11: Opções disponíveis ao clicar no menu *File* aparecem em uma janela flutuante: *Open TKGAnalyser output...* e *Exit*.

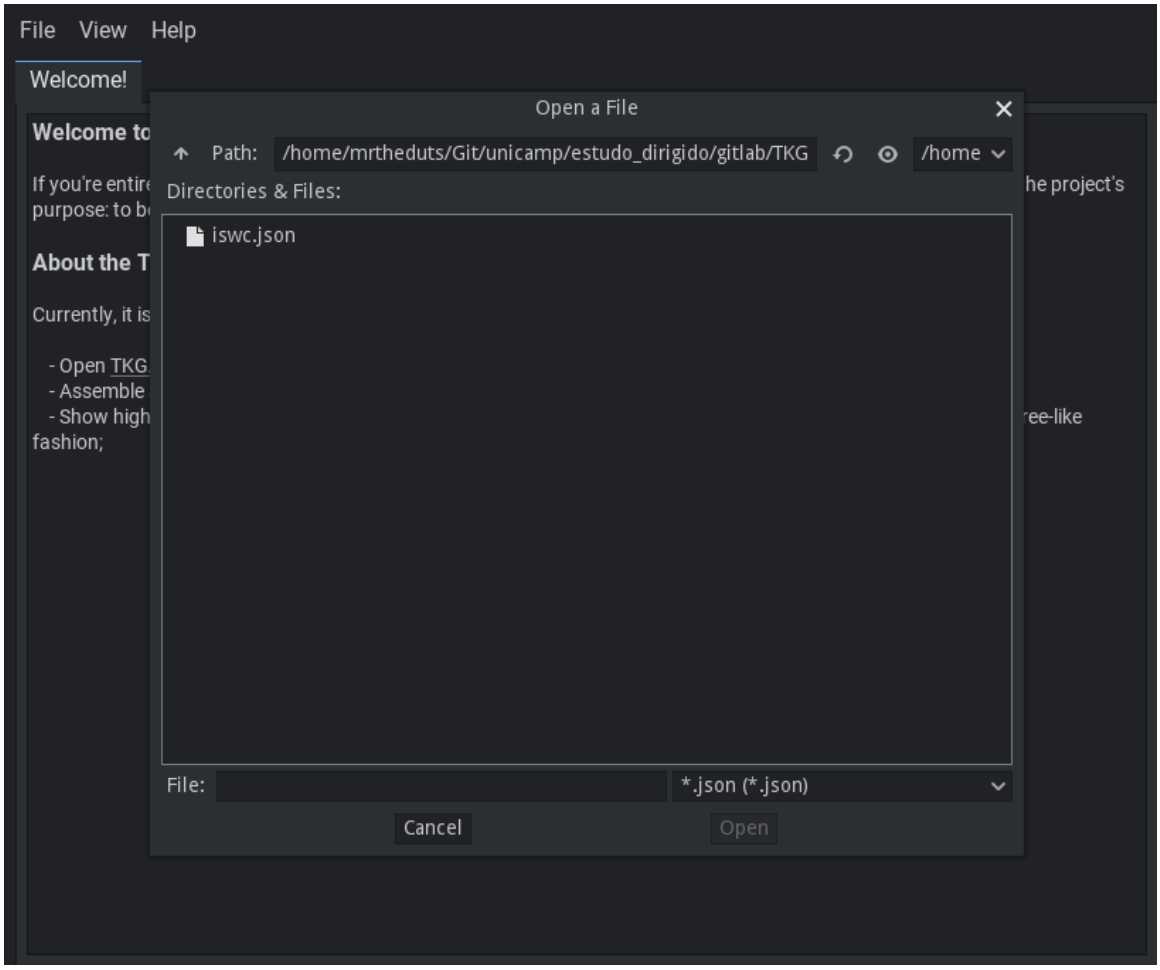


Figura 12: Tela flutuante de seleção de arquivos para ser carregado na ferramenta. Aparece logo após a opção *Open TKGAnalyser output...* ser selecionada no menu *File*. Nela há uma interface que mostra a localização atual do programa na máquina do usuário, bem como botões que permitem se deslocar dentro destas pastas e selecionar um arquivo como entrada para o programa.

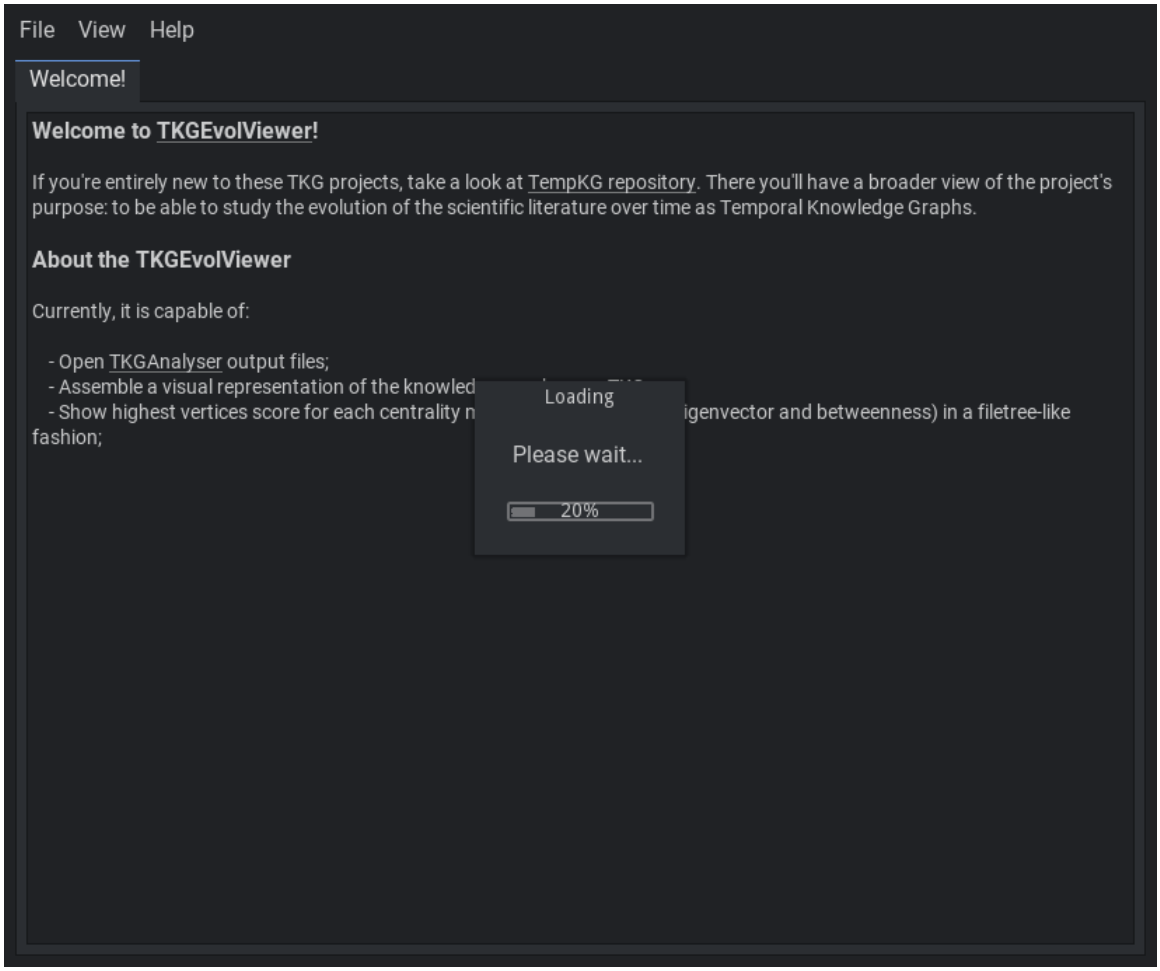


Figura 13: Após um arquivo válido ser selecionado para a visualização, a ferramenta exibe o progresso do carregamento dos seus dados em tempo real para o usuário na forma de uma janela persistente flutuante com uma barra de porcentagem.

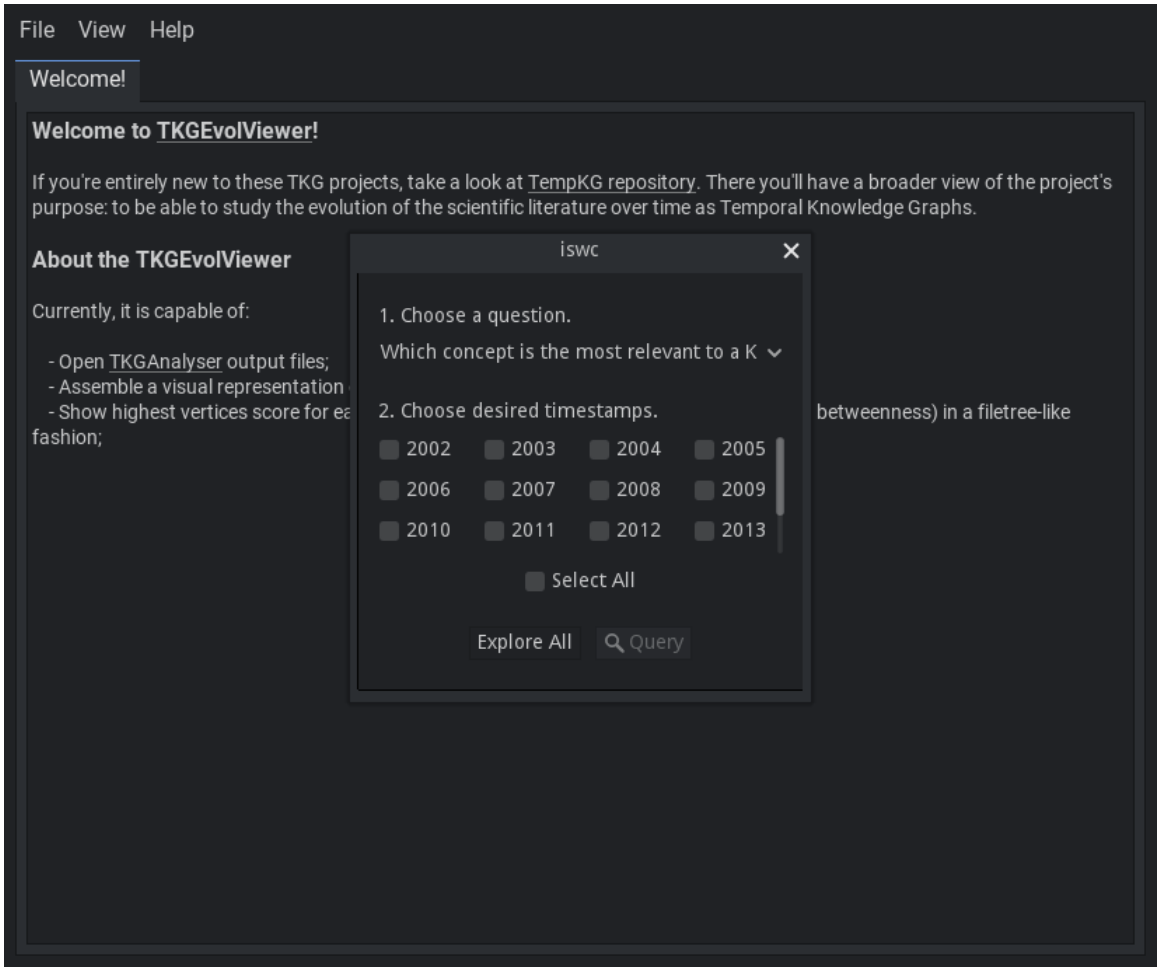


Figura 14: Após a informação necessária acerca do *TKG* ser carregada, é apresentada essa tela inicial, de forma a guiar o usuário até o seu objetivo de extrair determinada característica do conjunto de dados através das perguntas pré-selecionadas para *timestamps* selecionados.

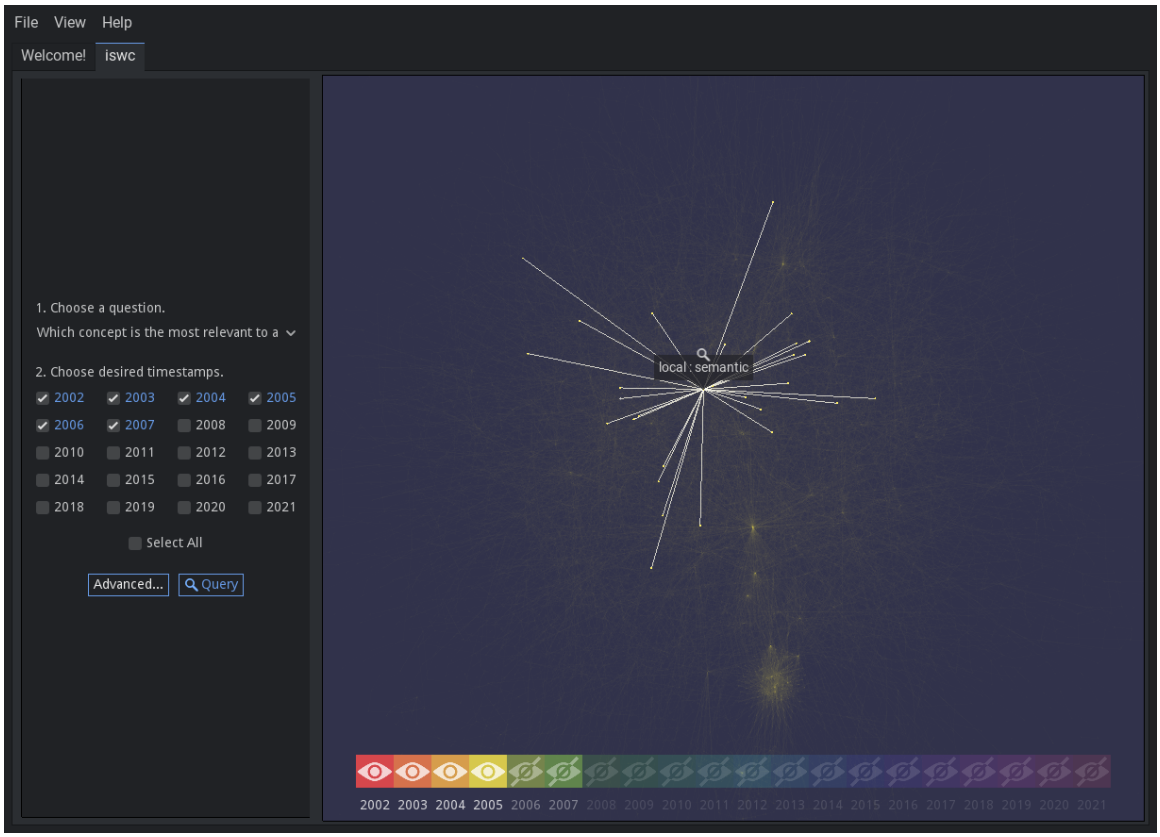


Figura 15: Exemplo de execução do TKGEvolViewer, com um enfoque na representação do Grafo de Conhecimento do conjunto de dados 'iswc' na iteração do ano de 2005, com o conceito 'semantic' em destaque. À esquerda observa-se um modal com uma seleção de questões, uma tabela com os anos disponíveis naquele conjunto de dados e dois botões: 'Advanced', e 'Query'. Abaixo do grafo, uma linha do tempo serve como referência para relacionar os *timestamps* com as cores empregadas em cada iteração deste, bem como ícones indicando a visibilidade atual de cada uma das iterações.