

Análise Visual de Dados de Patentes: Ferramentas e Oportunidades para Compreender a Produção Técnica Brasileira

por

Felipe Coelho de Oliveira Campos

Orientador:

Thiago Magela Rodrigues Dias



CEFET-MG Campus Divinópolis

31 de agosto de 2024

Resumo

A extração e análise visual de dados de patentes brasileiras tem como intuito entender melhor onde está concentrada a produção científica brasileira, quem financia essa produção e quem são os autores dessas produções. Sendo assim, o projeto realizado teve como intuito criar um grafo contendo três tipos de nós diferentes, sendo eles os autores, os depositantes e a própria patente, e suas respectivas arestas sendo criadas a partir de uma filtragem realizada analisando cada uma das patentes na base de dados. Com o grafo completamente criado é possível extrair informações muito valiosas, como por exemplo, quais depositantes estão financiando algum tipo de patente, principais autores por área de patente, interligação de patentes por meio de autores e depositantes, quem mais financia a produção técnica brasileira, entre muitas outras informações relevantes.

Conteúdo

Resumo	i
1 Introdução	1
2 Metodologia	2
3 Desenvolvimento	4
3.1 Arquivos XML	4
3.2 Leitura dos arquivos	4
3.3 Criação de tabelas	6
3.3.1 Tabela de nós	6
3.3.2 Tabela de arestas	7
3.4 Visualização de dados	9
4 Resultados e Discussões	15
Bibliografia	16

Capítulo 1

Introdução

Cada vez mais as novas tecnologias e sistemas digitais tem afetado o cotidiano das pessoas nas mais diversas áreas. Diariamente novos dispositivos, aplicações, meios digitais permeiam o mercado, trazendo versões melhores de recursos e/ou funcionalidades que até então conhecíamos ou apresentavam novas soluções.

De acordo com dados levantados pelo Instituto Nacional de Propriedade Intelectual (INPI) [4], são publicados anualmente cerca de 28 mil registros de novas patentes. Neste contexto as universidades tem desempenhando um papel importante no desenvolvimento tecnológico nacional, boa parte das invenções e inovações geradas possuem como origem pesquisas iniciadas em instituições de ensino públicas e privadas, que vem aumentando ao longo dos anos tornando-as um grande polo de inovação nacional, reflexo disso é o resultado do Ranking dos Depositantes Residentes de Patentes de Invenção, divulgado pelo INPI [3], apresentando universidades federais e estaduais no topo do ranking de maiores depositantes no ano de 2023.

Mediante ao exposto, acompanhar o desenvolvimento tecnológico, de um setor, ou um grupo de empresas tem se tornado um pré-requisito para instituições que desejam se destacar no meio tecnológico e de inovação e, investir no acompanhamento do desenvolvimento tecnológico por meio de documento de patentes é importante pois viabiliza identificar tecnologias emergentes, especialistas em determinadas tecnologias, identificação de possíveis parceiros para novos projetos.

Sendo assim, for utilizado nessa pesquisa dados já extraídos anteriormente pelo Professor Raulivan Rodrigo da Silva [4], que utilizou um web scraper para extrair as patentes do INPI e identificou as patentes na base de dados internacional Espacenet [1], gerando assim um único arquivo XML por patente, nele contendo o nome da patente, seus respectivos depositantes e autores, facilitando assim a criação de um grafo para a pesquisa, já que ter acesso aos dados de documentos de patentes não é considerada uma tarefa trivial.

Capítulo 2

Metodologia

Para o desenvolvimento do projeto, inicialmente foi-se pensado em como seria feito os arquivos para a criação do grafo, que inicialmente seria visualizado na plataforma do NetworkX [5] no Python [7], mas depois, visando uma melhor flexibilidade do grafo, foi migrado para o software Gephi [2].

Ao migrar para o Gephi [2], fez-se necessário duas bases de dados diferentes para a criação do grafo, sendo elas uma tabela de nós e uma tabela de arestas, que seriam submetidas no software e o mesmo faria o carregamento de todos os dados em forma de grafo.

A tabela de nós foi criada a partir de um algoritmo criado para percorrer toda a base de dados XML usada no projeto e, a partir de sua definição, o nó é adicionado em uma dataframe Pandas [6] com seu respectivo nome e sua definição, definindo se o nó é um autor, patente ou depositante. Após ter sido percorrida toda a base de dados, o dataframe é convertido em um arquivo CSV de três colunas, sendo elas: coluna de identificação, coluna de nome e coluna de definição.

Com a tabela de nós criada, é possível criar a tabela de arestas, onde foi criado outro algoritmo para que percorra toda a base de dados XML usada no projeto, juntamente com verificação na tabela de nós, onde no arquivo de uma patente é criada todas suas respectivas ligações/arestas entre a patente, os autores e os depositantes, estes sendo verificados seus respectivos identificadores na tabela de nós, e, em vez de usar os respectivos nomes, são usados os identificadores presentes na tabela de nós. Após ter sido percorrida toda a base de dados, o dataframe é convertido em um arquivo CSV de duas colunas, sendo elas as colunas com o número de identificação dos nós, presentes na tabela de nós, e a linha representando a aresta.

Possuindo as duas tabelas, de nós e de arestas, é possível utilizar o Gephi [2] para a criação, visualização e manipulação do grafo, sendo agora, possível extrair informações relevantes dos dados.

É possível analisar o fluxo de trabalho na figura 2.1 abaixo:

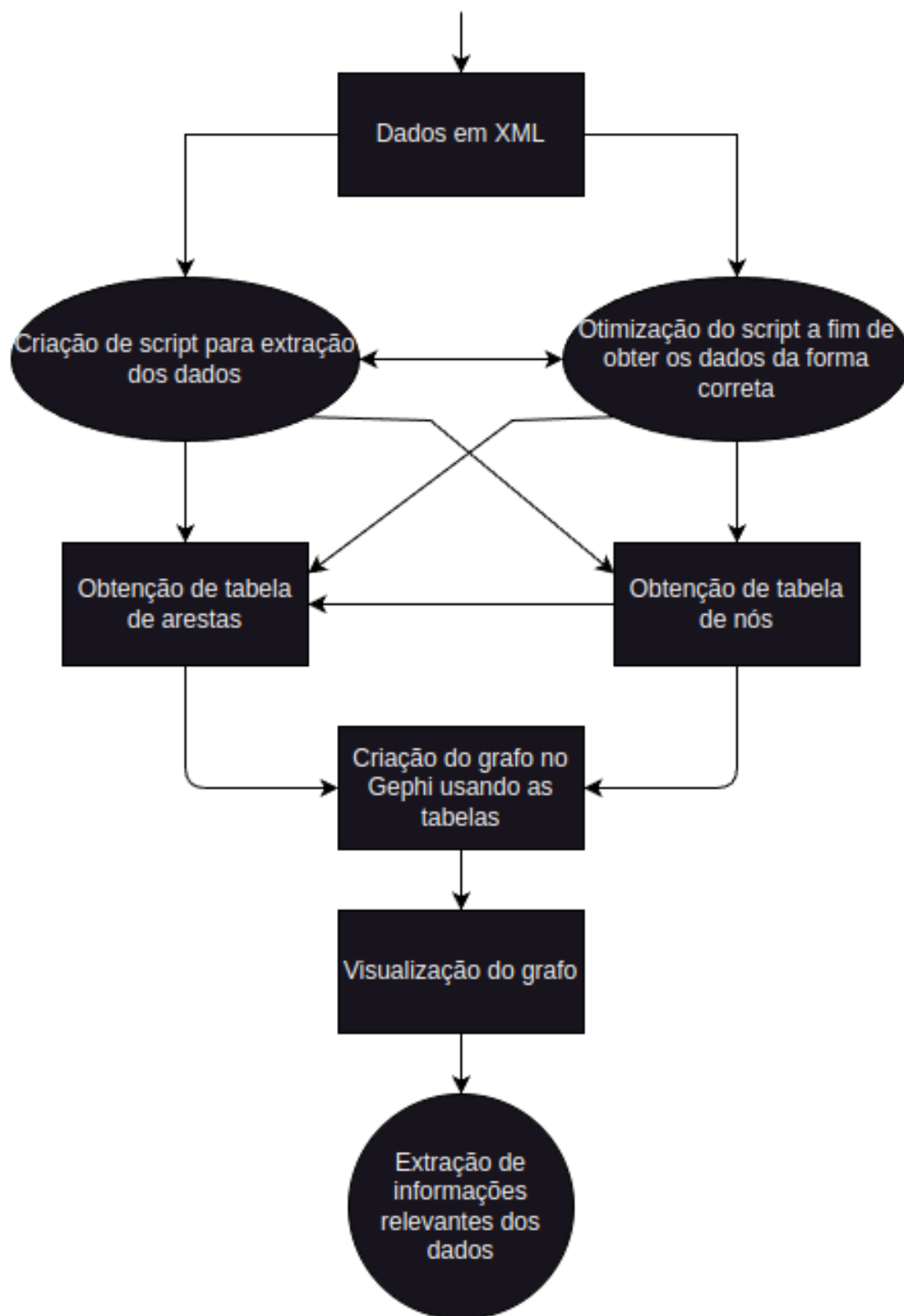


Figura 2.1: Diagrama metodológico do desenvolvimento

Capítulo 3

Desenvolvimento

Neste capítulo será apresentada detalhadamente as tecnologias usadas, algoritmos criados, e técnicas utilizadas. Resumindo, na maior parte do projeto, foi usado Python [7], a leitura dos arquivos XML foi feita usando uma biblioteca já nativa do Python chamada *xml.etree.ElementTree* [8], para o armazenamento do dataframe enquanto o algoritmo era executado e transformação do dataframe em um arquivo CSV, foi usada a biblioteca *Pandas* [6], e após a execução do algoritmo, foi usado o Gephi [2] para a visualização dos dados.

3.1 Arquivos XML

Os arquivos XML são bem estruturados e foi possível extrair dos mesmos quem são os autores, depositante e o nome da patente, pelo atributo *name* presente em cada *tag <field>*, onde se o atributo *name* é *inventor*, o valor associado àquela *tag <field>* é o nome de um autor, e caso o atributo *name* é *applicant*, o valor associado àquela *tag <field>* é o nome de um depositante. O nome da patente no arquivo pode ser encontrado na *tag <field>* que possui o atributo *name* igual a *title.lattes*. É possível ver como é a estrutura dos arquivos XML na figura 3.1.

3.2 Leitura dos arquivos

A leitura dos arquivos foi feita utilizando a biblioteca nativa do Python [7], chamada *xml.etree.ElementTree* [8], lendo o arquivo por completo e encontrando os dados necessários com base no atributo *name*. Possível ver como funciona no código na figura 3.2, onde todo o conteúdo de um arquivo XML é salvo na variável *root*:

Possuindo esse arquivo salvo em uma única variável, é possível extrair e manipular os dados, como é mostrado no código na figura 3.3:

Explicando o código da figura 3.3, é criado um loop que percorre todas as *tags <field>*, e o parâmetro *findall* funciona para encontrar as *tags <field>* que tenham como *name* o nome da patente, do autor ou do depositante, adicionando esses valores em uma variável para cada um dos três tipos de nó.

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <entity-relation-data lastUpdate="2023-03-27 16:24:06" record="11198" source="Espacenet">
3   <entities>
4     <entity ref="Patent11198" type="Patent">
5       <semanticIdentifier>brcris::eee760dd48b51f02e7453c8d5b50c96d</semanticIdentifier>
6       <semanticIdentifier>docdb::BR202014029063U2</semanticIdentifier>
7       <semanticIdentifier>epodoc::BR202014029063U</semanticIdentifier>
8       <semanticIdentifier>docdb::BR202014029063U</semanticIdentifier>
9       <semanticIdentifier>epodoc::BR202014029063U</semanticIdentifier>
10      <semanticIdentifier>original::202014029063</semanticIdentifier>
11      <field name="urlEspacenet" value="https://worldwide.espacenet.com/patent/search?q=202014029063"/>
12      <field name="identifier.lattes" value="0007660165104840"/>
13      <field name="identifier.espacenet" value="202014029063"/>
14      <field name="kindCode" value="U2"/>
15      <field name="title.espacenet" value="dispositivo aplicado na producao de lastros para estacas de torpedos de plataformas de petroleo"/>
16      <field name="title.lattes" value="Dispositivo aplicado na producao de lastros para estacas de torpedos de plataformas de petroleo"/>
17      <field name="depositDate" value="2014-11-21"/>
18      <field name="publicationDate" value="2016-05-24"/>
19      <field name="countryCode" value="BR"/>
20      <field name="inventor" value="CLEBER WILSON DE SOUSA"/>
21      <field name="inventor" value="DIEGO ANTUNES GUIMARAES"/>
22      <field name="applicant" value="CLEBER WILSON DE SOUSA"/>
23      <field name="applicant" value="DIEGO ANTUNES GUIMARAES"/>
24      <field name="classification.ipc">
25        <field name="sequence" value="1"/>
26        <field name="text" value="B22D15/04AI"/>
27        <field name="section" value="B"/>
28        <field name="class" value="22"/>
29        <field name="subclass" value="D"/>
30        <field name="group" value="15"/>
31        <field name="subgroup" value="04AI"/>
32      </field>
33    </entity>
34  </entities>
35  <relations>
36    <relation fromEntityRef="Patent11198" toEntityRef="Person7801" type="Inventor"/>
37    <relation fromEntityRef="Patent11198" toEntityRef="Person7801" type="Depositor"/>
38  </relations>
39 </entity-relation-data>
40
```

Figura 3.1: Exemplo de arquivo XML usado

```
1 for file in os.listdir("./patents"):
2     if file.endswith(".xml"):
3         tree = ET.parse(os.path.join("./patents", file))
4         root = tree.getroot()
```

Figura 3.2: Código para salvar arquivo XML em uma variável

```
1 for field in root.findall('.//field[@name="{ }"]'.format('title.lattes')):
2     patent = field.get('value').upper()
3
4     if df_temp['patent'].isin([patent]).any() == False:
5         df_temp = df_temp._append({'patent': patent}, ignore_index=True)
6
7     for field in root.findall('.//field[@name="{ }"]'.format('inventor')):
8         inventors.append(field.get('value'))
9
10    for field in root.findall('.//field[@name="{ }"]'.format('applicant')):
11        applicants.append(field.get('value'))
```

Figura 3.3: Código para extrair dados de arquivo XML


```
1 def savePatent(patent, lock):
2     global df, count
3
4     if df['label'].isin([patent]).any() == False:
5         lock.acquire()
6         df = df._append({'id': count, 'label': patent, 'role': "patent"}, ignore_index=True)
7         count+=1
8         lock.release()
```

Figura 3.4: Função que salva patente

```
1 def saveApplicants(applicants, lock):
2     global df, count
3
4     for applicant in applicants:
5         isThereSuchAnApplicant = (df['label'] == applicant) & (df['role'] == "applicant")
6
7         if isThereSuchAnApplicant.any() == False:
8             lock.acquire()
9             df = df._append({'id': count, 'label': applicant, 'role': "applicant"}, ignore_index=True)
10            count+=1
11            lock.release()
```

Figura 3.5: Função que salva depositantes

3.3 Criação de tabelas

A criação das tabelas pode ser dividida em duas partes, primeiro a tabela de nós e segundo a tabela de arestas, que depende da tabela de nós para ser criada.

3.3.1 Tabela de nós

Após possuir os dados necessários de uma patente salvos em três diferentes variáveis, é possível partir para a criação das tabelas, iniciando pela tabela de nós, que pode ser separada inicialmente em três funções: *savePatent()* que salva a patente, *saveApplicants()* que salve os inventores e *saveInventors()* que salva os depositantes. As funções podem ser visualizadas pelas figuras 3.4, 3.5 e 3.6, respectivamente.

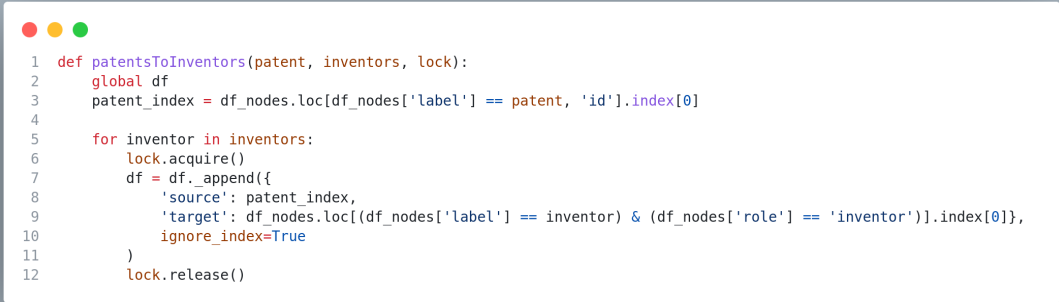
```
1 def saveInventors(inventors, lock):
2     global df, count
3
4     for inventor in inventors:
5         isThereSuchAnInventor = (df['label'] == inventor) & (df['role'] == "inventor")
6
7         if isThereSuchAnInventor.any() == False:
8             lock.acquire()
9             df = df._append({'id': count, 'label': inventor, 'role': "inventor"}, ignore_index=True)
10            count+=1
11            lock.release()
```

Figura 3.6: Função que salva inventores



```
1 df.to_csv('nodeTable.csv', index = False)
```

Figura 3.7: Código que transforma dataframe em arquivo CSV contendo tabela de nós



```
1 def patentsToInventors(patent, inventors, lock):
2     global df
3     patent_index = df_nodes.loc[df_nodes['label'] == patent, 'id'].index[0]
4
5     for inventor in inventors:
6         lock.acquire()
7         df = df._append({
8             'source': patent_index,
9             'target': df_nodes.loc[(df_nodes['label'] == inventor) & (df_nodes['role'] == 'inventor')].index[0]},
10            ignore_index=True
11        )
12        lock.release()
```

Figura 3.8: Função que liga a patente à inventores

Ao executar essas funções, os dados serão inseridos em uma dataframe *Pandas* [6] criado de forma global, juntamente com um contador que inicia em zero e serve como número identificador daquele dado. Conforme a primeira linha de código da figura 3.2, é executado um loop para percorrer todos os arquivos XML de patentes, e conforme as iterações vão sendo executadas, o dataframe cresce com os dados inseridos, juntamente com o contador, que é somado uma unidade a cada dado inserido no dataframe.

Após percorrer todos os arquivos e ter um dataframe consideravelmente carregado, é feita a transformação do dataframe em um arquivo CSV, nomeado de *nodeTable*, conforme código na figura 3.7.

3.3.2 Tabela de arestas

Inicialmente para a criação da tabela de arestas, é seguido os mesmos passos da criação da tabela de nós, lendo os arquivos de patentes, como mostrado nas figuras 3.2 e 3.3.

Após isso, são executadas funções que ligam os nós uns com os outros com as funções: *patentsToInventors()*, *patentsToApplicants()*, *inventorsToInventors()*, *applicantsToApplicants()* e *inventorsToApplicants()*, essas podendo ser visualizadas nas figuras 3.8, 3.9, 3.10, 3.11 e 3.12, respectivamente.

É possível observar nas funções, que a tabela de nós, denominada no código de *df_nodes*, é usada para verificar se o dado existe e se o mesmo é correto, e se essas condições forem verdadeiras, é atribuído o valor do índice do nó à coluna correspondente, seja *source* ou *target*, no dataframe *Pandas* [6].

3.3 Criação de tabelas

```
1 def patentsToApplicants(patent, applicants, lock):
2     global df
3     patent_index = df_nodes.loc[df_nodes['label'] == patent, 'id'].index[0]
4
5     for applicant in applicants:
6         lock.acquire()
7         df = df._append({
8             'source': patent_index,
9             'target': df_nodes.loc[(df_nodes['label'] == applicant) & (df_nodes['role'] == 'applicant')].index[0]},
10            ignore_index=True
11        )
12     lock.release()
```

Figura 3.9: Função que liga a patente à depositantes

```
1 def inventorsToInventors(inventors, lock):
2     global df
3
4     for i in range(len(inventors)):
5         for j in range(i+1, len(inventors)):
6             lock.acquire()
7             df = df._append({
8                 'source': df_nodes.loc[(df_nodes['label'] == inventors[i]) & (df_nodes['role'] == 'inventor')].index[0],
9                 'target': df_nodes.loc[(df_nodes['label'] == inventors[j]) & (df_nodes['role'] == 'inventor')].index[0]},
10                ignore_index=True
11            )
12     lock.release()
```

Figura 3.10: Função que liga inventores à inventores

```
1 def applicantsToApplicants(applicants, lock):
2     global df
3
4     for i in range(len(applicants)):
5         for j in range(i+1, len(applicants)):
6             lock.acquire()
7             df = df._append({
8                 'source': df_nodes.loc[(df_nodes['label'] == applicants[i]) & (df_nodes['role'] == 'applicant')].index[0],
9                 'target': df_nodes.loc[(df_nodes['label'] == applicants[j]) & (df_nodes['role'] == 'applicant')].index[0]},
10                ignore_index=True
11            )
12     lock.release()
```

Figura 3.11: Função que liga depositantes à aplicantes

```
1 def inventorsToApplicants(inventors, applicants, lock):
2     global df
3
4     for inventor in inventors:
5         for applicant in applicants:
6             lock.acquire()
7             df = df._append({
8                 'source': df_nodes.loc[(df_nodes['label'] == inventor) & (df_nodes['role'] == 'inventor')].index[0],
9                 'target': df_nodes.loc[(df_nodes['label'] == applicant) & (df_nodes['role'] == 'applicant')].index[0]},
10                ignore_index=True
11            )
12     lock.release()
```

Figura 3.12: Função que liga inventores à depositantes



Figura 3.13: Código que transforma dataframe em arquivo CSV contendo tabela de arestas

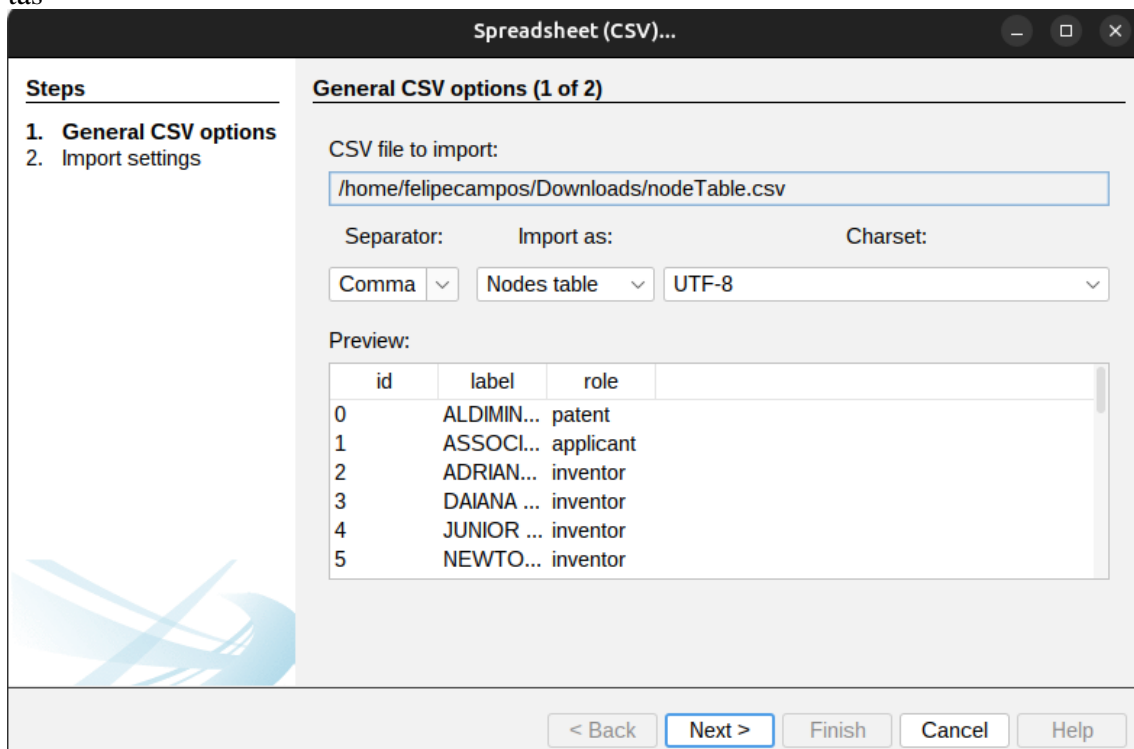


Figura 3.14: Janela de configuração para criar nós no Gephi

Após percorrer todos os arquivos e possuir um dataframe contendo todas as ligações existentes entre os nós, é feita a transformação do dataframe em um arquivo CSV, nomeado *edgeTable*, conforme código na figura 3.13.

3.4 Visualização de dados

Agora para a criação do grafo utilizando o Gephi [2], é necessário submeter as tabelas no software, seguindo os seguintes passos: *File > Import Spreadsheet* e selecionar o arquivo CSV da tabela de nós.

Após isso, é mostrado uma janela como a da figura 3.14. É necessário certificar que a configuração "*Separator*" esteja em "*Comma*" e "*Import as*" esteja em "*Nodes table*". Após certificar, basta clicar em "*Next*", e logo em seguida "*Finish*".

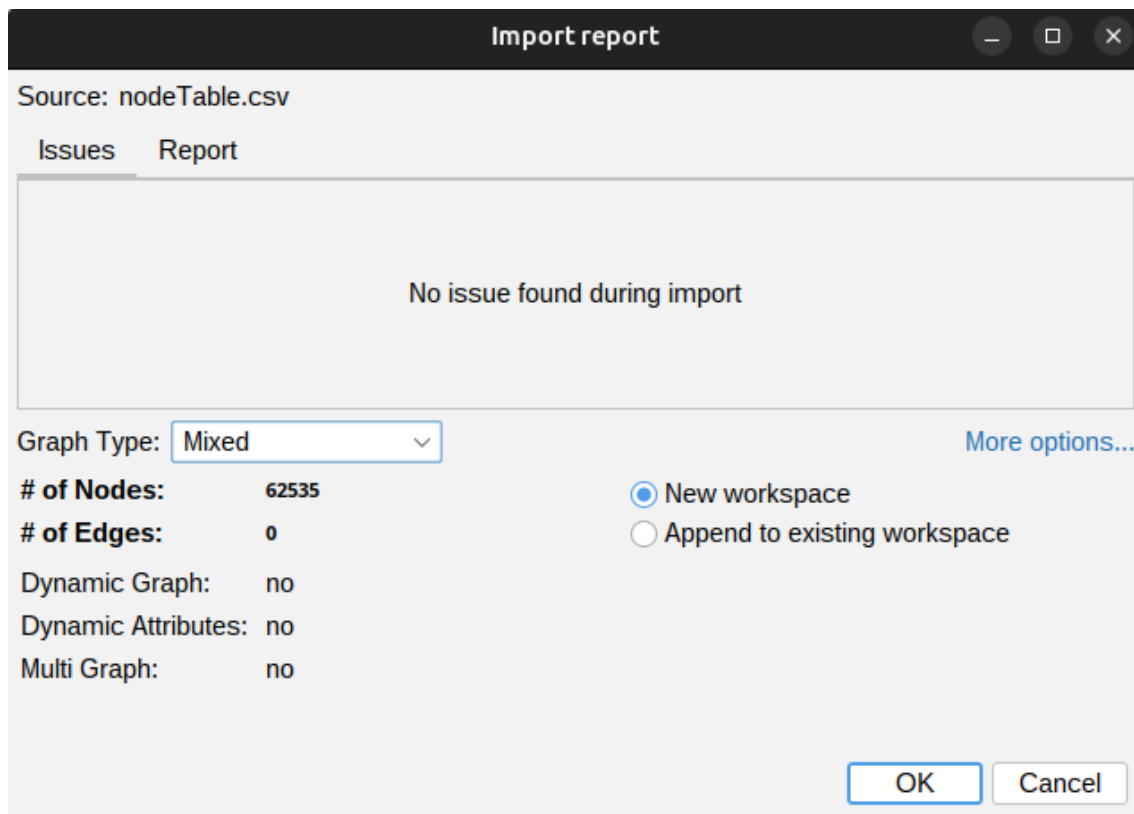


Figura 3.15: Janela de configuração para tipo do grafo

É mostrado uma outra janela, como a da figura 3.15. É necessário certificar que a opção "Graph type" seja "Undirected", e selecionar a opção "Append to existing workspace". Após certificar, basta clicar em "OK".

Com os nós criados, agora é possível criar as arestas entre os nós, seguindo os seguintes passos: *File > Import Spreadsheet* e selecionar o arquivo CSV da tabela de arestas.

Após isso, é mostrado uma janela como a da figura 3.16. É necessário certificar que a configuração "Separator" esteja em "Comma" e "Import as" esteja em "Edges table". Após certificar, basta clicar em "Next", e logo em seguida "Finish".

É mostrado uma outra janela, como a da figura 3.15. É necessário certificar que a opção "Graph type" seja "Undirected", e selecionar a opção "Append to existing workspace". Após certificar, basta clicar em "OK".

Com os nós e arestas criados, faz-se necessário alterar algumas coisas no grafo, como coloração e espaçamento para melhor visualização, e para mudar a cor dos nós e representá-los de acordo com a sua função, na aba de "Appearance" > "Nodes" > "Partition", é selecionado a opção "role", e clicado em "Apply", como pode-se observar na figura 3.18.

E quanto ao espaçamento, na aba de "Layout", é selecionado a opção "Force Atlas 2", e clicado em "Run", como pode-se observar na figura 3.19

Também para melhor visualização dos nós, faz-se necessário dimensionar os mesmos por peso, ou seja, quanto mais arestas ligadas a esse nó, maior ele deve ser, e é possível fazer isso na aba de "Appearance" > "Nodes" > "Ranking", é selecionado a op-

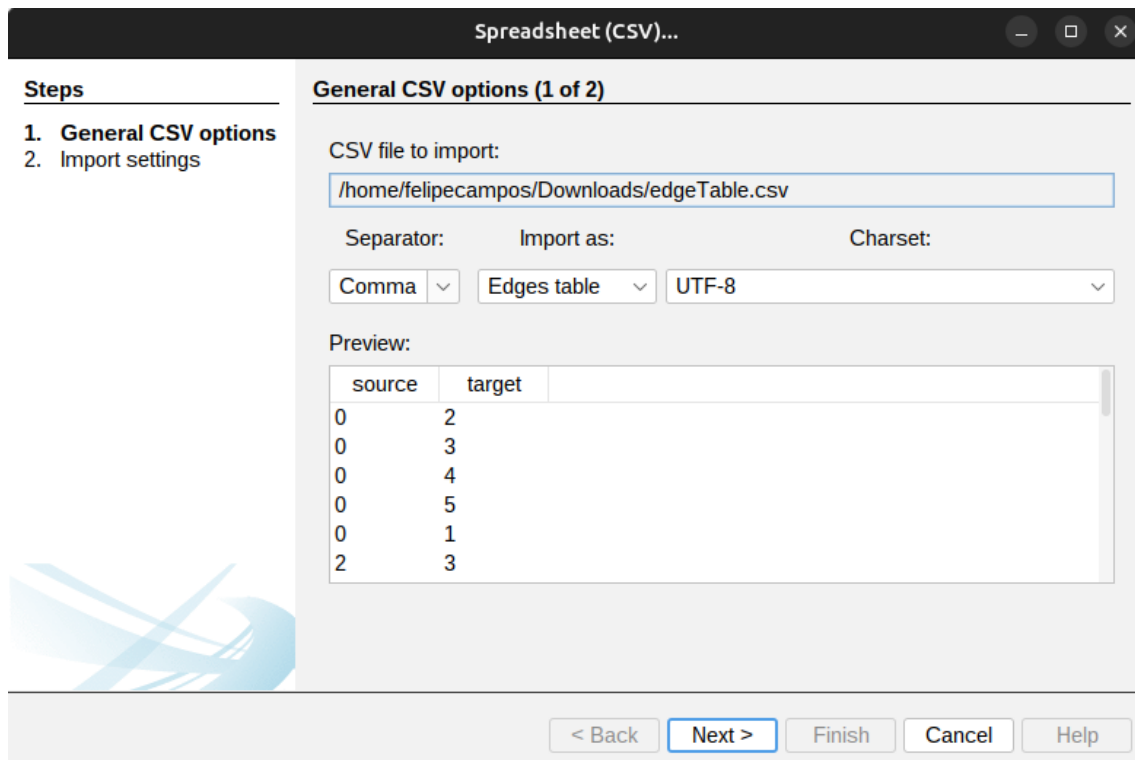


Figura 3.16: Janela de configuração para criar arestas no Gephi

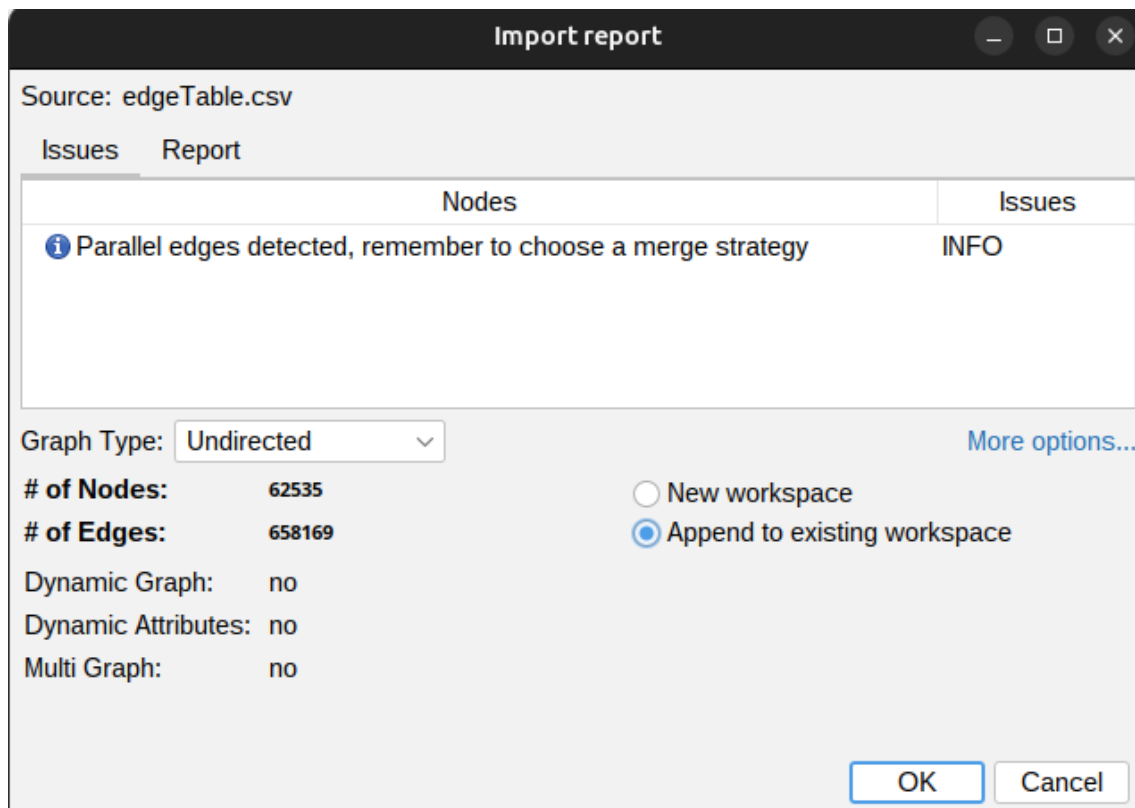


Figura 3.17: Janela de configuração para tipo do grafo

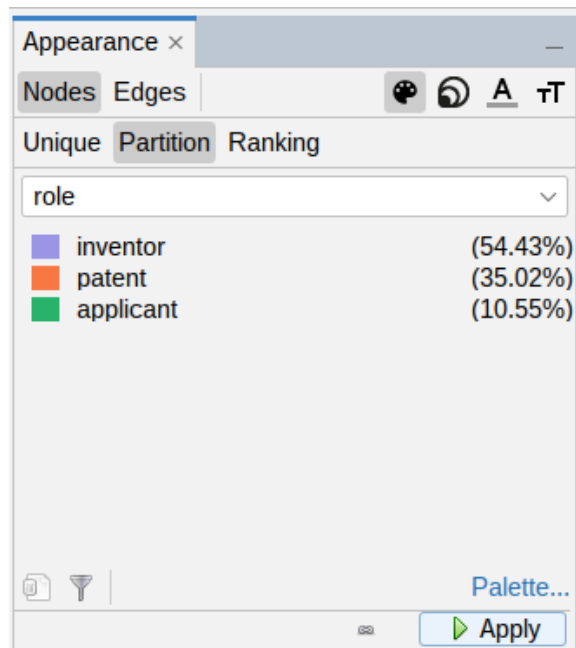


Figura 3.18: Coloração de nós por função

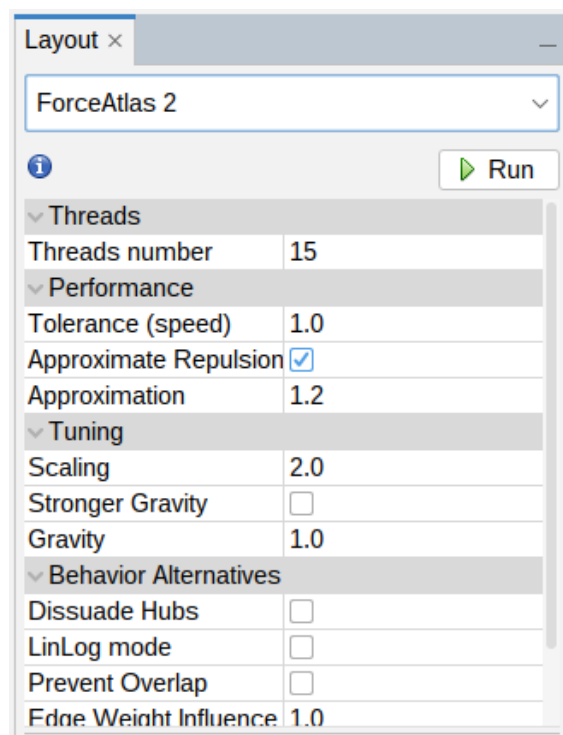


Figura 3.19: Espaçamento dos nós

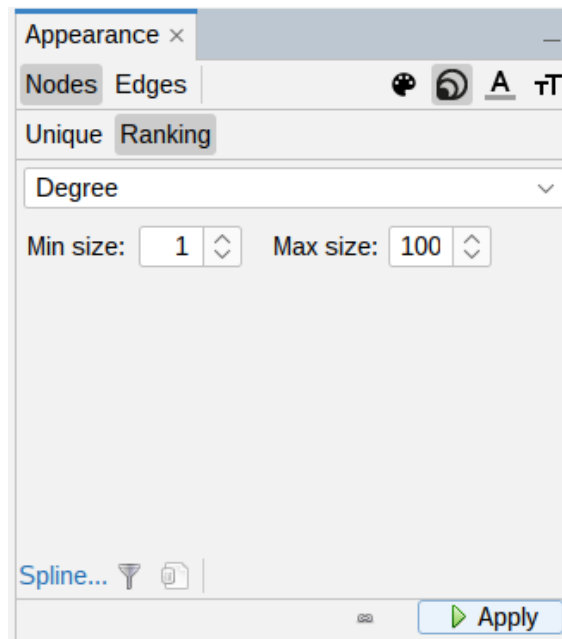


Figura 3.20: Tamanho dos nós por peso

ção "*Degree*", e o tamanho mínimo é setado em 1 e o tamanho máximo setado em 100, como pode ser visto na figura 3.20.

Por fim, é possível visualizar um grafo denso como mostrado na figura 3.21, sendo possível visualizar inicialmente quatro grandes depositantes coloridos em verde, sendo eles: UFRGS, Unicamp, USP e UFMG.

É possível você encontrar todo o código do projeto no seguinte repositório.

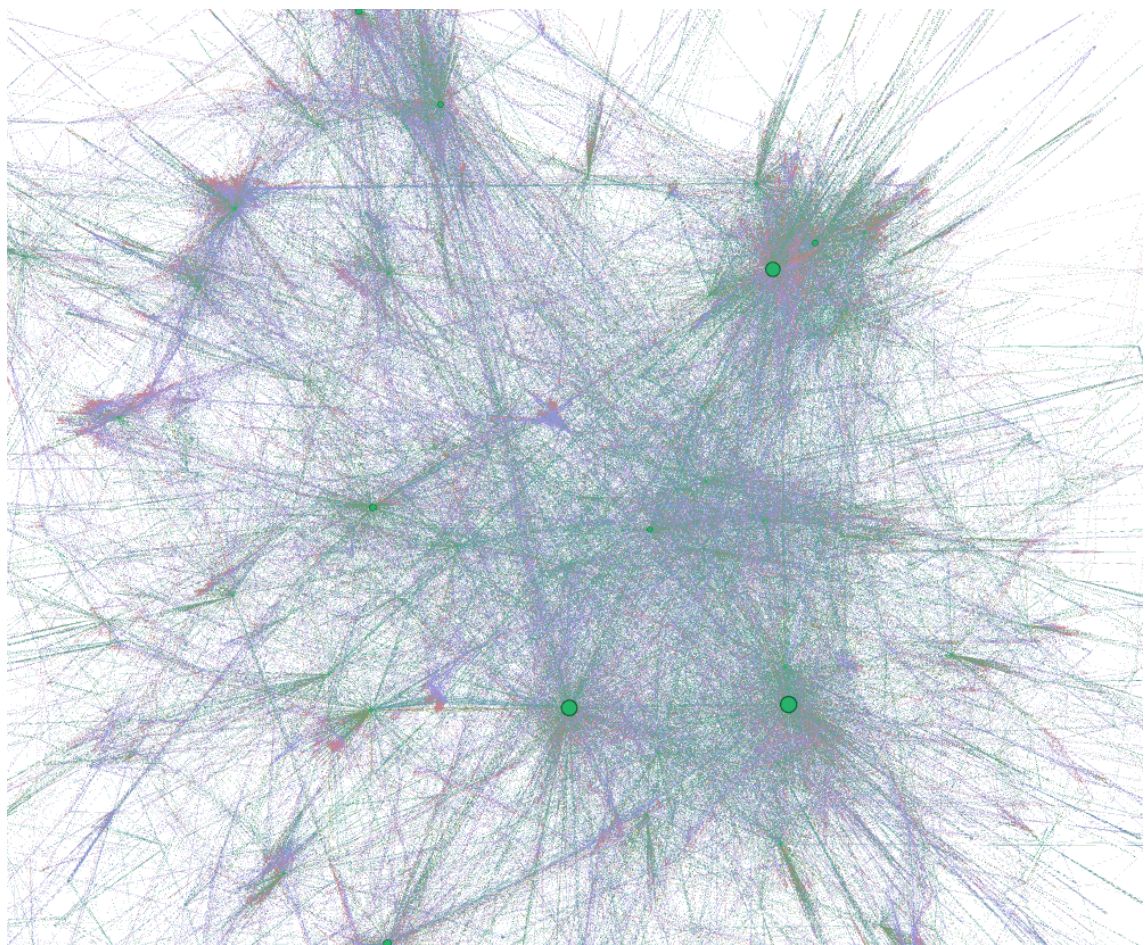


Figura 3.21: Grafo com coloração de nós por função e com peso por aresta

Capítulo 4

Resultados e Discussões

O grafo resultante ainda pode ser melhorado com filtragens e espaçamentos, mas dependendo de qual finalidade da pesquisa no grafo. É possível extrair muitas informações valiosas dos dados contidos no grafo resultante, pois é possível afirmar que possui dados de toda a produção técnica brasileira em um só lugar.

Bibliografia

- [1] Espacenet. **Worldwide Patent Search**. Accessed: 2024-08-07. Disponível em: <<https://worldwide.espacenet.com/>>.
- [2] Gephi Consortium. **Gephi**: The Open Graph Viz Platform. Accessed: 2024-08-07. Disponível em: <<https://gephi.org/>>.
- [3] Instituto Nacional de Propriedade Intelectual (INPI). **Ranking de Depositantes Residentes 2023**. Ranking de depositantes de patentes de invenção no Brasil. Disponível em: <<https://www.gov.br/inpi/pt-br/central-de-conteudo/estatisticas/arquivos/estatisticas-preliminares/ranking-de-depositantes-residentes-2023.pdf>>.
- [4] Instituto Nacional de Propriedade Intelectual (INPI). **Pesquisa traça um panorama das patentes depositadas no Brasil**. Disponível em: <<https://portal.redefederal.org.br/noticias/noticias/geral/pesquisa-traca-um-panorama-das-patentes-depositadas-no-brasil>: :text=Cerca
- [5] NetworkX. **NetworkX**: Network Analysis in Python. Accessed: 2024-08-07. Disponível em: <<https://networkx.org/>>.
- [6] Pandas Development Team. **Pandas**: Python Data Analysis Library. Accessed: 2024-08-07. Disponível em: <<https://pandas.pydata.org/>>.
- [7] Python Software Foundation. **Python Programming Language**. Accessed: 2024-08-07. Disponível em: <<https://www.python.org/>>.
- [8] Python Software Foundation. **xml.etree.ElementTree — The ElementTree XML API**. Accessed: 2024-08-07. Disponível em: <<https://docs.python.org/3/library/xml.etree.elementtree.html>>.