UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

LUCAS PERES GASPAR

CONSULTAS SOBRE FONTES DE DADOS LIGADOS BASEADAS EM RECONHECIMENTO DE ENTIDADES NOMEADAS

FORTALEZA

2019

LUCAS PERES GASPAR

CONSULTAS SOBRE FONTES DE DADOS LIGADOS BASEADAS EM RECONHECIMENTO DE ENTIDADES NOMEADAS

Dissertação apresentada ao Curso de do PROGRAMA DE PÓS-GRADUAÇÃO EM Ciência da Computação do Centro de ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Ciência de Dados

Orientador: Prof. Dr. José Antônio Fernandes de Macedo

FORTALEZA

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

G232c Gaspar, Lucas Peres.
Consultas sobre fontes de dados ligados baseadas em reconhecimento de entidades nomeadas / Lucas Peres Gaspar. – 2019.
66 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2019.
Orientação: Prof. Dr. José Antônio Fernandes de Macedo.

1. Geração de SPARQL. 2. Esquema RDF. 3. Named Entity Recognition. 4. Query by Example. I. Título.
CDD 005

LUCAS PERES GASPAR

CONSULTAS SOBRE FONTES DE DADOS LIGADOS BASEADAS EM RECONHECIMENTO DE ENTIDADES NOMEADAS

Dissertação apresentada ao Curso de do PROGRAMA DE PÓS-GRADUAÇÃO EM Ciência da Computação do Centro de ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Ciência de Dados

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. José Antônio Fernandes de Macedo (Orientador)
Universidade Federal do Ceará (UFC)

Prof[a]. Dr[a]. Ticiana Linhares Coelho da Silva
Universidade Federal do Ceará (UFC)

Prof. Dr. César Lincoln Cavalcante Mattos
Universidade Federal do Ceará (UFC)

Prof. Dr. Marco Antonio Casanova
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)

Á Deus, que sempre esteve ao meu lado. À minha mãe, que me inspira todo dia seguindo sua própria carreira acadêmica. Ao meu pai, que sempre me incentivou a fazer aquilo que eu amo.

AGRADECIMENTOS

À Deus, que nunca me abandonou e sempre fez planos maiores do que eu possa compreender.

Aos meus pais, que sempre estiveram aqui por mim. Á minha mãe, que sempre cuidou de mim e me inspira, seguindo sua própria carreira acadêmica. Ao meu pai, que despertou em mim essa paixão pela área em que hoje sigo com tanto empenho e dedicação.

Aos meus orientadores José Macedo e Ticiana Linhares, que me acompanham desde minha graduação e, agora, em meu mestrado. Seu incentivo, orientação e amizade foram fundamentais para trilhar este caminho e alcançar mais esta conquista.

Aos meus amigos e colegas do Insight Data Science Lab, não só por sua ajuda e apoio, mas também por me ajudarem em diversas etapas de meu trabalho (e me aturarem em momentos de desespero).

À minha família e meus amigos que sempre acreditaram em mim, me dando forças e motivação para seguir meu caminho.

Por fim, ao Doutorando em Engenharia Elétrica, Ednardo Moreira Rodrigues, e seu assistente, Alan Batista de Oliveira, aluno de graduação em Engenharia Elétrica, pela adequação do *template* utilizado neste trabalho para que o mesmo ficasse de acordo com as normas da biblioteca da Universidade Federal do Ceará (UFC).

"Os passos que você dá não precisam ser tão grandes. Eles só precisam te levar na direção certa."

(Jemma Simmons - *Agents of S.H.I.E.L.D.*)

RESUMO

A Web evoluiu de uma rede de documentos interligados a uma onde tanto documentos e dados estão ligados, resultando no que é comumente conhecido como a Web de Dados, que inclui uma grande variedade de dados, normalmente publicados no formato RDF, sobre múltiplos domínios. Métodos intuitivos de acessar os dados RDF possuem grande importância, uma vez que a abordagem padrão seria executar uma consulta em SPARQL. Entretanto, isso pode ser muito difícil para usuários não-técnicos. Neste trabalho, abordamos o problema de *question answering* sobre bases RDF. Dada uma busca em linguagem natural ou em palavras-chaves, nosso objetivo é traduzi-la em uma consulta formal em SPARQL que capture a informação necessitada. Nós propomos duas abordagens baseadas em esquema para buscar sobre dados RDF sem nenhum conhecimento prévio da ontologia: Von-QBE e Von-QBNER. Isso é diferente do estado da arte uma vez que suas abordagens são baseadas nas instâncias de dados. Entretanto, isso pode ser infactível em cenários de *Big Data*, onde os dados são demasiados grandes e requerem muitos recursos computacionais para manter a base em memória. Também, muitas dessas soluções requerem que a base esteja triplificada, o que pode ser uma tarefa difícil em bases de dados legado. Por esta razão, Von-QBE utiliza apenas o esquema da base RDF para responder a busca do usuário. Entretanto, a busca do usuário pode conter informações sobre as instâncias de dados, que não vai corresponder, sintaticamente, a nenhum conceito ou propriedade no esquema da ontologia. Por exemplo, a busca *filmes com Angelina Jolie*. Considere que o esquema apresente apenas os conceitos *Filme* e *Atriz*, e a propriedade *estrelando*, que relaciona os dois conceitos. Se utilizarmos apenas o esquema da ontologia, apenas *Filme* será identificado na busca. Von-QBNER resolve essa limitação identificando as instâncias envolvidas na busca e o conceito a que correspondem no esquema utilizando modelos de *Named Entity Recognition* (NER). Os resultados são promissores para alguns conjuntos de dados reais avaliados, considerando que apenas o esquema da ontologia foi utilizado para gerar as consultas em SPARQL.

Palavras-chave: Geração de SPARQL. Esquema RDF. *Named Entity Recognition*. *Query by Example*.

ABSTRACT

The Web has evolved from a network of linked documents to one where both documents and data are linked, resulting in what is commonly known as the Web of Data, which includes a large variety of data usually published in RDF from multiple domains. Intuitive ways of accessing RDF data become increasingly important since the standard approach would be to run SPARQL queries. However, this can be extremely difficult for non-experts users. In this work, we address the problem of question answering over RDF. Given a natural language question or a keyword search string, our goal is to translate it into a formal query as SPARQL that captures the information needed. We propose two schema-based approach to query over RDF data without any previous knowledge about the ontology entities and schema: Von-QBE and Von-QBNER. This is different from the-state-of-art since the approaches are instance-based. However, it can be unfeasible using such approaches in big data scenarios where the ontology base is huge and demands a large number of computational resources to keep the knowledge base in memory. Moreover, most of these solutions need the knowledge base triplified, which can be a hard task for legacy bases. For this reason, Von-QBE uses only the RDF schema to answer the user's question. However, the user query may contain information about the data instances which does not syntactically match with any concept or property on the ontology schema. For instance, the query *Movies with Angelina Jolie*. Consider that the ontology schema only presents the concepts *Movie* and *Actress*, and a property *starring* which relates both concepts. If we use only the ontology schema, just the concept *Movie* matches with the user query. Von-QBNER addresses such limitation by identifying the instances involved in the query and their correspondent concept or property in the ontology schema by using Named Entity Recognition (NER) models. The results are promising for the some real datasets evaluated, considering that only the ontology schema is used to generate SPARQL queries.

Keywords: SPARQL generation. RDF schema. Named Entity Recognition. Query by Example.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

LCS	*Longest Common Subsequence*
NER	*Named Entities Recognition*
QA	*Question Answering*
Von-QBE	*Virtual Ontology Query by Example*
Von-QBNER	*Virtual Ontology Query by Named Entity Recognition*
tsv	*tab-separated values*

LISTA DE SÍMBOLOS

Q_N	Busca entrada pelo usuário em linguagem natural
Q_S	Consulta formal escrita em SPARQL
O	Ontologia RDF
θ	Limite de similaridade

SUMÁRIO

1 INTRODUÇÃO

Com a adoção das iniciativas de Dados Ligados (BERNERS-LEE, 2006), a Web evoluiu de uma rede de documentos interligados para uma rede onde dados e documentos estão conectados, resultando no que hoje chamamos de Web de Dados. Esta rede inclui uma grande variedade de dados, o que requer que padrões e protocolos sejam adotados, a fim de permitir uma melhor integração entre esses dados. Os datasets publicados como Dados Ligados, adotam o formato RDF e utilizam um vocabulário comum, o qual define a terminologia utilizada naqueles dados.

Tal evolução trouxe a necessidade de métodos intuitivos para acessar as fontes de Dados Ligados, uma vez que usar a linguagem padrão de consulta SPARQL pode ser muito difícil para usuários que não tenham conhecimento dessas tecnologias(YAHYA *et al.*, 2012). Vamos tomar como exemplo uma base RDF de dados de filmes do IMDB. Suponha que um usuário deseje realizar a seguinte busca: "Encontre os títulos dos filmes de comédia produzidos na Ásia Oriental e o nome de seu estúdio". Para obter esta informação, é necessário: i) conhecimento do vocabulário da fonte de Dados Ligados e; ii) conhecimento da linguagem de consulta SPARQL. Uma possível consulta para satisfazer a busca pode ser a seguinte:

```
1  SELECT DISTINCT ?titulo ?estudio WHERE {
2      ?filme a :Filme;
3          :titulo ?titulo;
4          :foiProduzidaPor ?y;
5          :pertenceAoGenero [ a :Comedia ] .
6      ?y :nome ?nome;
7          :estaLocalizada [ a :AsiaOriental ] .
8  }
```

Esta consulta, a qual envolve múltiplas junções, terminologias e artifícios da sintaxe, é complexa para um usuário sem conhecimento técnico elaborar. Pois requer um conhecimento na linguagem de consulta, além de uma grande familiaridade com a terminologia e organização da base de dados, a qual, em geral, não é esperada para usuário não-técnicos.

Neste trabalho, abordamos o problema de *QA* sobre dados RDF. Dada uma pergunta em linguagem natural Q_N e uma ontologia O (representada em RDF), nosso objetivo é transformar Q_N em uma consulta formal Q_S em SPARQL que seja capaz de representar a informação

requisitada expressa em linguagem natural em Q_N. O trabalho foca em consultas conjuntivas de seleção e projeção, não considerando agregações, disjunções e negações.

Uma quantidade considerável de trabalhos abordam o problema de *QA* sobre dados RDF. Para citar alguns, (USBECK *et al.*, 2015), (ARNAOUT; ELBASSUONI, 2018), (XU *et al.*, 2014), (YIH *et al.*, 2015), (UNGER *et al.*, 2012), (YAHYA *et al.*, 2012) e (LOPEZ *et al.*, 2009). Entretanto, eles possuem algumas características que podem dificultar sua utilização:

- Eles necessitam das instâncias dos dados para funcionarem, o que pode ser bastante custoso em cenários de *Big Data*, uma vez que demandaria muitos recursos computacionais, como memória e processamento;
- Em bases de dados legado, onde os dados estão desestruturados ou em formato não compatível com RDF, é necessário realizar algum tipo de processamento para estruturar o dado em um formato adequado;
- É necessário uma etapa de pré-processamento para construir alguma estrutura de dados auxiliar sobre as instâncias dos dados, como dicionários, índices, etc, o que pode ser trabalhoso em cenários onde há muita mutabilidade ou volatilidade nos dados;
- Algumas abordagens necessitam de conhecimentos externos a base de dados, como regras, *templates*, glossários ,etc.

Neste trabalho, propomos duas ferramentas para contornar tais características. Inicialmente, apresentamos o *Von-QBE*. Seu nome deriva do termo *virtualized* pois ele não necessita manipular instâncias dos dados, permitindo trabalhar com bases virtualizadas utilizando ferramentas como o Ontop(CALVANESE *et al.*, 2017). O *Von-QBE* necessita apenas do esquema da base RDF (i.e. seus metadados) para operar as consultas, sendo desnecessário conhecimento prévio da organização da base ou de tecnologias RDF. O *Von-QBE* permite que o usuário escreva sua busca utilizando linguagem natural ou palavras chaves, traduzindo-a para SPARQL. *Von-QBE* também ajuda o usuário a enriquecer sua busca interativamente, através da sugestão de exemplos.

Como o *Von-QBE* utiliza apenas o esquema da base RDF, ele possui duas principais limitações:

- Caso a base de dados não possua um esquema definido, ele não poderá trabalhar sobre ela, uma vez que ele não terá conhecimento sobre quais os conceitos presente na base ou como esses conceitos estão relacionados;
- É impossível identificar informações sobre as entidades presentes nas instâncias de dados.

a busca do usuário pode conter informações sobre as instâncias de dados. Por exemplo, a busca *filmes com Angelina Jolie*. Considere que o esquema apresente apenas os conceitos *Filme* e *Atriz*, e a propriedade *estrelando*, que relaciona os dois conceitos. Se utilizarmos apenas o esquema da ontologia, apenas *Filme* será identificado na busca.

Infelizmente, a estratégia utilizada no *Von-QBE* precisa do esquema, o que dificulta superar a primeira limitação. Entretando, propomos uma estratégia para contornar a segunda limitação.

Como uma primeira melhoria ao *Von-QBE*, apresentamos o *Virtual Ontology Query by Named Entity Recognition* (*Von-QBNER*). Ele conta com a adição de um modelo de *Named Entities Recognition* (*NER*) para auxiliar o processamento da busca do usuário, permitindo que entidades sejam reconhecidas em Q_N. Embora o modelo *NER* precise dos dados para serem treinados, não é necessário que os mesmos estejam presentes para que o modelo trabalhe, tornando-o assim uma boa opção.

1.1 Contribuições

As principais contribuições deste trabalho são:

- Um algoritmo para identificar, a partir de uma pergunta Q_N e uma ontologia O, os conceitos e propriedades de O que estão presentes em Q_N;
- Um algoritmo para identificar, a partir dos elementos de O presentes em Q_N, um subconjunto do esquema de O (chamado de fragmento) que relacione todos esses elementos;
- Um algoritmo para identificar, a partir do fragmento obtido, informações que possam ser utilizadas para que o usuário enriqueça sua busca;
- Um algoritmo que permite traduzir o fragmento de O em uma consulta formal Q_S escrita em SPARQL.

Todos os algoritmos apresentados, em conjunto, compõem o *Von-QBE* e o *Von-QBNER* e encontram-se implementados no GitHub[1] como um *framework* chamado de *Linked-Graphast*. Embora eles sejam componentes das ferramentas propostas neste trabalho, eles podem ser utilizados individualmente em outra aplicações, uma vez que cada um resolve um sub-problema atomicamente.

[1] https://github.com/InsightLab/linked-graphast

1.2 Publicações

Os seguintes artigos foram publicados durante o desenvolvimento deste trabalho. O primeiro artigo explica os algoritmos e experimentações do *Von-QBE*. O segundo artigo demonstra o *Von-QBE* como ferramenta, apresentando suas componentes, a interface gráfica e um exemplo de cenário de uso.

- PERES, L.; SILVA, T. L. C. da; MACEDO, J.; ARAUJO, D. Ontology-schema based query byexample. In: SPRINGER.International Conference on Conceptual Modeling. [S.l.], 2019. p.204–212.
- PERES, L.; SILVA, T. L. C. da; MACEDO, J.; ARAUJO, D. Virtualized ontology query byexample. In:ER Forum and Poster Demos Session 2019. [S.l.: s.n.], 2019. p. 148–153.

Houve também um terceiro artigo, que foi submetido para a *International Conference on Advanced Information Systems Engineering*, apresenta os algoritmos e experimentações do *Von-QBNER*. Entretanto, não foi aceito.

1.3 Estrutura da Dissertação

Esta dissertação está organizada da seguinte forma: no Capítulo 2 estão apresentados conceitos básicos sobre grafos, *Linked Data* e processamento de texto; o Capítulo 3 apresenta os trabalhos relacionados. Nos capítulos 4 e 5 apresentam, respectivamente, o *Von-QBE* e *Von-QBNER*, apresentando uma visão geral, seus algoritmos e experimentações realizadas. Por fim, no Capítulo 6 temos a conclusão deste trabalho, resumindo os resultados desta dissertação e apresentando os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

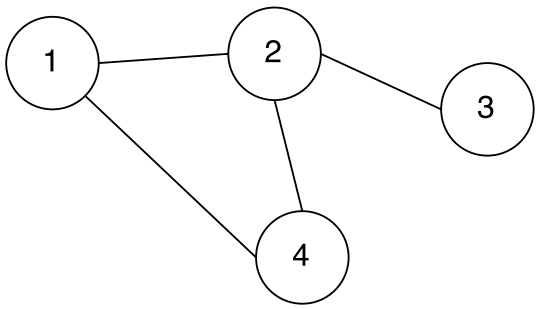
2.1 Introdução

Neste capítulo apresentaremos alguns conceitos essenciais para a compreensão do contexto em que esta dissertação está inserida, bem como da fundamentação necessária ao entendimento dos capítulos seguintes. Inicialmente, a Seção 2.2 apresenta alguns conceitos básicos sobre grafos; na Seção 2.3 apresentamos conceitos básicos de *Linked Data*, bem como RDF e SPARQL, por fim, na Seção 2.4, apresentamos as técnicas de processamento de texto que serão utilizadas neste trabalho.

2.2 Grafos

(BONDY; MURTY, 1982) apresenta uma definição tradicional para o conceito de grafo: um par ordenado $G = (V, E)$ composto de dois conjuntos: V, representando um conjunto de **nós** e E representando as **arestas**. Cada elemento de E é um par de dois elementos de V, representando uma relação (aresta) entre eles. A Figura 1 apresenta um grafo onde $V = \{1,2,3,4\}$ e $E = \{(1,2),(1,4),(2,4),(2,3)\}$. Nesse exemplo, o grafo é dito **não direcionado**, ou seja, as arestas do grafo representam uma relação simétrica entre os elementos (a relação (1,2) é a mesma de (2,1)). Em um grafo, suas arestas podem apresentar pesos, representando um valor para aquela relação.

Figura 1 – Exemplo de grafo não direcionado



Fonte: elaborado pelo autor

Existem diversos algoritmos que são aplicados sobre grafos, a fim de resolver diversos tipos de problemas (HARA *et al.*, 2015). Para este trabalho, focaremos nos seguintes problemas: menor caminho entre dois nós, Árvore Geradora Mínima, Árvore de Steiner. Também apresentamos a definição de fecho transitivo, que será utilizada na metodologia.

2.2.1 Menor Caminho entre dois nós

Um problema clássico em teoria dos grafos é, dado dois nós s e t presentes em um grafo G, qual o menor caminho (se houver) entre eles.

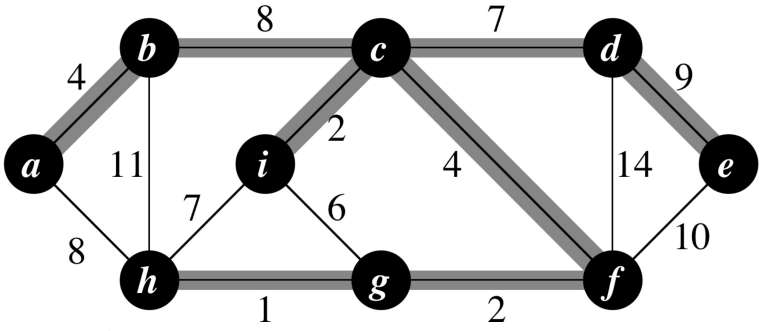
Existem diversos algoritmos para resolver esse problema. Dentre eles, o mais comum é o algoritmo de Dijkstra (DIJKSTRA, 1959). A partir de um nó s, ele explora os nós mais próximos a ele, percorrendo as arestas de menor peso. Quando um nó v é alcançado, registra-se o caminho percorrido até chegar nele. Quando um novo caminho até v é descoberto e ele é menor do que o já registrado, atualiza-se o registro do menor caminho. Quando o nó t é encontrado, o algoritmo encerra.

É possível, dependendo do grafo, que existam múltiplos caminhos mínimos entre s e t. Entretanto, o algoritmo de Dijkstra retorna apenas um deles (podendo ser o primeiro ou o último a ser encontrado, dependendo da implementação).

2.2.2 Árvore Geradora Mínima

(CORMEN *et al.*, 2009) define como Árvore Geradora Mínima de um grafo $G = (V,E)$, um grafo $G' = (V,T)$ onde T é um subconjunto acíclico de E. Para a árvore geradora mínima, o subconjunto acíclico de E é um subconjunto de arestas onde todos os os nós de V estão conectados por apenas um caminho e não existem ciclos no grafo. A Figura 2 apresenta um exemplo de uma árvore geradora mínima.

Figura 2 – Exemplo de árvore geradora mínima, onde as arestas que pertencem à árvore estão sombreadas..



Fonte: (CORMEN *et al.*, 2009)

O algoritmo tradicional para computar a árvore é o algoritmo de Prim (PRIM, 1957), o qual é muito similar ao algoritmo de Dijkstra. A partir de um nó s, o algoritmo de Prim vai explorando os nós mais próximos, percorrendo sempre as arestas de menor peso. Sempre que um

nó for alcançado por um caminho menor na árvore, ela é atualizada para conter esse novo menor caminho. Diferente do algoritmo de Dijkstra, o algoritmo de Prim não para quando encontra um certo nó t, mas sim quando ele percorre todas as arestas.

Um grafo pode conter mais de uma árvore geradora mínima. Como o algoritmo de Prim recebe um nó inicial s, é possível que, para cada diferente s, a árvore gerada seja diferente. No exemplo da Figura 2, a árvore foi criada a partir do nó c. Se o nó a, por exemplo, fosse escolhido, a árvore gerada conteria a aresta (a,h) estaria na árvore, enquanto a (b,c) não.

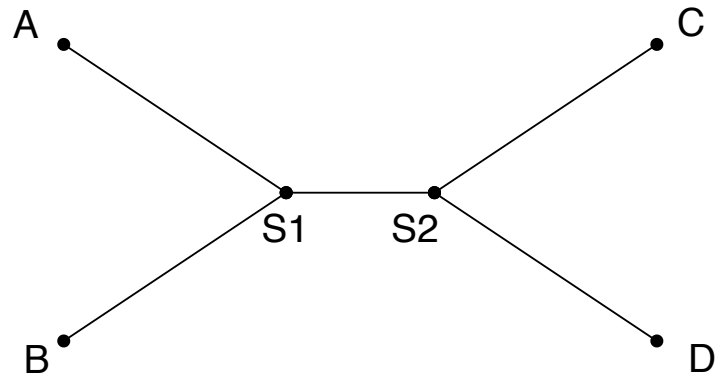
2.2.3 Fecho Transitivo

Dado um grafo $G = (V,E)$, chamamos de Fecho Transitivo de um nó v o conjunto de nós V' tal que, para cada $u \in V'$ existe um caminho de v à u. Normalmente o fecho transitivo é utilizado para resolver problemas de acessibilidade, ou seja, descobrir se um nó é alcançável por outro.

2.2.4 Árvore de Steiner

O problema da Árvore de Steiner requer encontrar o menor subgrafo que conecte um conjunto de nós (HWANG *et al.*, 1992). Para gerar tal subgrafo, muitas vezes são necessários nós que não estão no conjunto inicial. Esses novos nós são chamados de nós de Steiner. Como este problema é NP-Difícil, muitas das pesquisas focam em encontrar soluções aproximadas.

Figura 3 – Exemplo de Árvore de Steiner criada sobre os nós A, B, C e D.



Fonte: elaborado pelo autor

Na Figura 3 temos um exemplo de Árvore de Steiner para os nós A, B, C e D. Para conectar os nós A e B, é necessário utilizar o nó S1. Da mesma maneira, o nó S2 são necessários para conectar os nós C e D. Por fim, para conectar os nós A/B com C/D, é necessário conectar os

nós S1 e S2. Logo, a Árvore de Steiner gerada possui cinco arestas e dois nós de Steiner (S1 e S2).

2.3 Linked Data e RDF

A Web atual deixou de ser apenas um espaço global de documentos interligados e está se tornando um enorme espaço global de dados vinculados constituído de bilhões de triplas RDF que cobrem os mais variados domínios (HEATH; BIZER, 2011). Esta nova Web, denominada Web de Dados, visa pavimentar o caminho para a Web Semântica funcional, onde haverá a disponibilidade de uma grande quantidade de dados vinculados em formato RDF. A Web Semântica fornece tecnologias para efetivamente publicar, recuperar e descrever dados distribuídos na Web. A Web de Dados baseia-se nos princípios *Linked Data* delineados pelo diretor geral do W3C, o pesquisador Tim Berners-Lee. De fato, Linked Data é um conjunto de melhores práticas para publicação e conexão de dados estruturados na Web que se baseia em tecnologias da Web Semântica, e que permite reduzir a complexidade de integração de dados devido às ligações estabelecidas e descritas entre os conjuntos de dados. Desse modo, *Linked Data* tem o potencial de facilitar o acesso aos dados semanticamente relacionados, estabelecendo conexões explícitas entre conjuntos de dados.

Dentre as boas práticas e padrões utilizados na Web Semântica, está o uso do *Resource Description Framework* (RDF), para representar os modelos semânticos (Ontologias) e seu protocolo e linguagem de consulta, chamado SPARQL.

2.3.1 Ontologias

De acordo com (FENSEL, 2001), uma ontologia provê uma conceitualização explícita (ou uma meta-informação) que descreve a semântica dos dados. Elas são similares a esquemas de bancos de dados, com algumas diferenças:

- Uma linguagem para definir ontologias é sintática e semanticamente mais rica do que abordagens tradicionais de bancos de dados;
- A informação descrita na ontologia consiste em uma informação semi-estruturada e em linguagem natural, e não dados tabulares;
- Uma ontologia deve utilizar uma terminologia consensual, uma vez que ela pode ser utilizada para compartilhar informações com outras ontologias;

- Uma ontologia provê conhecimentos conceituais sobre o contexto da informação e não da estrutura de armazenamento propriamente dita.

Ontologias proveem um conhecimento comum de um domínio de informação, o qual permite a comunicação entre pessoas e aplicações. Permitir uma estrutura em comum é essencial, uma vez que as ontologias podem te um papel chave na troca de informações (FENSEL, 2001). Atualmente, a internet e a *World Wide Web* são as principais tecnologias de infraestrutura para troca de informações on-line, o que levou a um grande número de iniciativas na área para a criação de modelos para estruturação e semântica dos dados. Dentre esses modelos, encontra-se o RDF.

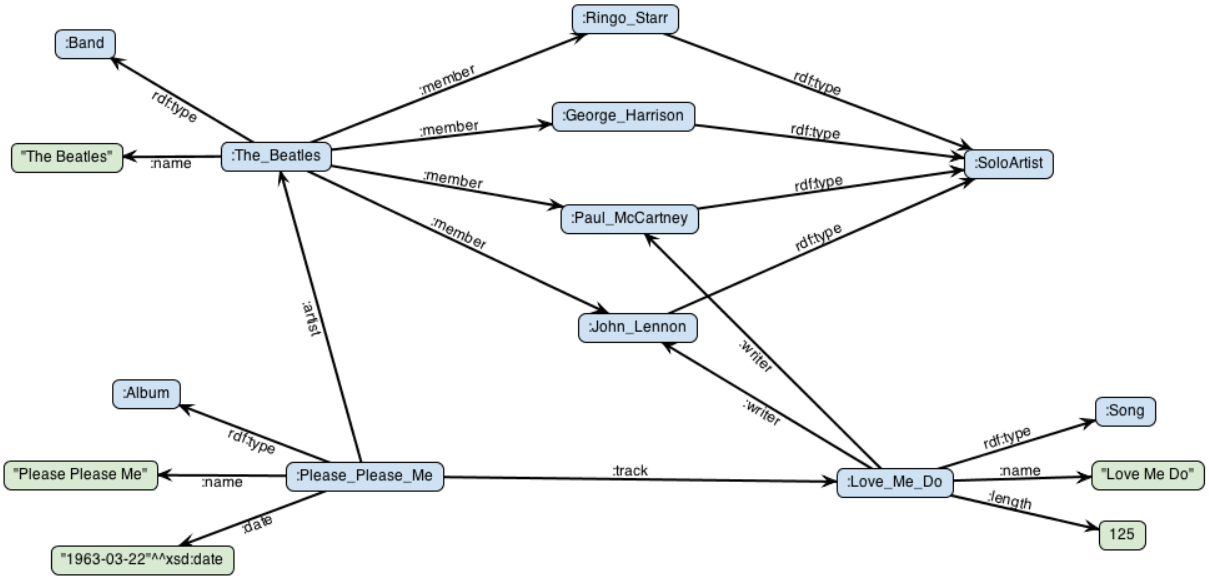
2.3.2 Resource Description Framework

A utilização um modelo de dados comum – RDF – torna possível a implementação de aplicações genéricas capazes de operar sobre a Web de Dados (HEATH; BIZER, 2011). O modelo de dados RDF (MANOLA *et al.*, 2004) descentralizado, baseado em grafo e customizável, possuindo um alto nível de expressividade e permitindo a interligação entre dados de diferentes conjuntos de dados. Ele foi projetado para a representação integrada de informações originárias de múltiplas fontes. Os dados são descritos na forma de *triplas*, da forma <sujeito, predicado, objeto>, onde o sujeito é uma URI, ou seja, o endereço de um documento na web; o objeto pode ser uma URI ou um literal (um valor constante como número, data, etc.) e o predicado é uma URI que define como sujeito e predicado estão relacionados. Por exemplo, de acordo com a Figura 4, a afirmação em português *"http://stardog.com/tutorial/Love_Me_Do foi escrita por http://stardog.com/tutorial/John_Lennon"* pode ser definida através de uma tripla RDF conforme ilustrado abaixo:

```
Sujeito: <http://stardog.com/tutorial/Love_Me_Do>
Predicado: <http://stardog.com/tutorial/writer>
Objeto: <http://stardog.com/tutorial/John_Lennon>
```

Quando temos um conjunto de triplas RDF que possuem sujeitos ou objetos em comum, a informação pode ser representada como um grafo, conforme podemos ver na Figura 4. No grafo RDF, os nós são sujeitos e objetos das triplas (podendo eles serem endereços ou constantes). A tripla propriamente dita é representada pela aresta, onde o tipo da aresta representa

Figura 4 – Exemplo de grafo RDF .



Fonte: https://www.stardog.com/tutorials/data-model/

a propriedade que conecta os dois elementos.

Embora não seja obrigatório, uma base de dados RDF pode apresentar um esquema: um conjunto de regras que definem as classes (conceitos) e propriedades (relações e atributos). Na Figura 4, temos que, por exemplo, *John Lennon* é uma *instância* da classe *SoloArtist*, ou seja, é uma entidade do conceito *SoloArtist*. Analogamente, podemos dizer que *Love Me Do* é uma instância de *Song*. A propriedade *writer*, que relaciona *John Lennon* e *Love Me Do*, é uma propriedade. No esquema, uma propriedade precisa ter duas informações: um domínio e uma imagem. Assim como na definição de funções matemáticas, o domínio representa que conceito poderá possuir a propriedade (sujeito da tripla), enquanto a imagem representa o valor em si da propriedade (objeto da tripla), que pode ser uma outra entidade (representando uma relação) ou um literal (representando um atributo). Ainda no mesmo exemplo, temos que a propriedade *writer* tem como domínio o conceito *Song* e, como imagem, *SoloArtist*.

2.3.3 SPARQL

Consultas à Web de Dados podem ser realizadas através da linguagem SPARQL (PRUD'HOMMEAUX *et al.*, 2008), que é a linguagem de consulta padrão da Web Semântica para recuperação de informações contidas em grafos RDF. O uso da linguagem de consulta de alto nível SPARQL abstrai detalhes da sequência de passos necessária para a execução de consultas sobre conjuntos de dados heterogêneos. Fontes *Linked Data* tipicamente fornecem um endpoint SPARQL, que é um serviço Web com suporte ao protocolo SPARQL. Esse serviço

possui uma URI específica para receber requisições HTTP com consultas SPARQL e retornar os resultados dessas consultas. Os resultados podem ter diferentes formatos, como, por exemplo, formatos XML, JSON, texto plano RDF/XML, NTriples, Turtle ou N3.

Uma consulta em SPARQL é baseada em *triple patterns*: triplas que podem conter variáveis em seu interior (sujeito, predicado ou objeto), onde as variáveis são iniciadas por "?"as entidades/propriedades (usualmente chamadas de IRIs) entre <> e os literais entre aspas. Os *triple patterns* são separados por ponto final (.). Como as IRIs podem ser muito longas, SPARQL tem suporte para a criação de prefixos. Por exemplo: podemos definir o prefixo *stardog:* para representar *http://stardog.com/tutorial/*. permitindo que a entidade *http://stardog.com/tutorial/Love_Me_Do* seja escrita apenas como *stardog:Love_Me_Do*. Nesse caso, na consulta SPARQL que utiliza um prefixo, não é necessário utilizar <> para representar as IRIs.

Suponha a seguinte consulta: "quais os membros da banda *The Beatles* que escreveram alguma música?". Para responder a consulta, precisaremos primeiro identificar os membros da banda *The Beatles*. Em seguida, precisamos saber qual música foi escrita por ele. Baseado na Figura 4, podemos escrever o seguinte SPARQL:

```
1  @prefix stardog: <http://stardog.com/tutorial/> .
2  SELECT ?musician ?song WHERE {
3      stardog:The_Beatles stardog:member ?musician .
4      ?song stardog:writer ?musician .
5  }
```

2.4 Processamento de Texto

A ideia de permitir um computador processar textos escritos em linguagem natural é tão antiga quanto a ideia dos próprios computadores (JURAFSKY; MARTIN, 2008). O estudo de processamento de textos em linguagem natural aborda métodos e algoritmos para construir modelos computacionais para a linguagem natural de maneira que permitam a comunicação entre humanos e máquinas (KHAN *et al.*, 2016). Existem diversas técnicas com diversas finalidades, porém, para este trabalho, abordaremos duas delas: similaridades entre palavras e *NER*.

2.4.1 Similaridade entre palavras

Medidas de similaridades textuais tem um papel importante em diversas pesquisas e aplicações, como recuperação de informações, desambiguação, clusterização de documentos, etc. (GOMAA; FAHMY, 2013). Encontrar a similaridade entre palavras é fundamental para a similaridade textual, sendo essa uma das primeiras etapas de processamento. As palavras podem ser similares de duas maneiras: léxica e semântica. Duas palavras tem uma similaridade léxica se elas possuem caracteres em comum (normalmente, levando-se em consideração a sequência de caracteres). A similaridade semântica dá-se quando as palavras, em um certo contexto, referem-se a mesma coisa.

Este trabalho utiliza a similaridade léxica entre palavras. Alguns exemplos de métricas para similaridade léxica são:

- ***Longest Common Subsequence* (*LCS*)**(CORMEN *et al.*, 2009): a similaridade entre duas palavras é baseada na maior subsequência em comum entre elas;
- **Jaccard**(JACCARD, 1901): são analisados os caracteres únicos em comuns entre as palavras. Essa métrica também pode ser aplicada a nível de frases, analisando, nesse caso, as palavras únicas em comum entre as frases;
- **N-gram**(BARRÓN-CEDENO *et al.*, 2010): a similaridade dá-se comparando as subsequências de tamanho n ($n-grams$) das duas palavras, dividindo o número de $n-grams$ iguais pelo total encontrado;
- **Jaro-Winkler**(WINKLER, 1999): atualmente conhecida como estado da arte, ela baseia-se no número e na ordem dos caracteres em comuns (Jaro) e na cadeia de prefixos (Winkler).

A Tabela 1 apresenta alguns exemplos de palavras e a similaridade entre elas de acordo com as métricas acima.

Tabela 1 – Exemplos de similaridades entre palavras de acordo com as métricas LCS, Jaccard, N-grams($n = 2$) e Jaro-Winkler.

Palavras	LCS	Jaccard	N-grams($n = 2$)	Jaro-Winkler
Harry Potter | Potter Harry	6	0.53	0.0	0.47
Quadrinhos | Revistas	2	0.	0.85	0.44
Petróleo | Petroleum	6	0.18	0.72	0.90
'Will' Smith | Will Smith	10	0.5	0.75	0.94
Universidade Federal | Federal University	8	0.47	0.14	0.68
Petróleo | Combustível	2	0	0.9	0.47

Fonte: elaborado pelo autor

2.4.2 Named Entity Recognition

NER é o ato de identificar entidades nomeadas, atribuindo rótulos a elas, como pessoas, lugares, organizações, etc., dentro de um texto. Aplicações de *NER* são comumente utilizadas como uma primeira etapa de processamento em recuperação de informações, co-referência, estruturação de dados, etc.

Existem diversas técnicas para realizar um processo de *NER* (NADEAU; SEKINE, 2007), os quais estão convergindo para o uso de modelos neurais (YADAV; BETHARD, 2019). Podemos modelar um problema de *NER* da seguinte maneira (SILVA *et al.*, 2019): dado um documento D onde $D = \{w_1, ..., w_n\}$, sendo w_i uma palavra, e um conjunto de rótulos $L = \{\lambda_j : j = 1...q\}$, o objetivo é atribuir, se necessário, um rótulo λ_j à cada w_i . Uma sequência consecutiva de w_i com um mesmo rótulo λ_j representa uma mesma entidade com rótulo λ_j.

Suponhamos a frase "Lucas trabalha no Insight Lab, localizado em Fortaleza/CE"e o conjunto de rótulos Pessoa, Organização e Local. Um modelo *NER* adequadamente treinado identificaria as seguintes entidades: "Lucas"(Pessoa), "Insight Lab"(Organização) e "Fortaleza/CE"(Local).

3 TRABALHOS RELACIONADOS

3.1 Introdução

O problema de *QA* têm sido bastante estudado pela comunidade científica. Os trabalhos a seguir apresentam uma parte do estado da arte nessa área, apresentando diversas abordagens para solucionar o problema. Na Seção 3.2, apresentamos estes trabalhos, em 3.3 apresentamos uma competição de *QA* e, em seguida, a Seção 3.4 conclui este capítulo.

3.2 Estado da Arte

Um *survey* apresentando diversas técnicas de processamento buscas sobre *Linked-Data* é apresentado em (BAST *et al.*, 2016).

Yaha et al. (YAHYA *et al.*, 2012) apresenta um método para traduzir perguntas em linguagens naturais em consultas formais SPARQL sobre fontes de *Linked-Data* baseado em programação linear inteira. Ele aborda o mesmo tipo de consultas que abordaremos neste trabalho: consultas conjuntivas de seleção, projeção e junção. Seu algoritmo segmenta a pergunta em frases menores, as quais são mapeadas em entidades da base de dados, bem como suas propriedades e, por fim, tais mapeamentos são convertidos em uma consulta SPARQL.

Em (UNGER *et al.*, 2012) é proposta uma abordagem baseada em uma análise sintática da pergunta para produzir a consulta SPARQL a partir de um conjunto pré-definido de *templates*. O template é então preenchido com as entidades presentes no texto da pergunta, as quais são identificadas utilizando cálculos estatísticos sobre as entidades presentes na base de dados.

Os autores de (XU *et al.*, 2014) propõem uma abordagem eficiente para modelar a intenção da pergunta do usuário em uma árvore de dependências frasais a qual é instanciada de acordo com a base consultada. Essa árvore de dependências é, então, convertida em uma consulta formal em SPARQL.

Hawk é introduzido em (USBECK *et al.*, 2015). Ele traz um grafo de predicados anotado com entidades da *Linked Data Web* utilizando etapas de processamento de linguagem natural como *part-of-speec* e *NER*. Hawk atribui um significado semântico para os nós do grafo e gera triplas representando as informações a ser buscadas de acordo com a pergunta do usuário. Essa striplas resultam em consultas SPARQL que contêm operadores de valores

(como filtros, por exemplo) e cláusulas de relacionamento (*triple patterns*). Hawk foi contruiído utilizando ferramentas de linguagem natural que são específicas para um certo domínio de dados (normalmente a DBPedia). Para usá-lo em qualquer outro domínio de dados (como dados de filmes ou relatórios policiais) é necessário encontrar (ou treinar) um modelo *NER* adequado para aquele contexto.

AquaLog (LOPEZ *et al.*, 2005) é um sistema portável de *QA* que recebe perguntas expressas em linguagem natural e uma ontologia e retorna as respostas a partir de um conjunto de possíveis marcações semânticas. AquaLog é, na prática, adequado para ontologias cujo domínio dos dado é pré-definido (ou para um conjunto deles), uma vez que as ferramentas de processamento de linguagem natural utilizadas precisam ter conhecimento da terminologia utilizada. PowerAqua (LOPEZ *et al.*, 2009) supera tais limitações provendo uma interface de linguagem natural que aceita perguntas sobre qualquer base de dados da web.

Arnaout et al. (ARNAOUT; ELBASSUONI, 2018) propõe um framework que permite o usuário buscar sobre uma base RDF utilizando tanto consultas expressas como *triple patters* como *triple pattenrs* extendidas com palavras chaves. Ele também propõe um modelo estatístico de ranqueamento dos resultados.

Os trabalhos (SINHA *et al.*, 2018; DUBEY *et al.*, 2016; OUKSILI *et al.*, 2018; CHANIAL *et al.*, 2018) utilizam palavras chaves para escrever consultas formais em SPARQL. Esses trabalhos tentam encontrar padrões sobre os grafos RDF a partir dessas palavras. Esses padrões são então utilizados para criar consultas formais sobre a ontologia RDF e obter seus resultados.

QWIOW (IZQUIERDO *et al.*, 2018) traduz uma busca em linguagem natural em uma linguagem de consulta intermediária, a qual pode ser traduzida em uma consulta relacional (SQL) ou SPARQL. Ela utiliza, em conjunto das instâncias dos dados, seu esquema para identificar como e quais elementos (sejam intâncias ou conceitos) podem ser utilizados para conectar as informações, geranto uma Árvore de Steiner.

OptiqueVQS é proposto em (SOYLU *et al.*, 2015). Ele apresenta elementos visuais para auxiliar o usuário a escrever sua consulta, apresentando seus conceitos e propriedades. Essa abordagem permite o usuário manipular elementos visuais para construir sua consulta. O OptiqueVQS utiliza o esquema da ontologia para identificar os elementos a serem apresentados na interface, enquanto os dados são virtualizados de uma base relacional utilizando o Ontop (CALVANESE *et al.*, 2017).

3.3 Competição de *QA*

Diversos trabalhos utilizam suas próprias fontes de dados, muitas das quais (devido a natureza privada dos projetos em que se encontram) não são abertos. Em 2011 foi criado a competição de *Question Answering over Linked Data*(UNGER *et al.*,), a qual ocorre tanto no *ISWC* quanto no *ESWC*, cujo objetivo é explorar soluções e promover pesquisas no contexto de *QA*. Os dados dessa competição (e suas edições com o passar dos anos) começaram a serem utilizados como *benchmarks* para validar os algoritmos da academia.

Todo ano é disponibilizado um novo dataset de perguntas. O dataset contém, para cada pergunta, sua representação em linguagem natural (em alguns anos, para abordar problemas soluções em diferentes línguas ou trabalhar com modelos multilinguísticos, as perguntas aparecem em vários idiomas), a consulta em SPARQL que representa a pergunta e o resultado da consulta. No exemplo abaixo, temos um recorte de uma pergunta do QALD-9, com a pergunta *List all boardgames by GMT.* e sua versão em português, a consulta SPARQL que a responde e parte do resultado da consulta.


```
{"id": "1",
  "question": [
    {"language": "pt",
      "string": "Lista todos jogos de tabuleiro de GMT. ",
      "keywords": "jogo de tabuleiro ,  GMT "},
    {"language": "en",
      "string": "List all boardgames by GMT.",
      "keywords": "boardgame, GMT"}
  ],
  "query": {
    "sparql": "PREFIX dbo: <http://dbpedia.org/ontology/>
    PREFIX res: <http://dbpedia.org/resource/>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    SELECT ?uri WHERE { ?uri dbo:publisher res:GMT_Games }"},
  "answers": [
    {"results": {
        "bindings": [
```

```
{"uri": {
    "type": "uri",
    "value": "http://dbpedia.org/resource/Washington's_War"
  }
},
{"uri": {
    "type": "uri",
    "value": "http://dbpedia.org/resource/Fields_of_Fire_(game)"
  }}]}}]}
```


3.4 Conclusão

Neste capítulo foram apresentadas algumas soluções para o problema de *QA*. A maior diferença entre estas abordagens e a proposta neste trabalho é que elas (exceto pelo OptiqueVQS) dependem das instâncias dos dados. No cenário de *Big Data*, manipular estas instâncias pode ser um grande problema: é necessário muito recurso computacional para armazenar os dados em memória, recursos que, muitas vezes, não são alcançáveis. Outro problema atrelado a isso é que muitas bases de dados legado não estão triplificadas no formato RDF, o que torna impraticável o uso de várias destas soluções.

No caso do OptiqueVQS, que apresenta os elementos em uma interface, quando o esquema da ontologia é muito complexo (seja por possuir muitos elementos ou as relações serem difíceis de serem interpretadas), o usuário poderá perder muito tempo tentando construir sua consulta.

Apresentamos também uma competição de *QA* no contexto de *Linked-Data* que vem sido explorada na comunidade acadêmica.

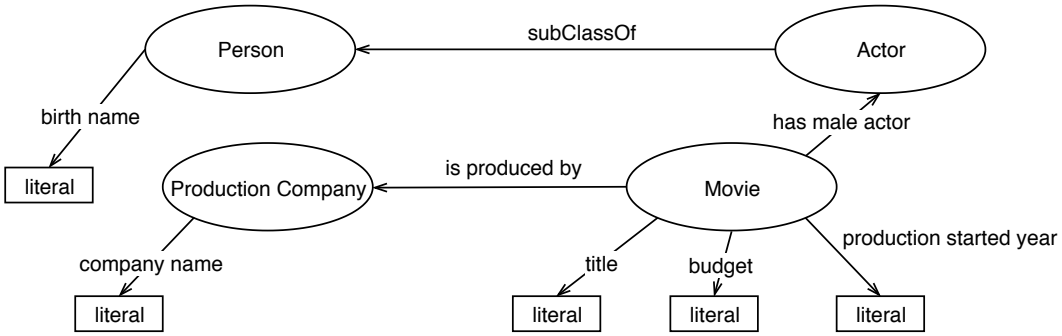
4 VON-QBE

4.1 Introdução

Neste capítulo, introduzimos a primeira ferramenta apresentada neste trabalho, o *Von-QBE*(PERES *et al.*, 2019a), apresentando seus componentes, seus algoritmos, as experimentações realizadas e o protótipo desenvolvido.

A partir de uma pergunta em linguagem natural Q_N, o *Von-QBE* utiliza uma combinação de métricas de similaridade entre palavras e algoritmos em grafos sobre a ontologia O para conseguir gerar a consulta formal Q_S em SPARQL. Suponha o esquema ontológico apresentado na Figura 5[1], representado como um grafo conforme (CONSORTIUM *et al.*, 2014), onde as classes são nós do grafo e as propriedades, arestas. Suponha também que um usuário inicie Q_N com *filme*. *Von-QBE* sugere exemplos de informações, a partir do esquema de O, para permitir que o usuário melhore Q_N até que representa a informação que ele ou ela deseje extrair. Por fim, *Von-QBE* transforma Q_N em uma consulta SPARQL e apresenta as respostas.

Figura 5 – Parte do esquema de ontologia de filmes do IMDB .



Fonte: adaptado de (PERES *et al.*, 2019a)

Figura 6 apresenta a arquitetura do *Von-QBE*. O sistema possui três componentes principais:

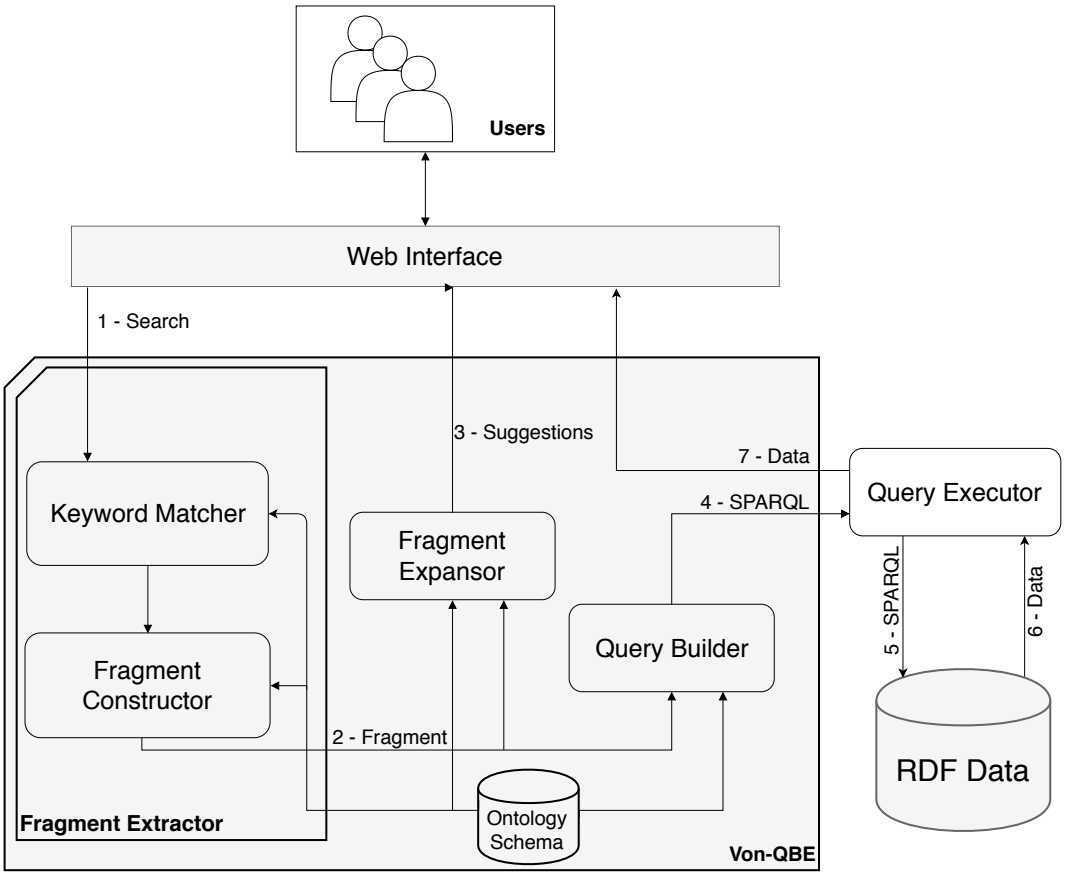
- **Fragment Extractor:** responsável por computar um subconjunto (fragmento) do esquema de O que represente as informações apresentadas em Q_N. Este módulo divide a tarefa em duas componente menores: o *Keyword Matcher*, responsável por identificar os elementos da ontologia O presentes em Q_N, utilizando métricas de similaridades entre palavras, e o *Fragment Constructor*, que identifica como relacionar esses elementos;
- **Fragment Expansor:** responsável por expandir o fragmento obtido pelo *Fragment Extractor*, identificando conceitos e propriedades conectadas a ele. Uma vez realizada a

[1] https://sites.google.com/site/ontopiswc13/home/imdb-mo

expansão, os elementos encontrados são apresentados para o usuário como sugestões, as quais ele ou ela pode utilizar para melhorar Q_N.

- **Query Builder:** responsável por transformar o fragmento obtido pelo *Fragment Extractor* em uma consulta SPARQL Q_S, a qual pode ser executada sobre os dados.

Figura 6 – Aquitetura do *Von-QBE*



Fonte: (PERES *et al.*, 2019b)

O restante do capítulo está organizado da seguinte maneira: na Seção 4.2 apresentamos o processo de extração de fragmentos do esquema da ontologia RDF (módulo *Fragment Extractor*); na Seção 4.2 explicamos o algoritmo de expansão do fragmento (módulo *Fragment Expansor*); na Seção 4.4 mostramos como é feita a geração da consulta SPARQL a partir do fragmento (módulo *Query Builder*). Experimentos são apresentados na Seção 4.5 e um protótipo da aplicação web é apresentada na Seção 4.6.

4.2 Extração de Fragmentos

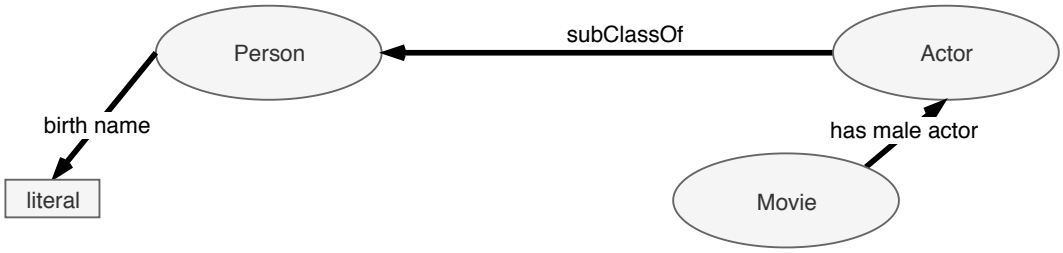
Como mencionado anteriormente, um *fragmento* de um grafo RDF corresponde às classes e propriedades do esquema da ontologia O que aparecem em Q_N. A extração de um

fragmento ocorre em duas etapas: identificação dos elementos mencionados em Q_N e conexão desses elementos no esquema.

Considere o esquema apresentado na Figura 5 e Q_N sendo ”Give the movie actors”. O módulo *Keyword Matcher* identifica os conceitos *Movie* e *Actor*, que são os termos mais similares presentes em Q_N. A partir desses conceitos, o módulo *Fragment Constructor* extrai o fragmento *Movie has male actor Actor*, uma vez que os conceitos *Movie* e *Actor* estão diretamente conectados no grafo.

Considere agora Q_N como ”Find the birth name of actors from movies”. O *Keyword Matcher* identificaria os conceitos *Movie* e *Actor* e a propriedade *birth name*. Entretanto, de acordo com o esquema, *birth name* não é uma propriedade de *Actor* e nem de *Movie*. Logo, o *Fragment Constructor* utiliza o conceito *Person* (de quem *birth name* é propriedade) e a propriedade *subClassOf* (que relaciona *Person* e *Actor*) para relacionar a propriedade *birth name* com o conceito *Actor*. Por fim, o fragmento é construído utilizando a propriedade *has male actor*, que relaciona *Movie* e *Actor*; *subClassOf*, que relaciona *Actor* e *Person* e *birth name*, que é propriedade de *Person*. O fragmento gerado é o em destaque na Figura 7.

Figura 7 – Fragmento do esquema de ontologia de filmes do IMDB para a busca *Find the birth name of actors from movies* .



Fonte: adaptado de (PERES *et al.*, 2019a)

Os módulos *Keyword Matcher* e *Fragment Construction* são apresentados, respectivamente, nas subseções 4.2.1 e 4.2.2.

4.2.1 Identificação de Elementos

O trabalho em (GOMAA; FAHMY, 2013) apresenta métricas de similaridades para serem aplicadas sobre dados textuais. Métricas como Jaccard (JACCARD, 1901) consideram apenas os caracteres envolvidos, não importando sua ordem. Portanto, tal métrica pode não ser adequada para essa aplicação, uma vez que um mesmo conjunto de caracteres pode gerar palavras com significados bem diferentes. Optamos por escolher métricas que consideram a

estrutura da palavra, como a *LCS* (CORMEN *et al.*, 2009), N-gram (BARRÓN-CEDENO *et al.*, 2010) e Jaro-Winkler (WINKLER, 1999). Adotamos Jaro-Winkler uma vez que ela é muito utilizada em diversas aplicações. Entretanto, o *Keyword Matcher* pode ser usado com qualquer métrica de similaridade que o usuário achar mais conveniente. Descartamos o uso de *exact match* uma vez que, como o objetivo é lidar com usuários reais, é comum haver erros de digitação em Q_N.

Algoritmo 1: Algoritmo Keyword Matcher

```
   Entrada: palavras, esquemaRDF, ρ
   Saída: elementos //elementos da ontologia
 1 elemmentos := {}; n :=(palavras.tamanho-1); conceitoComposto := ""
 2 para i := 0 até n-1 faça
 3     palavra = palavras[i]
 4     palavraTeste = conceitoComposto + palavra
 5     se possuiConceitoSimilar(palavraTeste, esquemaRDF) > ρ então
 6         conceitoComposto = palavraTeste
 7     senão
 8         conceito := esquemaRDF.obterConceitoMaisSimilar(conceitoComposto)
 9         elementos.add(conceito)
10         conceitoComposto = palavra
11     fim
12 fim
13 se possuiConceitoSimilar(conceitoComposto, esquemaRDF) > ρ então
14     conceito := esquemaRDF.obterConceitoMaisSimilar(conceitoComposto)
15     elementos.add(conceito)
16 fim
17 retorne elementos
```

Algoritmo 1 apresenta o algoritmo do *Keyword Matcher*. Ele recebe como entrada a lista de *palavras* de Q_N, o *esquema da ontologia* e um limite mínimo de similaridade ρ e retorna os elementos da ontologia (conceito e propriedades) que estão presentes na lista de palavras. Para cada palavra (linha 3) em Q_N, a variável *palavraTeste* recebe a palavra com as anteriores (armazenado na variável *conceitoComposto*) em Q_N que juntas eram similar a algum elemento no *esquema RDF*(linha 4). Em seguida, a linha 5 checa se há algum elemento no *esquema RDF* que seja similar à *palavraTeste* com um valor maior ou igual à ρ. Caso não haja, o algoritmo adiciona à lista de *elementos* o elemento no *esquema RDF* mais similar à *conceitoComposto* (linhas 8 e 9) e atualiza a variável *conceitoComposto* para a palavra do laço atual. A intuição por trás desta etapa é que a palavra atual pode iniciar uma nova sequencia de palavras que seja similar a outro elemento do esquema. O algoritmo precisa também checar se existe algum conceito

similar ao valor atribuído por último em *conceitoComposto* na linha 13. Se sim, tal elemento é adicionado na lista de *elementos* (linhas 14 e 15). Por fim, na linha 17, o algoritmo retorna os elementos identificados.

4.2.2 Construção de Fragmentos

Uma vez que os elementos de Q_N presentes em O foram identificados pelo Algoritmo 1, o *Von-QBE* precisa identificar um subgrafo (que chamamos de fragmento) da ontologia O que conecte todos os elementos. Para isso, o *Fragment Constructor* utiliza a solução proposta em (SAHA *et al.*, 2016), identificando o fragmento computando uma *Árvore de Steiner* a partir dos elementos identificados no *Keyword Matcher*. O Algoritmo 2 apresenta o *Fragment Constructor*.

Algoritmo 2: Algoritmo Fragment Constructor

Entrada: esquemaRDF, *elementos* //elementos da ontologia
Saída: fragmento
1 fecho := computarFecho(*elementos*, *esquemaRDF*)
2 fragmento := prim(fecho)
3 **retorne** *fragmento*

O Algoritmo 2 inicia computando o grafo do *Fecho Transitivo* (KOMPELLA *et al.*, 1993) (linha 1). Embora o fecho transitivo informe apenas se um nó u é alcançável por um certo nó v, utilizamos o algoritmo de Dijkstra para não só verificar a acessibilidade como também o menor caminho para ela. Logo, a variável $fecho$ é um subgrafo construído a partir dos caminhos mínimos entre as combinações de elementos dados na entrada (retornados pelo Algoritmo 1). Com o $fecho$ computado, já temos um grafo que conecta (se possível) todos os nós dados na entrada. Porém, esse grafo pode ter ciclos, o que é comum em ontologias RDF.

Suponha duas propriedades onde uma é a simetrica da outra, como *has male actor*, que conecta *Movie* com *Actor*, e *is male actor in*, conectando *Actor* com *Movie*. A presença dessa simetria permite que um menor caminho que envolva os nós *Movie* e *Actor* possa ser construído de duas maneiras: uma com *has male actor* e outra com *is male actor in*. Isso pode acarretar na geração de ciclos, uma vez que podemos ter um caminho utilizando uma propriedade e outro utilizando sua simétrica.

Para remover esses ciclos, o Algoritmo 2 computa a Árvore Geradora Mínima utilizando o algoritmo de Prim (PRIM, 1957) (linha 2). Esse algoritmo gera um subgrafo menor ou igual ao grafo informado (caso o grafo já seja uma árvore, ele é a própria resposta). Isso

significa que o subgrafo contém apenas a quantidade minimal de relações (arestas) para conectar todos os elementos retornados pelo Algoritmo 1. Um grafo pode ter múltiplas Árvores Geradoras Mínimas, entretanto o algoritmo de Prim retorna apenas uma delas, sem nenhuma heurística ou *ranking* para auxiliar em sua construção. Na linha 3, o Algoritmo 2 retorna o subgrafo, o qual é um fragmento da ontologia.

Uma vez que o fragmento é computado, *Von-QBE* pode usá-lo em dois outros módulos: 1) *Fragment Expansor*, onde o fragmento pode ser expandido com outros conceitos e propriedades da ontologia, a partir dos quais o usuário pode expandir Q_N; 2) *Query Builder*, onde o fragmento é traduzido para uma consulta Q_S em SPARQL.

4.3 Expansão de Fragmentos

Von-QBE sugere opções para o usuário expandir Q_N com conceitos e propriedades que são diretamente conectadas com o fragmento. O *Fragment Expansor* expande o fragmento com todas as arestas que entram ou saem dos nós do fragmento (exceto pelas arestas já presentes no fragmento). Vale lembrar que nossa ontologia O é representada como um grafo onde os nós são conceitos e as arestas, propriedades.

Suponha Q_N como ”Find the movies and their actors” e a ontologia em questão possui o esquema apresentado na figura 5. Os nós do fragmento derivado de Q_N são *Movie* e *Actor*. A partir de *Movie*, o *Fragment Expansor* pode identificar as seguintes propriedades: *title*, *budget*, *production started year*, *is produced by* e *has male actor*. Entretanto, *has male actor* já está presente no fragmento. Portanto, apenas as demais propriedades serão apresentadas como sugestão para o usuário expandir sua Q_N.

O conceito *Actor*, por sua vez, apresenta um caso particular. Sabe-se que a propriedade *has male actor* já está no fragmento. Nesse caso, resta apenas a propriedade *subClassOf* para o *Fragment Expansor* sugerir. Porém, quando um conceito é subclasse de outro, ele deve herdar as propriedades do conceito pai. Logo, ao invés de sugerir *subClassOf*, o *Fragment Expansor* sugere a propriedade *birth name*.

O Algoritmo 3 recebe como entrada o fragmento (gerado pelo Algoritmo 2) e o *esquemaRDF* da ontologia O e retorna uma lista de propriedades como sugestões para que o usuário possa expandir Q_N. Algoritmo 3 mantém todos os nós do fragmento em uma fila (linha 2). Para cada nó (ou conceito) (linha 4), o algoritmo recupera seus vizinhos no *esquemaRDF* (linha 8). Dois conceitos são tido como vizinhos se existe uma aresta (propriedade) que os

conecta. Se essa propriedade for *subClassOf*, o conceito vizinho precisa ser visitado depois, logo ele é incluído na pilha (linha 10). Para evitar um loop infinito, o algoritmo mantém um conjunto de nós já visitados. O algoritmo checa se o nó da pilha já foi visitado (linha 6). Se ele não tiver sido visitado, o algoritmo insere-o no conjunto de nós visitados (linha 7) e checa seus vizinhos.

Algoritmo 3: Algoritmo Fragment Expansor

```
   Entrada: fragmento, esquemaRDF
   Saída: sugestões
 1 sugestões := conjuntoVazio()
 2 filaDeNos := fragmento.obterNosEmFila()
 3 nosVisitados := conjuntoVazio()
 4 enquanto filaDeNos não for vazia faça
 5 |   no := filaDeNos.desenfileirar()
 6 |   se no ∉ nosVisitados então
 7 |   |   nosVisitados.add(no)
 8 |   |   para vizinho em obterVizinhos(no, esquemaRDF) e vizinho ∉ nosVisitados faça
 9 |   |   |   se no é subclasse de vizinho então
10 |   |   |   |   filaDeNos.enfileirar(vizinho)
11 |   |   |   senão
12 |   |   |   |   se vizinho ∉ fragmento então
13 |   |   |   |   |   sugestões.add(vizinho)
14 |   |   |   |   fim
15 |   |   |   fim
16 |   |   fim
17 |   fim
18 fim
19 retorne sugestões
```

4.4 Construção de Consultas

Uma vez que o fragmento é gerado a partir de Q_N (a qual pode ou não ter sido enriquecida com as sugestões do *Fragmento Expansor*), o *Von-QBE* pode transformá-lo em uma consulta SPARQL Q_S utilizando o módulo *Query Builder*. O módulo funciona da seguinte maneira: cada aresta do fragmento (gerado pelo Algoritmo 2) é adicionada como uma cláusula (ou *triple pattern*) e os nós que ela conecta são renomeados para variáveis. Uma vez que as propriedades do esquema podem conter múltiplas imagens e domínios, o *Query Builder* necessita também adicionar cláusulas para informar o tipo de conceito daquela variável. Todas as cláusulas são passadas para a biblioteca Apache Jena[2] a qual escreve Q_S utilizando a sintaxe da linguagem

[2] http://jena.apache.org

SPARQL.

Suponha Q_N "Find the birth name of actors from movies" e o fragmento com as seguintes relações: *Movie has male actor Actor*, *Actor subClassOf Person* e *Person birth name* (conforme destacado na Figura 5). Quando o *Query Builder* encontra uma aresta do tipo *subClassOf* que conecta dois nós *u* e *v*, ele substitui a aresta por novas arestas conectando *u* aos outros vizinhos de *v* (exceto aqueles que também são sublcasses de *v*). Considerando o esquema da Figura 5, o *Query Builder* gera as seguintes arestas: *A1: Movie has male actor Actor* and *A2: Actor birth name*. Nesse caso, a aresta *Actor subClassOf Person* foi removida, dando espaço para *Actor birth name*.

A aresta *A1* é uma relação conectando dois conceitos. Uma vez que ela é a primeira aresta a ser processada, existem duas novas variáveis (uma para *Movie* e uma para *Actor*). O *Query Builder* vai, para essa aresta, adicionar três cláusulas: a definição da variável do tipo *Movie*, a definição da variável do tipo *Actor* e a conexão dessas variáveis pela propriedade *has male actor*. Logo, a aresta *A1* corresponde as seguintes cláusulas:

TP1: ?Movie a imdb:Movie

TP2: ?Actor a imdb:Actor

TP3: ?Movie imdb:has_male_actor ?Actor

A aresta *A2* é uma propriedade que aponta para um literal, ou seja, um atributo (conhecido também, em RDF, como *DataTypeProperty*). Esse tipo de aresta também precisa utilizar duas variáveis, mas apenas a primeira é uma instância de um conceito, enquanto a segunda é um valor literal (número, string, data, etc.). Uma vez que a variável do tipo *Actor* já foi definida, o *Query Builder* necessita adicionar apenas mais uma cláusula:

TP4: ?Actor imdb:birth_name ?Actor_birth_name

Uma vez que todas as cláusulas foram geradas a partir das arestas do fragmento, o Apache Jena escreve a seguinte consulta SPARQL Q_S:

```
1  SELECT ?Movie ?Actor ?Actor_birth_name
2  WHERE{
3      ?Movie a imdb:Movie .
4      ?Actor a imdb:Actor .
5      ?Movie imdb:has_male_actor ?Actor .
```

```
6           ?Actor imdb:birth_name ?Actor_birth_name .
7       }
```

Algoritmo 4: Algoritmo Query Builder

Entrada: fragmento
Saída: consulta SPARQL

```
1  consulta = consultaVazia();
2  fragmentoSemSubclasses := substituirArestasDeSubclasse(fragmento);
3  para aresta em fragmetnoSemSubclasses faça
4      se aresta.fonte ∉ consulta então
5          consulta.adicionarClausulaDeTipo(aresta.fonte)
6      fim
7      se aresta.destino ∉ consulta então
8          consulta.adicionarClausulaDeTipo(aresta.destino)
9      fim
10     consulta.adicionarClausula(aresta);
11 fim
12 retorne consulta
```

O Algoritmo 4 mostra o algoritmo do *Query Builder*. Ele recebe de entrada o fragmento da ontologia (gerado pelo Algoritmo 2) e retorna a consulta em SPARQL Q_S. O algoritmo inicia com uma consulta vazia (linha 1) e em seguida substitui as arestas do tipo *subClassOf* (linha 2), conforme explicado anteriormente. O algoritmo então itera sobre as arestas do fragmento (linha 3), adicionando-as cláusulas na consulta (linha 10). O algoritmo checa, para cada nó da aresta, se o mesmo já foi definido na consulta (linhas 4 e 7). Caso não, é adicionada uma cláusula declarando o tipo daquele nó. Por fim, o algoritmo retorna a consulta SPARQL (linha 12).

4.5 Experimento

Nesta seção apresentamos a experimentação realizada com o *Von-QBE*. Nenhuma experimentação foi feita comparando-o com outras soluções apresentadas no Capítulo 3 uma vez que elas usam as instâncias do dados para melhorar sua performance, o que tornaria a comparação injusta. Antes do *Von-QBE*, até onde sabemos, não havia nenhuma outra solução baseada em esquema (exceto pelo OptiqueVQS(SOYLU *et al.*, 2015), porém ele não aceita busca textual, tornando-o incomparável). Contudo, experimentamos o *Von-QBE* utilizando duas bases de dados reais.

4.5.1 Dados utilizados nos experimentos

Nossa experimentação foi realizada com duas coleções de perguntas: uma sobre os dados da ontologia IMDB Movie Ontology[3], que corresponde a uma base relacional de filmes do IMDB, a qual foi virtualizada para RDF utilizando o Ontop(CALVANESE *et al.*, 2017) e possui 37 perguntas em palavras-chaves[4]; as perguntas da competição QALD[5] de *QA* sobre bases RDF. Ela contém um conjunto de perguntas sobre a base DBPedia(AUER *et al.*, 2007) anotadas com as consultas SPARQL que retornam suas respostas. Utilizamos as coleções de perguntas de treinamento do QALDs 5, 6, 7 e 9 (uma vez que o QALD 8 utilizou as mesmas perguntas do 7).

Exemplos de perguntas do QALD são "Was Margaret Thatcher a chemist?" e "List all boardgames by GMT". A primeira pergunta é do tipo *ASK*, a qual retorna uma resposta *booleana* (verdadeiro ou falso). Esse tipo de pergunta está fora do escopo do *Von-QBE*, uma vez que ele só responde perguntas do tipo seleção/projeção. Quanto a segunda pergunta, *boardgames* é um conceito na ontologia da DBPedia, porém *GMT* é uma entidade, a qual não está presente no esquema, apenas nas instâncias. Em relação as perguntas em palavras chaves do IMDB, perguntas como "Find writer with birth name[=;Cliver Barker]" estão fora do escopo, uma vez que elas envolvem conceitos que não aparecem no esquema disponibilizado. Perguntas que envolvem agregações, como contagens, também estão fora do escopo do *Von-QBE*. Após remover tais perguntas, nosso conjunto de perguntas de teste consiste com 12 perguntas do QALD-(5, 6, 7 e 9) e 29 perguntas do IMDB. Também não consideramos perguntas com conceitos que não existem no esquema da ontologia, uma vez que as consultas geradas não retornam resultados. Logo, o número de perguntas analisadas no QALD é bastante reduzido.

4.5.2 Métricas de Avaliação

Para avaliar a experimentação do *Von-QBE*, utilizamos duas métricas bem estabelecidas: *recall*(R) e precisão(P).

$$R = \frac{|\text{elementos corretos retornados pelo Von-QBE}|}{|\text{elementos esperados pelo } \textit{gold standard}|} \tag{4.1}$$

[3] https://sites.google.com/site/ontopiswc13/home/imdb-mo
[4] https://raw.githubusercontent.com/wiki/ontop/ontop/attachments/Example_MovieOntology/movieontology.q
[5] http://qald.aksw.org

$$P = \frac{|\text{elementos corretos retornados pelo Von-QBE}|}{|\text{elementos retornados pelo Von-QBE}|} \tag{4.2}$$

4.5.3 Resultados da Experimentação

As tabelas 2 e 3 apresentam os resultados dos experimentos utilizando as coleções de perguntas do IMDB e QALD, respectivamente. Para ambos os experimentos, utilizamos 0.9 como limite para a métrica de similaridade (ρ) no algoritmo do *Keyword Matcher*.

Nos experimentos do IMDB (Tabela 2) o *Von-QBE* teve uma boa performance, alcançando um alto valor de *recall* (0.96). Isso é bastante promissor, uma vez que a ferramenta utiliza apenas o esquema da ontologia para responder as perguntas. Entretanto, algumas perguntas apresentaram resultados de baixa precisão, como a pergunta 24: ”title movies company name production company located East Asia”, por exemplo, que pede o título dos filmes e nome das companhias produtoras localizadas no leste asiático. *Von-QBE* gera uma consulta SPARQL que projeta todas as propriedades e entidades utilizadas para construir a consulta, logo as URIs, títulos e nomes dos filmes e das companhias são retornados. Porém, existem, na base, filmes com ids diferentes mas com mesmo título, fazendo com que o Von-QBE retorne mais resultados do que o esperado, diminuindo a precisão.

Nos experimentos do QALD (Tabela 3) o *Von-QBE* teve uma performance inferior, atingindo um *recall* médio de de 0.53 e precisão média de 0.0047 . Isso ocorre pois as perguntas do QALD são perguntas fora do contexto do *Von-QBE*, que contém entidades em seu texto, como a pergunta 271 do QALD-9 ”Which awards did Douglas Hofstadter win?”. O SPARQL definido na coleção de perguntas retorna apenas os prêmios ganhos por *Douglas Hofstader*, enquanto o SPARQL gerado pelo *Von-QBE* retorna todos os prêmios. Isso acontece pois o *Von-QBE* utiliza apenas o esquema da ontologia e *Douglas Hofstader* é uma entidade, não um conceito. Isso traz um grande impacto aos resultados do QALD, uma vez que apenas uma pequena parcela das perguntas o *Von-QBE* consegue apresentar alguma resposta. Analisando apenas as perguntas que tiveram alguma resposta, ainda conseguimos resultados aceitáveis de *recall*, visto que o *Von-QBE* usa apenas o esquema. É necessário ainda melhorar a identificação de dos elementos, uma vez que é muito comum que usuários desejem inserir informações sobre as instâncias em suas perguntas.

Os dados da experimentação pode ser encotnrados no Github[6], bem como os módulos

[6] https://github.com/InsightLab/linked-graphast/tree/evaluation

Tabela 2 – Precisão e *Recall* para as perguntas da base do IMDB.

Id	Pergunta	Recall(R)	Precisão(P)
1	actress birth name[=;Pfeiffer, Michelle]	1.0	1.0
2	actor birth name[=;Aaker, Lee]	1.0	1.0
3	movie title[=;Finding Nemo]	1.0	1.0
4	TV Series title[=;24]	1.0	1.0
6	producer birth name[=;Silver, Joel]	1.0	1.0
7	film director birth name[=;Tarantino, Quentin]	1.0	1.0
8	editor birth name[=;Rawlings, Terry]	1.0	1.0
9	genre movie title[=;Finding Nemo]	1.0	1.0
10	genre tv series title[=;24] 24	1.0	1.0
11	budget movie title[=;Finding Nemo]	1.0	1.0
12	budget tv series title[=;24]	1.0	1.0
13	gross movie title[=;Finding Nemo]	1.0	1.0
15	production start year movie title[=;Finding Nemo]	1.0	1.0
16	production start year series title[=;24]	1.0	1.0
17	birth name actors movie title[=;Finding Nemo]	0.72	0.05
18	birth name male actors movie title[=;Finding Nemo]	1.0	0.05
19	birth name actress movie title[=;Finding Nemo]	1.0	1.0
20	birth name film director movie title[=;Finding Nemo]	1.0	1.0
21	birth name producer movie title[=;Finding Nemo]	1.0	1.0
22	birth name editor movie title[=;Finding Nemo]	1.0	0.25
23	company name production company movie title[=;Finding Nemo]	1.0	0.4
24	title movies company name production company located East Asia	1.0	0.0004
25	top 25 movie titles brute action imdb rating production started year	1.0	0.0
26	bottom 10 movie titles imdb rating	1.0	0.0
29	title company name movies production company located Eastern Asia production started year[>=;2000][<=;2010]	1.0	0.27
30	movie title company name brute action production company located Eastern Asia	1.0	0.09
32	movie title film director birth name[=;Tarantino, Quentin]	1.0	1.0
35	budget gross film director birth name producer birth name editor birth name company name movie title[=;Finding Nemo]	0.25	0.0
37	movie production company located Eastern Asia	1.0	1.0
-	**Média**	0.96	0.69

Fonte: traduzido de (PERES *et al.*, 2019a)

descritos.

Tabela 3 – Resultados dos experimentos para QALD-(5,6,7,9).A média é ponderada pelo número de questões responíveis

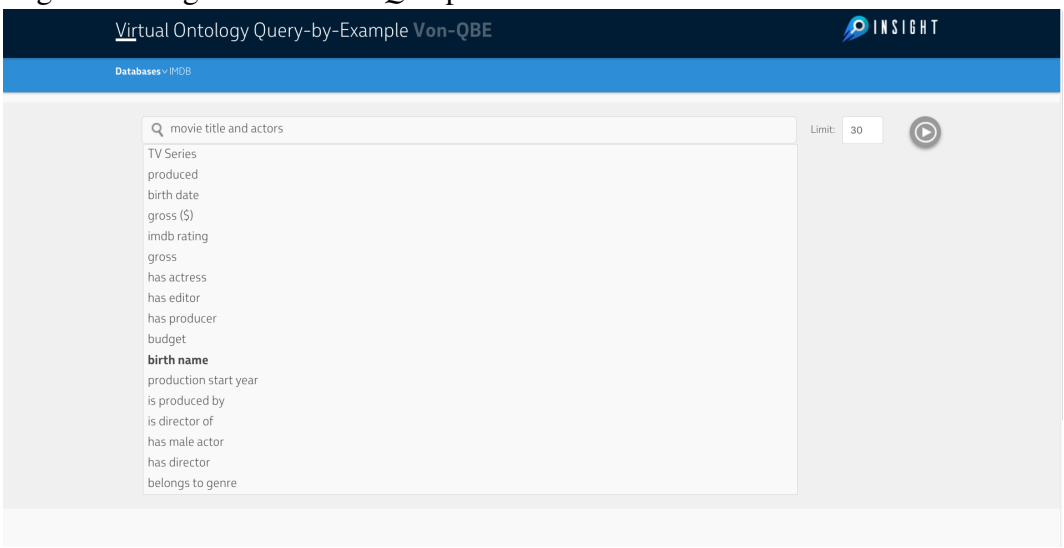
QALD	# Perguntas	# Perguntas de Seleção/Projeção	# Perguntas Respondíveis	Recall(R)	Precisão(P)
5	286	269	1	0.2	0.0001
6	335	308	5	0.55	0.001
7	215	193	2	0.39	0.00003
9	408	371	4	0.66	0.013
Média ponderada				0.53	0.0047

Fonte: traduzido de (PERES *et al.*, 2019a)

4.6 Protótipo

Um protótipo do *Von-QBE* é apresentado em (PERES *et al.*, 2019b) e está disponível no Github[7]. Um vídeo demonstrando o funcionamento da ferramenta está disponível no YouTube[8]. Para demonstrar o protótipo, utilizamos a mesma base do IMDB[9] utilizada como exemplo para os algoritmos e na experimentação.

Figura 8 – Sugestões do *Von-QBE*para a busca: *movie title and actors* .



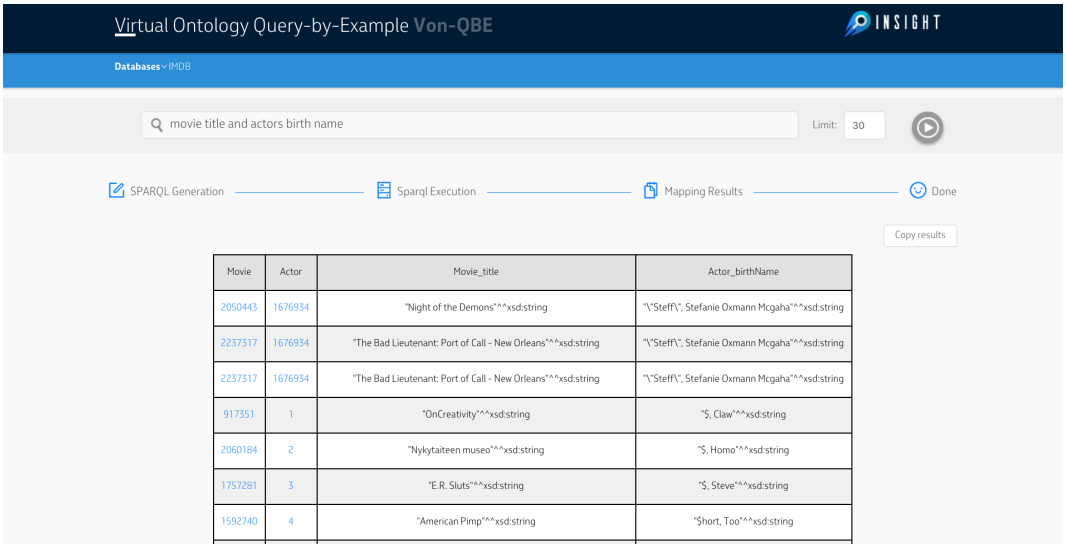
Fonte: (PERES *et al.*, 2019b)

A Figura 8 apresenta a interface do *Von-QBE*. Primeiro, o usuário deve entrar com a sua busca no campo textual. Uma vez que ele/ela tenha escrito algo, o *Von-QBE* mostra outros conceitos ou propriedades, a partir do esquema da ontologia, baseado no que o usuário escreveu. Na Figura 8 o *Von-QBE* sugere as sugestões para a busca *movie title and actors*. Em destaque, temos a propriedade *birth name*, a qual é um atributo de *Actor* (o qual é subclasse de *Person*). O

[7] http://github.com/insightlab/von-qbe
[8] https://youtu.be/ScXgGzhbx50
[9] https://sites.google.com/site/ontopiswc13/home/imdb-mo

usuário continua escolhendo sugestões até que termine de compor sua busca.

Figura 9 – Resultados para a busca *movie title and actors birth name*.

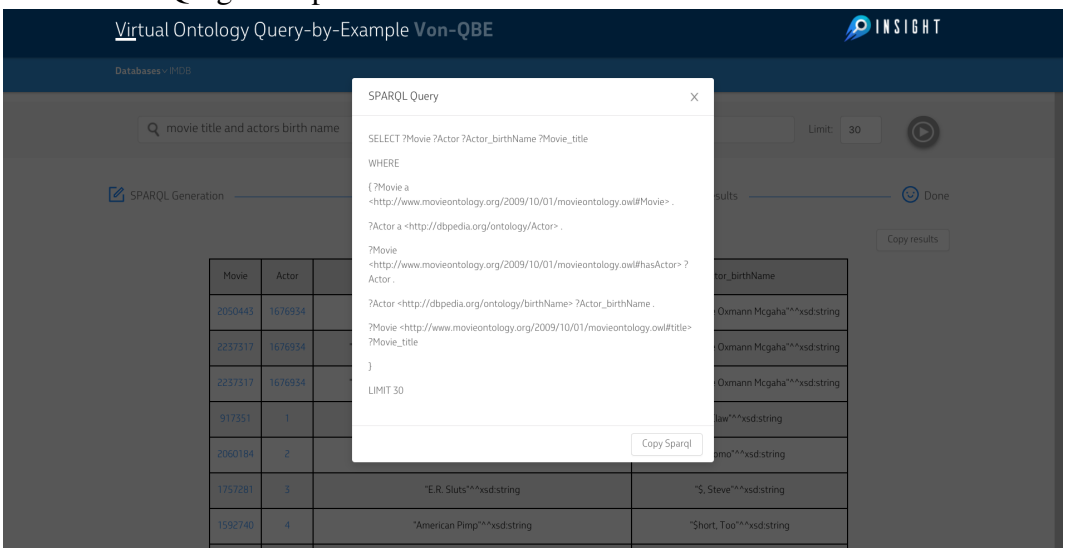


Movie	Actor	Movie_title	Actor_birthName
2050443	1676934	"Night of the Demons"^^xsd:string	"\"Steff\", Stefanie Oxmann Mcgaha"^^xsd:string
2237317	1676934	"The Bad Lieutenant: Port of Call - New Orleans"^^xsd:string	"\"Steff\", Stefanie Oxmann Mcgaha"^^xsd:string
2237317	1676934	"The Bad Lieutenant: Port of Call - New Orleans"^^xsd:string	"\"Steff\", Stefanie Oxmann Mcgaha"^^xsd:string
917351	1	"OnCreativity"^^xsd:string	"$, Claw"^^xsd:string
2060184	2	"Nykytaiteen museo"^^xsd:string	"$, Homo"^^xsd:string
1757281	3	"E.R. Sluts"^^xsd:string	"$, Steve"^^xsd:string
1592740	4	"American Pimp"^^xsd:string	"Short, Too"^^xsd:string

Fonte: (PERES *et al.*, 2019b)

Uma vez que o usuário tenha finalizado a escrita de sua busca, ele/ela pode limitar a quantidade de elementos a serem retornados da consulta ao banco, para evitar uma sobrecarga de dados no navegador. O *Von-QBE* vai então gerar a consulta SPARQL, a qual também pode ser obtida pela interface, conforme apresentado na Figura 10. O usuário também pode copiar os resultados da tabela para o *clipboard*, formatado como um *tab-separated values* (*tsv*).

Figura 10 – SPARQL gerado para a busca: *movie title and actors birth name* .



Fonte: (PERES *et al.*, 2019b)

5 VON-QBNER

5.1 Introdução

Neste capítulo, introduzimos a segunda ferramenta apresentada neste trabalho, o *Von-QBNER*, descrevendo seus componentes, algoritmos e os experimentos realizados.

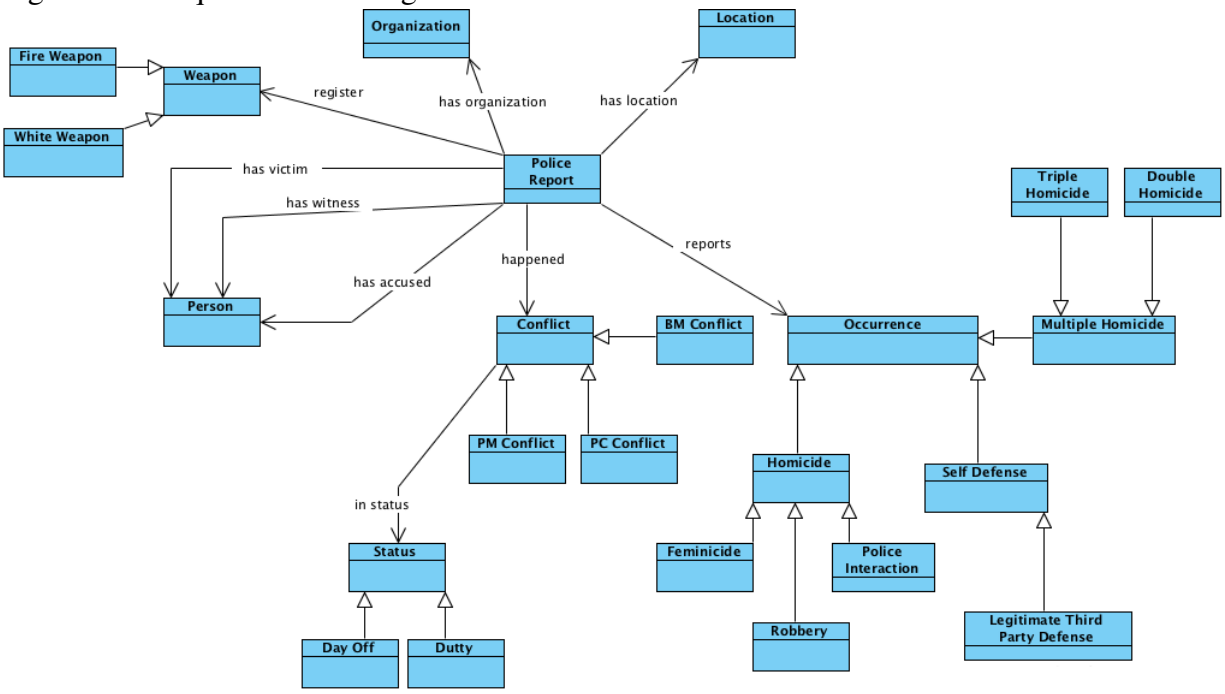
Dado o problema de *QA* "a partir de uma pergunta em linguagem natural Q_N e uma ontologia O, traduzir Q_N em uma consulta formal Q_S em SPARQL", vimos que o *Von-QBE* o resolve da seguinte maneira: Primeiro, o *Fragment Extractor* é responsável por, a partir de Q_N, identificar o fragmento do esquema de O presente na busca. O fragmento obtido pode ser então expandido pelo *Fragment Expansor* para apresentar sugestões de conceitos e propriedades que podem enriquecer Q_N. Quando o usuário terminar de escrever Q_N, o fragmento gerado será enviado para o *Query Builder* para ser transformado em uma consulta SPARQL Q_S. Por fim, a consulta é aplicada na base e os resultados são retornados ao usuário.

Essa abordagem tem dois problemas principais: primeiro, Q_N tem de ser uma *busca conceitual*, i.e., uma busca que contém apenas informações presentes no esquema (conceitos e propriedades), como "Find homicide(s) whith fire weapon"(ver a Figura 11, onde *Homicide* e *Fire Weapon* são conceitos da ontologia). Enquanto uma *busca não conceitual* não consiste apenas de conceitos e propriedades, mas também de informações sobre as instâncias de dados, como a consulta "Find homicide(s) with a 9mm", onde *9mm* é um tipo (ou uma instância) de *Fire Weapon*. Segundo, se houver mais de um menor caminho (ou caminho minimal) entre dois elementos da ontologia, o *Von-QBE* gera um fragmento (e uma consulta SPARQL) levando em consideração apenas um desses caminhos. Tome como exemplo a busca "Find the involved person of a Police Report": existem três maneiras de relacionar *Person* e *Police Report* (*has victim*, *has witness* e *has accused*), porém, o *Von-QBE* escolherá apenas um deles.

Para superar essas limitações, o *Von-QBNER* realiza melhorias nessa abordagem adicionando duas modificações: um modelo *NER* (o qual pode ser identificado na Figura 12), para identificar entidades em Q_N e o conceito ao qual ele se refere, e uma modificação no algoritmo de menor caminho, permitindo encontrar os caminhos *minimais*.

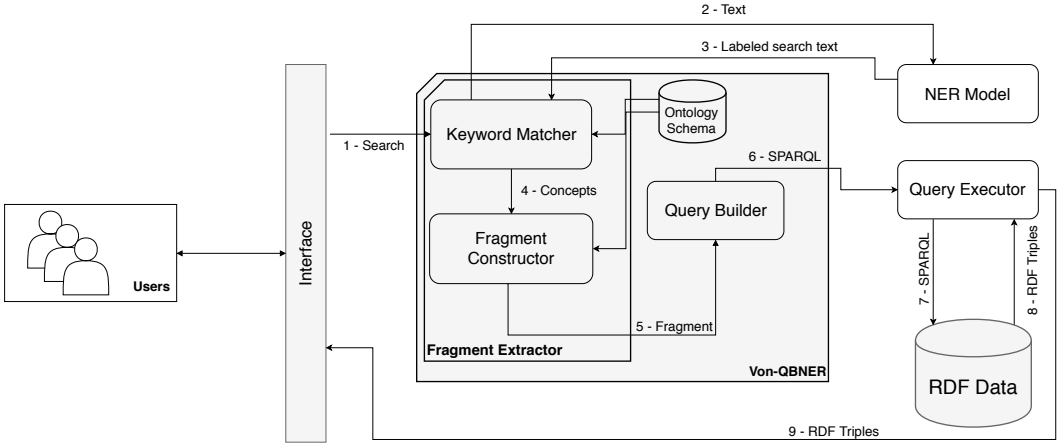
O capítulo está organizado da seguinte maneira: as Seções 5.2 e 5.3 apresentam, respectivamente, as modificações do *Keyword Matcher* e *Fragment Constructor*; a Seção 5.4 apresenta as modificações feitas no *Query Builder* para dar suporte ao novo fragmento, o qual agora pode conter caminhos minimais. Por fim, experimentos são apresentados na Seção 5.5.

Figura 11 – Esquema da ontologia de BOs.



Fonte: elaborado pelo autor

Figura 12 – Arquitetura do Von-QBNER.



Fonte: elaborado pelo autor

5.2 *Keyword Matcher*

Para suprir a limitação de processar apenas buscas conceituais, o *Keyword Matcher* utiliza um modelo *NER*. Esse modelo é responsável por, dada uma entidade, informar o conceito (ou *label*) associado a ela. Embora, para ser treinado, o modelo necessite de dados do domínio da informação sobre o qual ele atuará, uma vez treinado, esses dados não são mais necessários. Portanto, ele é uma alternativa viável para mantermos o princípio de não utilizar as instâncias de dados para responder às perguntas.

Mesmo utilizando um modelo *NER*, ainda existe um problema atrelado à etapa de

keyword matching: o ranqueamento. Desde às palavras identificadas aos resultados da consulta, o ranqueamento tem um papel importante para garantir um bom resultado (MENENDEZ *et al.*, 2019). Entretanto, para realizar o cálculo de ranking é necessário ter acesso às instâncias dos dados, o que é evitado neste trabalho. Existem abordagens como (LEAL *et al.*, 2018) que permitem realizar esses cálculos apenas analisando a topologia do grafo, o que no contexto deste trabalho seria ranquear os conceitos e propriedades do esquema da base. Entretanto, neste trabalho tais abordagens não chegaram a ser adotadas, contudo são enumeradas como trabalhos futuros.

Suponha Q_N como "Find homicides with a 9mm". A entidade *9mm* é uma instância do conceito *Fire Weapon* (Figura 11). O modelo *NER* classifica *9mm* com o conceito *Fire Weapon* como sua *label*, permitindo o *Keyword Matcher* reescrever Q_N como "Find homicides with Fire Weapon[9mm]". É necessário manter a informação da entidade *9mm*, uma vez que ela terá de ser utilizada para a geração do SPARQL no módulo *Query Builde*. Vale ressaltar que o modelo *NER* tem de ser adequado para o domínio do dado a ser trabalhado, uma vez que a má eficiência do modelo pode comprometer toda a abordagem. Uma vez que Q_N é reescrita, podemos utilizar o Algoritmo 1 apresentado anteriormente.

Como foi mencionado, existem diversas métricas de similaridades que podem ser utilizadas neste módulo, cada uma analisando aspectos diferentes das palavras. No *Von-QBE* optamos por utilizar o Jaro-Winker por ser conhecido como estado da arte. Entretanto, há um problema em usar essa métrica quando temos elementos com múltiplas palavras (que chamamos de *termos compostos*). Suponha a busca "Find people that have created comics". Na DBPedia[1], existe o conceito *comics creator*. Se utilizarmos Jaro-Winkler, a similaridade entre o que está presente em Q_N e o conceito *comics creator* é 0.1. Porém, se trocarmos a ordem das palavas em Q_N (i.e., *comics created*), a similaridade sobre para, aproximadamente, 0.94.

Como o objetivo final é um sistema que permita usuários realizarem buscas sobre as bases, sem nenhum conhecimento técnico, é necessário não só permitir erros ortográficos (como apresentado no *Von-QBE*), mas também permitir que o arranjo da ordem das palavras possa ser modificado (uma vez que usuário não precisa saber exatamente como o termo está escrito na base). Por isso, o *Von-QBNER* traz consigo uma nova função chamada de *Similaridade Permutada*, a qual tenta identificar a similaridade entre termos compostos desconsiderando a ordem das palavras. Seu algoritmo é apresentado no Algoritmo 5

[1] http://dbpedia.org

Algoritmo 5: Algoritmo Similaridade Permutada

```
   Entrada: a, b, // termos a serem comparados
   funçãoDeSimilaridade // função da métrica de similaridade
   Saída: valor da similaridade
 1 tokensA := obterTokens(a)
 2 tokensB := obterTokens(b)
 3 similaridadeTotal := 0
 4 para tA ∈ tokensA faça
 5 |   (tB,maiorSimilaridade) := obterTokenMaisSimilar(tokensB,tA, funçãoDeSimilaridade)
 6 |   tokensB.remover(tB)
 7 |   similaridadeTotal += maiorSimilaridade
 8 fim
 9 retorne similaridadeTotal / max(tokensA.length, tokensB.length)
10
```

O Algorithm 5 funciona da seguinte maneira: ele recebe dois termos a e b e uma *função de similaridade* (a qual, por padrão, utilizamos Jaro-Winkler). O primeiro passo é obter os tokens/palavras a partir dos termos dados como entrada (linhas 1 e 2). O algoritmo então percorre os tokens de a (lina 4). Para cada token ta, ele obtém o token mais similar tb dentre os tokens de b de acordo com a métrica utilizada (linha 5), recuperando tb e o valor da similaridade. Para evitar que um mesmo token tb seja utilizado mais de uma vez, ele é removido da lista de tokens de b (linha 6). Então o algoritmo soma a similaridade encontrada para o token ta. Após o algoritmo encerrar a iteração, ele dividirá a *similaridade total* pelo tamanho do termo com mais tokens (linha 9). Quando dois termos tem o mesmo número de tokens, o resultado é nada mais do que a média da similaridade total. Porém, para evitar que termos com diferentes números de tokens tenham uma alta similaridade, optamos por dividir a similaridade total pelo maior termo, o que diminui o valor final da similaridade.

Para melhor explicar a divisão aplicada na linha 9, suponhamos os seguintes termos: a sendo *Gerente Assistente* e, b, *Assistente do Gerente*. Existem apenas duas diferenças entre a e b: a ordem das palavras e a presença da preposição *do* em b. Ao executarmos o Algoritmo 5, ele identificará, em b, tokens com similaridade 1 para cada token em a, fazendo com que o valor da similaridade total seja 2. Se dividirmos pelo tamanho de a, teremos um resultado igual a 1, dizendo que os dois termos são iguais, o que não é verdade (nem sintática ou semanticamente). Logo, se dividirmos pelo comprimento de b, obteremos 0.66 de similaridade. Dessa forma, dividindo sempre pela maior termo (em número de palavras), garantimos que, em termos com diferentes números de palavras, a similaridade nunca será 1.

5.3 *Fragment Constructor*

Como apresentado no *Von-QBE*, o *Framgent Constructor* (Algoritmo 2) utiliza uma combinação dos algoritmos de menor caminho de Dijkstra e o da Árvore Geradora Mínima de Prim. *Von-QBNER* utiliza uma versão modificada com algoritmo de Dijkstra para recuperar todos os menores caminhos (ou *caminhos minimais*) entre os pares de nós dentre os encontrados em Q_N pelo *Keyword Matcher* (Algorimto 1).

Esta seção é dividida em duas: a primeira (Subseção 5.3.1) apresenta o algoritmo para encontrar os caminhos minimais entre dois nós; a segunda ((Subseção 5.3.2) apresenta como o *Von-QBNER* gera o fragmento.

5.3.1 Encontrando os caminhos minimais

Dado um grafo G e dois nós s e t, o algoritmo de Dijkstra computa o menor caminho entre eles através da expansão dos nós vizinhos de s até chegar a t. Para representar o caminho encontrado, o algoritmo mantém, para cada nó v pertencente ao caminho, uma referência ao seu nó pai (aquele que o precede no caminho). Quando v tem mais de um pai possível, o algoritmo mantém apenas um (normalmente o primeiro).

Neste trabalho, modificamos o algoritmo de Dijkstra para manter a referência para todos os nós pais possíveis (em um caminho minimal) de v, como apresentado no Algoritmo 6.

O Algoritmo 6 inicia criando um conjunto vazio de *nós visitados* (linha 1), o qual é usado para detectar se um nó já foi visitado. A variável *distancias* é um mapa (linha 2) que armazena, para cada nó, a menor distância de s até esse nó. A variável *pais* é um mapa (linha 3) que armazena, para cada nó, os possíveis pais em um caminho minimal a partir de s. O algoritmo uma fila de prioridade (linha 10) para controlar a ordem de visitação dos nós (linha 12), priorizando os mais próximos. O loop inicia desenfileirando o nó mais próximo u (linha 13) e checa se u é o destinho do algoritmo t (linha 14), encerrando o algoritmo caso a condição passe. Em seguida, o algoritmo explora os vizinhos v de u (linha 17), mas apenas os não visitados (linha 19). Se um novo caminho a partir de s para o nó atual v tem o mesmo custo do caminho minimal já conhecido (de s à v) (linha 20), então u é armazenado como um dos pais de v naquele caminho (linha 21). Essa é a principal diferença entre o algoritmo de Dijkstra e o Algoritmo 6. Se um novo caminho de s à v (utilizando a aresta (u,v)) possui um custo menor do que o já descoberto, o Algoritmo 6 procede como Dijkstra: o conjunto de pais de v contém apenas u

Algoritmo 6: Algoritmo de Caminhos Minimais

```
   Entrada: G,// um grafo
   s, //o nó inicial
   t//o nó de destino
   Saída: Lista de caminhos minimais
 1 nósVisitados := conjuntoVazio()
 2 distancias := Map()
 3 pais := Map()
 4 foreach v ∈ G.nos do
 5 |   distancias[v] := ∞
 6 |   pais[v] := conjuntoVazio()
 7 end
 8 nosVisitados.add(s)
 9 distancias[s] = 0
10 Q := filaDePrioridade()
11 Q.enfileirar(s)
12 while Q ≠ ∅ do
13 |   u := Q.desenfileirar()
14 |   se u = t então
15 |   |   break
16 |   fim
17 |   para v ∈ G.vizinhos(u) faça
18 |   |   peso := G.obterAresta(u, v).peso
19 |   |   se v ∉ nosVisitados então
20 |   |   |   se distancias[u] + peso = distancias[v] então
21 |   |   |   |   pais[v].add(u)
22 |   |   |   fim
23 |   |   |   se distancias[u] + peso < distancias[v] então
24 |   |   |   |   pais[v] := conjunto(u)
25 |   |   |   |   distancias[v] := distancias[u] + peso
26 |   |   |   fim
27 |   |   |   Q.enfileirar(v)
28 |   |   fim
29 |   fim
30 |   nosVisitados.add(u)
31 end
32 caminhosMinimais := computarCaminhos(s, t, pais)
33 retorne caminhosMinimais
```

(linha 24) e o custo do caminho minimal entre s e v é o custo do caminho de s à u mais o custo da aresta (u, v) (linha 25). Em seguida, o algoritmo enfileira v na fila de prioridade. Finalmente, o algoritmo computa e retorna (se existir) todos os caminhos minimais (linha 32) de s à t utilizando os mapeamentos de *pais* descobertos.

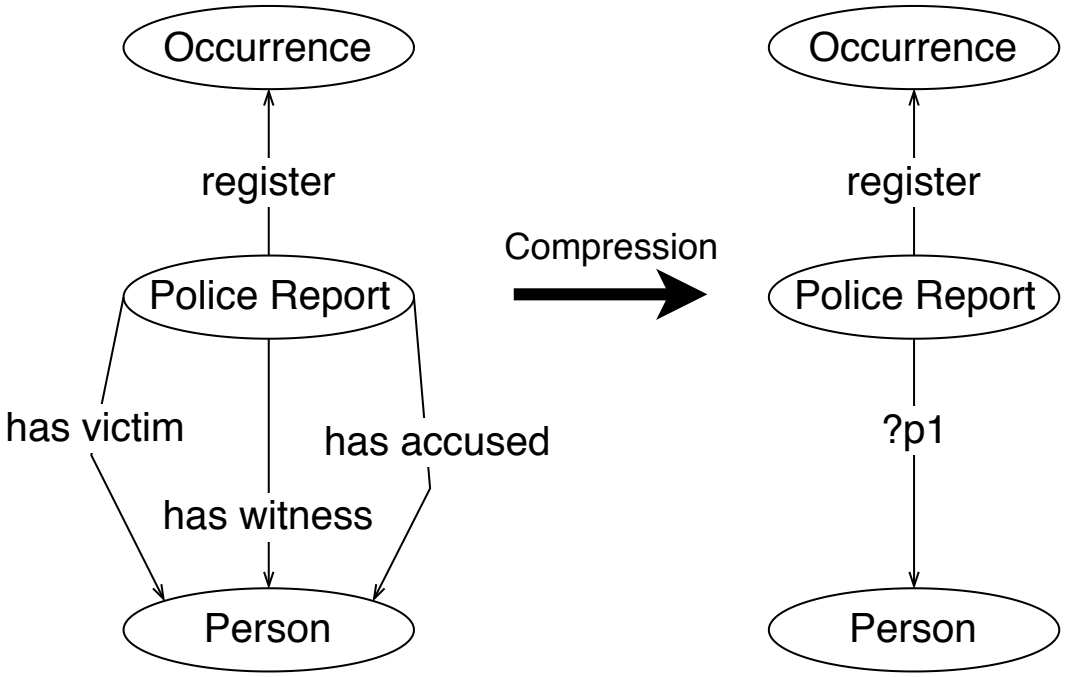
5.3.2 Construindo o fragmento

Para construir o fragmento, o *Von-QBNER*, assim como o *Von-QBE*, computa o grafo *Closure* a partir dos elementos identificados pelo *Keyword Matcher* (Algoritmo 1) e, em seguida, encontrar a Árvore geradora mínima (utilizando o algoritmo de Prim). Porém, o *Von-QBNER*

utiliza os caminhos minimais (retornados pelo Algoritmo 6) ao invés do menor caminho. Além desses algoritmos, o *Von-QBNER* adiciona uma nova etapa na construção do fragmento.

Imagine que um usuário insira a busca "Find for each occurrence the person(s) related"sobre o esquema da ontologia apresentado na Figura 11. O *Keyword Matcher* identifica os conceitos *Person* e *Occurrence*. O *Fragment Constructor* como apresentado no lado esquerdo da Figura 13, utilizando todos os caminhos minimais entre *Person* e *Occurrence* retornados pelo Algoritmo 6. Os três caminhos possuem semânticas distintas: uma representa uma vítima, a outra uma testemunha e, por fim, um acusado. Porém, na busca inserida pelo usuário, ele/ela não está interessado na semântica do envolvimento da pessoa na ocorrêcia. Uma vez que *Person* pode relacionar-se com *Occurrence* de três (ou mais) maneiras diferentes, *Von-QBNER* nomeia a propriedade que relaciona-os como uma variável, gerando o fragmento comprimido apresentado no lado direito da Figura 13.

Figura 13 – Compressão de fragmento gerado utilizando caminhos minimais.



Fonte: elaborado pelo autor

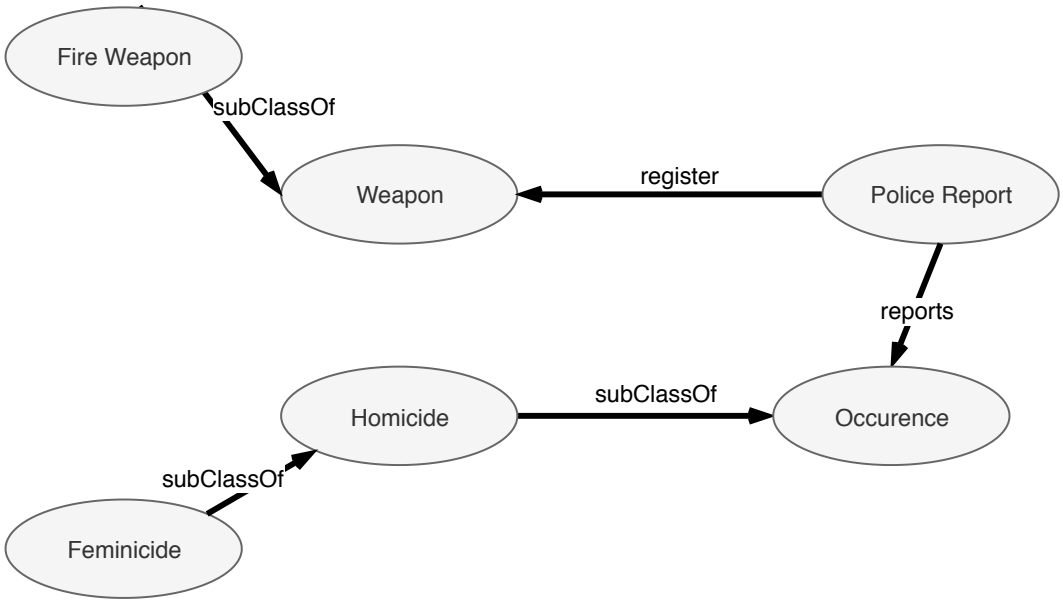
5.4 *Query Builder*

Von-QBNER utiliza o fragmento extraído para gerar a consulta formal Q_S em SPARQL. Assim como no *Von-QBE*, o *Query Builder* é o módulo responsável por esta tarefa. Todo o fluxo do *Von-QBE* é aproveitado nesta etapa: cada aresta do fragmento é adicionada

como uma cláusula (ou *triple pattern*) e os nós que ela conecta são renomeados para variáveis. Como as propriedades do esquema podem conter múltiplas imagens e domínios, é necessário adicionar cláusulas para informar o tipo de conceito daquela variável. Porém, como no fluxo do *Von-QBNER* há a informação da entidade (a qual foi identificada pelo modelo *NER* no *Keyword Matcher*), o *Query Builder* precisará utilizar esta informação.

Considere Q_N "Find the Police Reports with Feminicide that have a fire weapon[9mm]". *Von-QBNER* identifica o fragmento presente na Figura 14. Considerando o esquema apresentado na Figura 11, o *Query Builder* gera as seguintes arestas: *E1: Police Report reports Feminicide* e *E2: Police Report registries Fire Weapon*.

Figura 14 – Fragmento obtido da ontologia de BOs para Q_N *Find the Police Reports with Feminicide that have a fire weapon[9mm].*



Fonte: elaborado pelo autor

Após a execução do Algoritmo 4, as seguintes cláusulas serão geradas:

TP1: ?police_report a :PoliceReport

TP2: ?Feminicide a :Feminicide

TP3: ?police_report :reports ?Feminicide

TP4: ?fire_weapon a :FireWeapon

TP5: ?police_report :registries ?fire_weapon

Como a busca original Q_N havia sido reescrita no *Keyword Matcher* (devido a uma entidade reconhecida pelo modelo *NER*), o *Query Builder* tem a informação de que a instância

de *Fire Weappon* tem de ter relação com *9mm*. Porém, não sabe-se que propriedade pode ser utilizada para caracterizar uma instância de *Fire Weapon* como uma *9mm*, logo, o *Query Builder* adiciona um filtro (representado pela cláusula *FILTER*) informando que *?fire_weapon* precisa ter alguma propriedade cujo valor contenha *9mm*, gerando as seguintes cláusulas:

TP6: ?fire_weapon ?fire_weapon_property ?fire_weapon_value

TP7: FILTER(contains(lcs(str(?fire_weapon_value)), "9mm")

Uma vez que todas as cláusulas foram geradas, o Apache Jena gera a consulta em SPARQL Q_S:

```
SELECT ?police_report, ?Feminicide,
?fire_weapon, ?fire_weapon_value
WHERE{
    ?Feminicide a :Feminicide .
    ?police_report a :PoliceReport ;
        :reports ?Feminicide .
    ?fire_weapon a :FireWeapon ;
    ?fire_weapon_property ?fire_weapon_value .
    ?police_report :registries ?fire_weapon .

    FILTER(contains(
        lcs(str(?fire_weapon_value)), "9mm")
        )
}
```

5.5 Experiment

Nós avaliamos o *Von-QBNER* comparando o seu ganho em relação ao *Von-QBE*. Utilizamos dois conjuntos de perguntas sobre datasets reais: o QALD-9 (apresentado na experimentação do *Von-QBE*, Capítulo 4 Seção 4.5), que contém perguntas sobre a DBPedia, e um conjunto de perguntas criadas sobre um dataset de Boletins de Ocorrências de Fortaleza, Ceará, Brasil. Optamos por utilizar apenas um dos conjuntos do QALD neste experimento para

podermos focar mais nas perguntas respondidas e suas particularidades e optamos pelo QALD-9 por ser o mais recente deles.

5.5.1 Dados utilizados nos experimentos

Conforme apresentado anteriormente, o dataset QALD contém perguntas sobre a DBPedia, acompanhadas de consultas SPARQL que geram o *golden standart*. Para este dataset, o *Von-QBNER* utiliza, como modelo *NER*, o Wikifier[2], o qual foi gerado sobre os dados de conceitos e propriedades da Wikipedia.

O outro dataset utilizado é um benchmark com perguntas criadas sobre um conjunto de Boletins de Ocorrências. Foram utilizados em torno de 1000 boletins, cujos dados foram processados usando o Human NERD(SILVA *et al.*, 2019) e estruturados em RDF. Em seguida foram elaboradas, por especialistas do domínio, 20 perguntas dentro do contexto dos dados. Para cada pergunta foi criada uma consulta em SPARQL que representa a informação requisitada na pergunta, criando assim o benchmark. O *Von-QBNER* utiliza o modelo *NER* utilizado no Human NERD, uma vez que o mesmo foi treinado para o contexto dos boletins.

5.5.2 Métricas de Avaliação

Para comparar o *Von-QBNER* com o *Von-QBE*, utilizamos as mesmas métricas da experimentação do *Von-QBE*: *recall* (R) e precião (P).

$$R = \frac{|\text{elementos corretos retornados pelo Von-QBE}|}{|\text{elementos esperados pelo } \textit{gold standard}|} \tag{5.1}$$

$$P = \frac{|\text{elementos corretos retornados pelo Von-QBE}|}{|\text{elementos retornados pelo Von-QBE}|} \tag{5.2}$$

5.5.3 Experimentação dos Boletins de Ocorrência

A Tabela 4 apresenta os resultados dos experimentos para a base de Boletins de Ocorrência. Na maioria das perguntas, *Von-QBNER* supera o *Von-QBE* em termos de precisão, uma vez que o modelo *NER* identifica as entidades e suas labels, permitindo o *Von-QBNER*

[2] http://wikifier.org/

construir uma consulta SPARQL mais precisa, filtrando as instâncias pelas entidades encontradas, enquanto o *Von-QBE* normalmente detecta apenas os conceitos (como nas perguntas de 4 à 12, onde ele identifica apenas o conceito *Police Report*). Assim, *Von-QBE* retorna todas as instâncias dos conceitos identififcados, o que gera um alto valor para o *recall*. Note que as perguntas possuem termos sublinhados, os quais são informações sobre as instâncias de dados (ou entidades reconhecidas pelo modelo *NER*) e necessitam resultados mais precisos do que apenas retornar todas as instâncias dos conceitos. Por isso, *Von-QBNER* alcança resultados mais precisos.

Von-QBNER não obtem um bom *recall* nas perguntas 2, 3, 8 e 9. Para a 2 e 3, o modelo *NER* classifica *fire weapon* como uma instância do conceito *Fire Weapon* (o mesmo na pergunta 3, onde *white weapon* foi classificada como *White Weapon*), fazendo com que o *Von-QBNER* retorne apenas instâncias de dados que contenham as palavras *fire* e *wepon* em alguma de suas propriedades. Entretanto, ele deveria retornar todas as instâncias de *Police Reports* que estejam relacionadas com *Fire Weapon*, independente de suas propriedades (analogamente, para a questão 3, *White Weapon*).

Em relação as perguntas 8 e 9, o modelo *NER* classifica corretamente as entidades *killed by two men on a motorcycle* e *at Frotinha from Antonio Bezerra* com as labels *Executon* e *LOC*, respectivamente. Como *Von-QBNER* não realiza nenhuma etapa de pré-processamento para remover *stopwords* das entidades e nem das perguntas, as perguntas 8 e 9 obtiveram baixos valores de *recall*. Por exemplo, vários documentos da base contém *two men over/sit on/ride/on a motorcycle*, porém, o SPARQL gerado para a pergunta 8 retorna apenas resultados com, exatamente, *two men on a motorcycle*. O mesmo aplica-se para a pergunta 9, onde o *Von-QBNER* falha em obter todos os resultados corretos.

5.5.4 Experimentação do QALD-9

A Tabela 5 sumariza os resultados obtidos pelo *Von-QBNER* e *Von-QBE* para as perguntas do QALD-9. Novamente, as perguntas que envolvem agregação, ordenação e limites estão fora do escopo deste experimento. Na Tabela 5 analisamos apenas as questões onde, pelo menos uma das abordagens (*Von-QBNER* ou *Von-QBE*), retornou uma resposta correta. Para cada pergunta, utilizamos o spaCy[3] para remover os pronomes e artigos. Nesses experimentos, analisamos o impacto nos resultados quando variamos o limite de similaridade (θ) do *Keyword*

[3] https://spacy.io/

Tabela 4 – *Recall* e Precisão para Von-QBE (R^- e P^-) e Von-QBNER (R e P) utilizando o dataset de Boletins de Ocorrências. As entidades reconhecidas estão sublinhadas.

Id	Pergunta	R^-	P^-	R	P
1	Police Reports with person and homicides	1.0	1.0	1.0	1.0
2	Police Reports that reports homicides and fire weapons	1.0	1.0	0.05	1.0
3	Police Reports that reports homicides and white weapons	1.0	1.0	0.07	1.0
4	Police Reports with trampling victim	1.0	0.0	1.0	1.0
5	Police Reports with bullet homicide	1.0	0.53	1.0	1.0
6	Police Reports that have a female victim	0.0	0.0	1.0	1.0
7	Police Reports that reports knife homicide	0.82	0.13	1.0	1.0
8	Police Reports where someone was killed by two men on a motorcycle	1.0	0.0	0.5	1.0
9	Police Reports where the victim passed away at Frotinha from Antonio Bezerra	1.0	0.01	0.0	0.0
10	Police Reports with subject named Marcell	1.0	0.01	1.0	1.0
11	Police Reports with subject know as Warlock	1.0	0.0	1.0	1.0
12	Police Reports with homicides in Fortaleza	1.0	0.03	1.0	1.0
13	Cases with bullets	0.0	0.0	1.0	1.0
14	Cases where the victim was rescued at IJF Centro	0.0	0.0	1.0	1.0
15	Cases with subject named Francis	0.0	0.0	1.0	1.0
16	Cases with gunshots	0.0	0.0	1.0	1.0
17	Trampling victim	0.0	0.0	1.0	1.0
18	Cases where the victim was taken to Municipal Hospital of Caucaia	0.0	0.0	1.0	1.0
19	Cases where the victim was stabbed	0.0	0.0	1.0	1.0
20	Cases of fatal tampling	0.0	0.0	1.0	1.0
-	**Mean**	0.54	0.18	0.83	0.95

Matcher. Os valores de θ utilizados foram 0.9 (representando o uso de métrica de similaridade) e 1 (representando *exact match*). Não utilizamos valores menores de θ uma vez que os resultados possuíam muitos ruídos e as consultas geradas não traziam respostas corretas.

Tabela 5 – Média do *Recall* e Precisão para o Von-QBE (MR^- e MP^-) e para o Von-QBNER (MR e MP) usando o dataset QALD 9, variando o limite de similaridade θ entre 0.9 e 1

θ	# Perguntas Respondidas	MR^-	MP^-	MR	MP
0.9	13.0	0.434	0.001	0.58	0.008
1.0	23.0	0.77	7.9E-4	0.47	0.002

Utilizando $\theta = 0.9$, *Von-QBNER* supera o *Von-QBE*, tendo 15% a mais de *recall* e sendo oito vezes mais preciso. Entretando, com $\theta = 1$, *Von-QBE* recupera mais informações que o *Von-QBNER*, embora o segundo seja, ainda sim, mais preciso. Em ambos os casos, *Von-QBNER* apresenta ser mais preciso, graças ao modelo *NER*. A seguir, analisamos ambos os

experimentos, explicando seus resultados.

5.5.4.1 Utilizando similaridade ($\theta = 0.9$)

Quando $\theta = 0.9$, o Algoritmo 1 (*Keyword Matcher*) consegue recuperar conceitos e propriedades que são similares aos escritos, como um mesmo verbo com a conjugação em tempo ou número diferente. Um exemplo é a pergunta 37 "Who developed Minecraft". Não existe uma propriedade *developed* na DBPedia, mas existe *developer*. Logo, *Von-QBE* e *Von-QBNER* retornam todos os desenvolvedores. Para a pergunta 19 ”Who created the comic Captain America”, o *Keyword Matcher* identificou que *created comic* é suficientemente similar a *comics creator*, de acordo com a Similaridade Permutada (Algoritmo 5), utilizando Jaro-Winkler e θ, o que resulta, em ambas as abordagens, em um SPARQL que retorna todos as instâncias *comics creator*.

Em algumas dessas perguntas o modelo *NER* não identificou nenhuma entidade, fazendo com que o *Von-QBE* e o *Von-QBNER* performassem da mesma maneira. Entretanto, ocorreram casos em que o modelo *NER* encontrou uma entidade, porém o conceito atribuído a ela não estava de acordo com o escrito na ontologia. Um exemplo é a pergunta 20 ”Give me the Apollo 14 astronauts”: o modelo *NER* identifica *Apollo 14* como entidade de *Space Mission*, nomeando-a com o rótulo *SpaceMission*, sem espaçar as duas palavras. Ao calcular a similaridade entre os dois termos, por serem termos com número de palavras diferentes, a similaridade é baixa (0.44).

Quando o modelo *NER* identifica e classifica uma entidade com um rótulo presente no esquema da ontologia, o *Von-QBNER* torna-se mais preciso, como na pergunta 72 ”Which pope succeeded John Paul II?”. Enquanto o *Von-QBE* apenas gera um SPARQL que retorna todas as instâncias de *Pope* (alcançando 1.0 de *recall* e 0.002 de precisão), *Von-QBNER* identifica, utilziando o modelo *NER*, que *John Paul II* é uma entidade do conceito *Pope*, logo ele retorna todas as instâncias de *Pope* que possuem, em algum atributo, a informação *John Paul II* (alcançando 1.0 de *recall* e 0.14 de precisão). Claro que a interpretação feita pelo *Von-QBNER* não está correta, mas ainda sim é uma melhoria em relação ao *Von-QBE*.

Existem também perguntas que o *Von-QBE* não consegue responder, como a pergunta 109 ”Give me all books written by Danielle Steel”. Apenas o *Von-QBNER* por causa da métrica de similaridade: *Von-QBE* identifica *Steel* como o conceito *Kasteel* e gera um SPARQL que tenta relacionar esse conceito com *Book* (o que também foi identificado devido à similaridade) e *Writer*.

Von-QBNER, por sua vez, identifica *Danielle Steel* como instância de *Writter*. Porém, ao tentar relacioná-la com *Book*, ele não usa *Writer* como conceito, mas sim *writer* como propriedade (uma vez que ambos possuem a mesma escrita). Essa escolha é feita pois a propriedade *writer* está mais próxima de *Book* do que o conceito *Writer*. O SPARQL gerado consegue retornar algumas instâncias corretas de *Book*, mas não todas.

Por fim, existem perguntas que não são possíveis de responder devido à organização do esquema, como a pergunta 117 ”Which ships were called after Benjamin Franklin?”. O *Von-QBNER* identifica que *Benjamin Franklin* refere-se ao conceito *Person* e tenta relacioná-lo com o conceito *Ship*. Porém, de acordo com a consulta provida no QALD 9, é necessário utilizar a propriedade *shipNamesake* para relacionar os elementos, a qual não está presente no esquema da DBPedia, apenas nas instâncias dos dados. Portanto, o *Von-QBNER* falha ao tentar relacionar os conceitos. Enquanto isso, como o *Von-QBE* retorna apenas todas as instâncias de *Ship*, ele consegue responder a pergunta (embora com baixa precisão).

5.5.4.2 Utilizando exact match ($\theta = 1$)

Quando o *Keyword Matcher* utiliza $\theta = 1$, ele identifica apenas *exact matches*, ou seja, propriedades e conceitos que foram escritos exatamente como estão no esquema. No experimento anterior, a pergunta 37 ”Who developed Minecraft” foi respondida pois *developed* era similar a *developer*. Utilizando *exact match*, isso não acontece. Embora essa abordagem seja mais restritiva, ela permite que mais perguntas sejam respondidas na experimentação. Muitas perguntas contêm palavras que são similares a elementos do esquema, mesmo sem necessariamente estarem representando um desses elementos. Um exemplo é a pergunta 63 "In which films directed by Garry Marshall was Julia Roberts starring?". Com $\theta = 0.9$, *In* é identificado como *INN* (International Nonproprietary Name). Tais erros repercutem na geração do fragmento, gerando fragmentos muito grandes (uma vez que os conceitos são bem distantes), os quais produzem SPARQL que, geralmente, não têm respostas. Utilizando o *exact match*, reduzimos esses eventos, resultando em mais perguntas sendo respondidas.

Algumas perguntas não são respondidas pelo *Von-QBNER* devido á vários erros de classificação do modelo *NER*. Quando o modelo *NER* analisa a pergunta 203 "In which country is Mecca located", por exemplo, ele classifica *coutry is* como um *Single*, o que não é adequado no contexto da frase, fazendo com que o SPARQL gerado retorne todos as instâncias de *Single* que contém *country is* em algum atributo. Enquanto isso, o *Von-QBE* recupera todas as instâncias

de *Country*.

Existem também perguntas que, para serem respondidas, envolvem repostas de múltiplos conceitos, como uma conjunção. Na pergunta 247 "In which programming language is GIMP written?", ambas as abordagens apresentem o mesmo *recall*, sendo o *Von-QBNER* é mais preciso devido ao modelo *NER*. Enquanto o *Von-QBE* apenas identifica o conceito *Programming Language*, o modelo *NER* identifica que *GIMP* é um *Software*. Porém, o modelo também diz que *programming language* é um *TopicalConcept*, o qual, devido a falta de espaço entre as palavras, não existe na ontologia. Logo, o *Von-QBNER* retorna todas as instâncias de *Software* que contenham *GIMP* em um de seus atributos. Nenhuma abordagem conseguiu *recall* de 1.0 pois a resposta esperada envolve tanto *Software* quanto *Programming Language*.

O *Von-QBE* não consegue responder algumas das perguntas devido ao fragmento ser construído utilizando apenas o menor caminho. Na pergunta 273 ”In what city is the Heineken brewery?”, tanto o *Von-QBE* quanto o *Von-QBNER* identificam os conceitos *City* e *Brewery*. Devido ao algoritmo de Dijkstra, o *Von-QBE* escolhe um menor caminho para conectar os conceitos, porém o caminho escolhido não responde a pergunta. Em contrapartida, devido ao algoritmo de caminhos minimais, o *Von-QBNER* gera um SPARQL que relaciona os dois conceitos, independente da propriedade que os conecta.

5.5.4.3 Considerações Finais

Os resultados apresentados mostram que o uso de *exact match* permite que mais perguntas sejam respondidas e que, neste cenário, o *Von-QBE* tem melhor *recall*, embora não seja tão preciso. Porém, muitas perguntas não foram respondidas devido a erros de classificação do modelo *NER*. Outro fator foi a organização do esquema: em alguns casos, não foi possível relacionar os conceitos corretamente devido a maneira como o esquema estava organizado, o que implicava na geração de fragmentos que, ao serem traduzidos para SPARQL, geravam consultas sem respostas.

Vale ressaltar que nosso objetivo é prover uma aplicação real, que possa ser útil aos usuários em contextos reais, e não um algoritmo específico para datasets como o QALD. Os usuários podem cometer erros de escrita, por exemplo, ou não saber a terminologia exata utilizada no esquema da ontologia. Além disso, é esperado o uso de um modelo *NER* o mais eficiente possível para o contexto do dado a ser trabalhado. O modelo *NER* do Wikifier não pareceu adequado para o dataset do QALD 9, embora, até onde sabemos, não existe um modelo

específico para este dataset. Acreditamos que, com um modelo *NER* mais adequado para o contexto dos dados, o *Von-QBNER* possa mostrar-se ainda mais promissor.

6 CONCLUSÕES E TRABALHOS FUTUROS

Parte deste trabalho já foi validade pela comunidade em (PERES *et al.*, 2019a) e (PERES *et al.*, 2019b). Esta dissertação abordou o problema de *QA*: dada uma pergunta em linguagem natural Q_N e uma ontologia RDF O, nosso objetivo é traduzir Q_N em uma consulta formal Q_S em SPARQL que seja capaz de representar a informação requisitada expressa em Q_N. A dificuldade do problema encontra-se em conseguir traduzir a pergunta, que está em linguagem natural, em uma consulta formal utilizando a linguagem SPARQL, que pode ser entendida por bancos de dados RDF.

No Capítulo 2 apresentamos conceitos básicos para melhor entender sobre *Linked Data* e algumas ferramentas de processamento de texto necessárias para o entendimento da dissertação. No Capítulo 3, apresentamos os trabalhos relacionados, descrevendo brevemente como eles resolvem o problema abordado. Nos capítulos 4 e 5 foram apresentados, respectivamente, o *Von-QBE* e o *Von-QBNER*, explicando seus algoritmos e as experimentações realizadas para validar suas contribuições.

6.1 Objetivos Alcançados

No Capítulo 1 desta dissertação, listamos as contribuições que desejamos alcançar. Durante o trabalho apresentado, procurou-se alcançar todos os objetivos específicos, objetivando solucionar o problema. Dessa forma, foi proposta uma técnica para transformar a busca em linguagem natural Q_N em uma consulta formal Q_S em SPARQL (*Von-QBE*). Apresentamos também algumas limitações desta técnica e propomos melhorias para suprir tais limitações (*Von-QBNER*).

Nos experimentos foi destacado as limitações de ambas as abordagens, ressaltando as dificuldades de trabalhar apenas com o esquema da base RDF, bem como a importância de que o mesmo esteja bem definido. Também ressaltamos, para o *Von-QBNER*, a importância do uso de um modelo de *NER* adequado para o contexto do dado, uma vez que classificações erradas levam a técnica a performar de maneira não eficiente.

Finalmente, os resultados também mostram que trabalhar apenas com o esquema da base RDF é uma ideia promissora (além de pioneira, de acordo com o Capítulo 3).

6.2 Trabalhos Futuros

Como trabalho futuro, desejamos expandir o conjunto de consultas que possam ser respondidas pela nossa técnica, suportando operações como negação e agregação. Além disso, utilizarmos estratégias de ranqueamento sobre o esquema como proposta em (LEAL *et al.*, 2018) a fim de melhorar a eficiência da etapa de *keyword matching*

REFERÊNCIAS

ARNAOUT, H.; ELBASSUONI, S. Effective searching of rdf knowledge graphs. **Journal of Web Semantics**, v. 48, p. 66 – 84, 2018. ISSN 1570-8268.

AUER, S.; BIZER, C.; KOBILAROV, G.; LEHMANN, J.; CYGANIAK, R.; IVES, Z. Dbpedia: A nucleus for a web of open data. In: **The semantic web**. [*S. l.*]: Springer, 2007. p. 722–735.

BARRÓN-CEDENO, A.; ROSSO, P.; AGIRRE, E.; LABAKA, G. Plagiarism detection across distant language pairs. In: ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the 23rd COLING**. [*S. l.*], 2010. p. 37–45.

BAST, H.; BUCHHOLD, B.; HAUSSMANN, E. *et al.* Semantic search on text and knowledge bases. **Foundations and Trends® in Information Retrieval**, Now Publishers, Inc., v. 10, n. 2-3, p. 119–271, 2016.

BERNERS-LEE, T. **Linked data, 2006**. 2006.

BONDY, J. A.; MURTY, U. S. R. **GRAPH THEORY WITH APPLICATIONS**. [*S. l.*: *s. n.*], 1982.

CALVANESE, D.; COGREL, B.; KOMLA-EBRI, S.; KONTCHAKOV, R.; LANTI, D.; REZK, M.; RODRIGUEZ-MURO, M.; XIAO, G. Ontop: Answering sparql queries over relational databases. **Semantic Web**, IOS Press, v. 8, n. 3, p. 471–487, 2017.

CHANIAL, C.; DZIRI, R.; GALHARDAS, H.; LEBLAY, J.; CHANIAL, C.; DZIRI, R.; GALHARDAS, H.; LEBLAY, J.; NGUYEN, M.-h. L.; CHANIAL, C.; GALHARDAS, H. ConnectionLens : Finding Connections Across Heterogeneous Data Sources To cite this version : HAL Id : hal-01841009 ConnectionLens : Finding Connections Across Heterogeneous Data Sources. 2018.

CONSORTIUM, W. W. W. *et al.* Rdf 1.1 concepts and abstract syntax. World Wide Web Consortium, 2014.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to algorithms**. [*S. l.*]: MIT press, 2009.

DIJKSTRA, E. W. A note on two problems in connexion with graphs. **Numerische mathematik**, Springer, v. 1, n. 1, p. 269–271, 1959.

DUBEY, M.; DASGUPTA, S.; SHARMA, A.; HÖFFNER, K.; LEHMANN, J. Asknow. **Eswc**, n. c, 2016.

FENSEL, D. Ontologies. In: **Ontologies**. [*S. l.*]: Springer, 2001. p. 11–18.

GOMAA, W. H.; FAHMY, A. A. A survey of text similarity approaches. **IJCA**, Citeseer, v. 68, n. 13, p. 13–18, 2013.

HARA, C. S.; PORTO, F.; OGASAWARA, E. Tópicos em gerenciamento de dados e informações 2015. 2015.

HEATH, T.; BIZER, C. Linked data: Evolving the web into a global data space. **Synthesis lectures on the semantic web: theory and technology**, Morgan & Claypool Publishers, v. 1, n. 1, p. 1–136, 2011.

HWANG, F.; WINTER, P.; RICHARDS, D. The steiner tree problem. North-Holland, 1992.

IZQUIERDO, Y. T.; GARCÍA, G. M.; MENENDEZ, E. S.; CASANOVA, M. A.; DARTAYRE, F.; LEVY, C. H. Quiow: a keyword-based query processing tool for rdf datasets and relational databases. In: SPRINGER. **International Conference on Database and Expert Systems Applications**. [*S. l.*], 2018. p. 259–269.

JACCARD, P. Étude comparative de la distribution florale dans une portion des alpes et des jura. **Bull Soc Vaudoise Sci Nat**, v. 37, p. 547–579, 1901.

JURAFSKY, D.; MARTIN, J. **Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition**. [*S. l.*: *s. n.*], 2008. v. 2.

KHAN, W.; DAUD, A.; NASIR, J. A.; AMJAD, T. A survey on the state-of-the-art machine learning models in the context of nlp. **Kuwait journal of Science**, v. 43, n. 4, 2016.

KOMPELLA, V. P.; PASQUALE, J. C.; POLYZOS, G. C. Multicast routing for multimedia communication. **IEEE/ACM Transactions on Networking (TON)**, IEEE Press, v. 1, n. 3, p. 286–292, 1993.

LEAL, V. V. B.; MACÊDO, J. A. F. de; PERES, L.; ARAÚJO, D. Pytology: rumo ao cálculo de relevância sobre dados rdf. In: **SBBD**. [*S. l.*: *s. n.*], 2018. p. 229–234.

LOPEZ, V.; PASIN, M.; MOTTA, E. Aqualog: An ontology-portable question answering system for the semantic web. In: SPRINGER. **European Semantic Web Conference**. [*S. l.*], 2005. p. 546–562.

LOPEZ, V.; UREN, V.; SABOU, M. R.; MOTTA, E. Cross ontology query answering on the semantic web: an initial evaluation. In: ACM. **Proceedings of the fifth international conference on Knowledge capture**. [*S. l.*], 2009. p. 17–24.

MANOLA, F.; MILLER, E.; MCBRIDE, B. Rdf primer. w3c recommendation (2004). **URL http://www. w3. org/TR/rdf-primer**, 2004.

MENENDEZ, E. S.; CASANOVA, M. A.; LEME, L. A. P.; BOUGHANEM, M. Novel node importance measures to improve keyword search over rdf graphs. In: SPRINGER. **International Conference on Database and Expert Systems Applications**. [*S. l.*], 2019. p. 143–158.

NADEAU, D.; SEKINE, S. A survey of named entity recognition and classification. **Lingvisticae Investigationes**, John Benjamins, v. 30, n. 1, p. 3–26, 2007.

OUKSILI, H.; KEDAD, Z.; LOPES, S.; NUGIER, S. Pattern oriented RDF graphs exploration. **Data and Knowledge Engineering**, Elsevier B.V., v. 113, n. March 2017, p. 116–128, 2018. ISSN 0169023X. Disponível em: <https://doi.org/10.1016/j.datak.2017.06.003>.

PERES, L.; SILVA, T. L. C. da; MACEDO, J.; ARAUJO, D. Ontology-schema based query by example. In: SPRINGER. **International Conference on Conceptual Modeling**. [*S. l.*], 2019. p. 204–212.

PERES, L.; SILVA, T. L. C. da; MACEDO, J.; ARAUJO, D. Virtualized ontology query by example. In: **ER Forum and Poster Demos Session 2019**. [*S. l.*: *s. n.*], 2019. p. 148–153.

PRIM, R. C. Shortest connection networks and some generalizations. **Bell Labs Technical Journal**, Wiley Online Library, v. 36, n. 6, p. 1389–1401, 1957.

PRUD'HOMMEAUX, E.; SEABORNE, A. *et al.* Sparql query language for rdf. w3c. **Internet: https://www. w3. org/TR/rdf-sparql-query/[Accessed on April 5 th, 2017]**, 2008.

SAHA, D.; FLORATOU, A.; SANKARANARAYANAN, K.; MINHAS, U. F.; MITTAL, A. R.; ÖZCAN, F. Athena: an ontology-driven system for natural language querying over relational data stores. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 9, n. 12, p. 1209–1220, 2016.

SILVA, T. L. C. da; MAGALHÃES, R. P.; MACÊDO, J. A. de; MELO, V. de; REGO, P.; NETO, A. V. L. Improving named entity recognition using deep learning with human in the loop. In: **EDBT**. [*S. l.*: *s. n.*], 2019.

SINHA, S. B.; LU, X.; THEODORATOS, D. Personalized Keyword Search on Large RDF Graphs based on Pa ern Graph Similarity. 2018.

SOYLU, A.; KHARLAMOV, E.; ZHELEZNYAKOV, D.; JIMENEZ-RUIZ, E.; GIESE, M.; HORROCKS, I. Optiquevqs: Ontology-based visual querying. In: **VOILA@ ISWC**. [*S. l.*: *s. n.*], 2015. p. 91.

UNGER, C.; BÜHMANN, L.; LEHMANN, J.; NGOMO, A.-C. N.; GERBER, D.; CIMIANO, P. Template-based question answering over rdf data. In: ACM. **Proceedings of the 21st international conference on World Wide Web**. [*S. l.*], 2012. p. 639–648.

UNGER, C.; CIMIANO, P.; LOPEZ, V.; MOTTA, E. **QALD-1 open challenge, 2011**.

USBECK, R.; NGOMO, A.-C. N.; BÜHMANN, L.; UNGER, C. Hawk–hybrid question answering using linked data. In: SPRINGER. **European Semantic Web Conference**. [*S. l.*], 2015. p. 353–368.

WINKLER, W. E. The state of record linkage and current research problems. In: CITESEER. **Statistical Research Division, US Census Bureau**. [*S. l.*], 1999.

XU, K.; ZHANG, S.; FENG, Y.; ZHAO, D. Answering natural language questions via phrasal semantic parsing. In: **Natural Language Processing and Chinese Computing**. [*S. l.*]: Springer, 2014. p. 333–344.

YADAV, V.; BETHARD, S. A survey on recent advances in named entity recognition from deep learning models. **arXiv preprint arXiv:1910.11470**, 2019.

YAHYA, M.; BERBERICH, K.; ELBASSUONI, S.; RAMANATH, M.; TRESP, V.; WEIKUM, G. Natural language questions for the web of data. In: ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning**. [*S. l.*], 2012. p. 379–390.

YIH, S. W.-t.; CHANG, M.-W.; HE, X.; GAO, J. Semantic parsing via staged query graph generation: Question answering with knowledge base. 2015.