# Investigando Similaridade para Diferenciar Instâncias de Conceitos em Grafos de Conhecimento

**Rodrigo Oliveira Caus**[1], **Julio Cesar dos Reis**[2]

[1] Faculdade de Engenharia Elétrica e de Computação

[2]Instituto de Computação
Universidade Estadual de Campinas – SP – Brasil.

rodrigo.caus@students.ic.unicamp.br, jreis@ic.unicamp.br

***Abstract.*** *This investigation studies how to differentiate resources in a knowledge graph (KG) that stores compatibility relations between products and vehicles in automotive domain of an e-commerce. The knowledge encoded in the KG is used for automatic question answering in online stores. In this article, we present a technique to select the best candidate to answer a posed question, using a similarity measurement between the car registered in KG and the car identified in the question, obtained from features such as design and model specifier. Our solution was evaluated in distinct scenarios to explore the suitable candidate answer for the customer's questions.*

***Resumo.*** *Esta investigação estuda como diferenciar instâncias codificadas em um grafo de conhecimento (GC) que armazena relações de compatibilidade entre produtos e veículos no setor de peças automotivas em e-commerce. Nosso GC é empregado para responder automaticamente a perguntas em lojas online. Este artigo apresenta um método para seleção do melhor candidato à resposta a uma pergunta de um usuário utilizando uma métrica de similaridade entre o veículo registrado no GC e o item da pergunta do consumidor. O desafio é diferenciar dois ou mais veículos similares a partir de suas características, como "design" ou especificador do modelo. Nosso estudo foi avaliado em cenários para averiguar o candidato adequado à resposta do cliente.*

## 1. Introdução

Em plataformas de *e-commerce* é comum clientes fazerem perguntas sobre um produto quando há interesse em comprá-lo. Para as lojas *online*, há vantagens em empregar sistemas de inteligência artificial para responder a essas perguntas de maneira imediata, correta e autônoma, de um modo que melhore a experiência do consumidor ao consultar o produto e aumente a chance de conversão de venda. Em particular, há um grande volume de perguntas sobre compatibilidade entre produtos no *e-commerce*. Perguntas como 'Esse pneu funciona no palio 2014?" são frequentes no setor de peças e componentes automotivos, correspondendo de 30% a 40% das perguntas feitas. Para responder a essa pergunta precisa-se do conhecimento específico que expressa se o dado produto é compatível ou não com o carro.

A empresa *GoBots*[1] desenvolve soluções para estruturar conhecimento sobre compatibilidade entre produtos armazenando as informações em um Grafo de Conhecimento

[1]Empresa especializada em soluções de inteligência artificial para plataformas de *e-commerce* da América Latina. Este trabalho foi apoiado pela *GoBots*. Site disponível em `https://gobots.ai`

(GC) baseado em uma ontologia definida. O GC é uma base de conhecimento que descreve entidades do mundo real e suas inter-relações, organizadas na forma de grafo [Paulheim 2016]. A consulta ao GC permite formular respostas automáticas às perguntas de compatibilidade entre produtos do domínio automotivo e carros.

Em uma pergunta de compatibilidade em linguagem natural (LN) é possível identificar palavras-chave que indicam qual o item em posse do consumidor em que ele deseja utilizar o produto em questão. Para a pergunta "Este para-choque servirá para o meu corsa sedan 2013?", é possível avaliar o modelo, ano e *design* do carro em posse do usuário, sendo "corsa", "2013" e "sedan", respectivamente. No GC desenvolvido na *GoBots*, é possível diferenciar dois itens do consumidor similares a partir das suas características. Para o setor automotivo, os carros "Corsa Hatch 2013" e "Corsa Sedan 2013" podem ser considerados semelhantes: tratam-se do mesmo modelo de carro, do mesmo ano, mas diferem-se no *design*. No GC, esses carros correspondem a instâncias distintas.

Um determinado produto pode servir para um *design* e não para outro. Há diversas possibilidades de valores para o *design* de um veículo, inclusive a possibilidade de registrá-lo no GC sem um valor determinado, como "Corsa 2013" apenas. Esse problema escala quanto mais características além do *design* são previstas para registrar o item do consumidor no GC, como tamanho e tipo do motor, ano de fabricação, dentre outros. Nesse contexto, observamos a necessidade do desenvolvimento de técnicas de seleção do melhor candidato à elaboração da resposta para fornecer informações corretas e precisas aos clientes com o que está registrado no GC. Necessitamos levar em conta os atributos que diferenciam itens do consumidor similares ao consultar o GC.

Soluções para respostas automáticas para *e-commerce* tem sido investigadas na literatura. Shiqian *et al.* [Chen et al. 2019] propuseram um *framework* para geração de respostas automáticas que consideraram revisões de usuários para responder perguntas relacionadas aos produtos. O uso de grafos de conhecimento em sistemas de perguntas e resposta (P&R) é uma aplicação conhecida em tarefas de compreensão de LN. Abujabal *et al.* [Abujabal et al. 2017] introduziram QUINT, um sistema que aprende a produzir *templates* de *queries* a partir de perguntas de usuários e respectivas respostas em linguagem natural. No campo da similaridade entre relações representadas em bases de conhecimento, Fu *et al.* [Fu et al. 2012] apresentaram diferentes medidas de similaridade, com base em diferentes modelos de representações de triplas. As medidas são utilizadas para restringir o escopo de *queries* em uma ferramenta de busca semântica sobre o conjunto de dados da DBpedia [Auer et al. 2007]. Nossa análise de literatura indicou que análise de similaridade em GCs tem amplo espaço para serem explorados considerando a aplicação em sistemas de P&R, e ainda mais no contexto do *e-commerce*.

Neste trabalho apresentamos um modelo para obtenção de uma métrica que fornece um grau de similaridade entre os itens do consumidor que estão registrados no GC e o item em que se está sendo perguntado. A partir desse modelo, apresentamos um método que implementa consultas ao GC na linguagem SPARQL para o cálculo da métrica proposta e para a seleção de um candidato à resposta ao cliente. O método desenvolvido foi avaliado em cenários do grafo voltado ao setor de peças automotivas. Em particular, conduzimos análises no emprego dele na *GoBots*. Contribuímos com uma medida de similaridade entre instâncias em um GC, aplicada a um domínio específico, utilizando uma ontologia de modelo para representar, particularmente, relações de compatibilidade.

## 2. Fundamentos

Sant'Anna *et al.* [Sant'Anna et al. 2020] apresentaram a ontologia usada neste trabalho para definir conceitos e relações registrados no GC, criada com base no domínio de *e-commerce*. A ontologia especifica informações de compatibilidade entre produtos. A Figura 1 apresenta uma visão geral das principais classes e relações. O trabalho de Sant'Anna *et al.* investigou como estruturar relações de compatibilidade a partir de perguntas sobre produtos do *e-commerce* já respondidas por atendentes humanos. A investigação abordou ainda como o GC produzido é consultado por um serviço desenvolvido para formular respostas à consumidores para questões envolvendo compatibilidade. Esse GC, em produção atualmente na *GoBots*, tem permitido formular respostas a mais de 20 mil perguntas de compatibilidade para 28 lojas de um dos maiores *marketplaces* da América Latina.

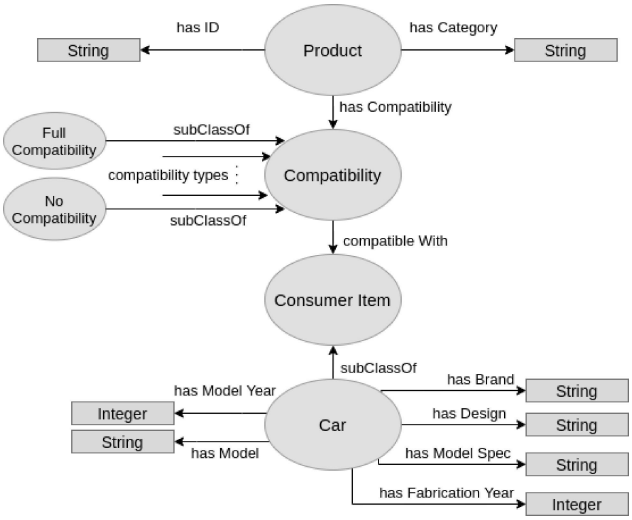


**Figura 1. Ontologia definida por Sant'Anna *et al.* [Sant'Anna et al. 2020] para representar compatibilidade entre produtos.**

A classe *Product* representa um produto numa plataforma *e-commerce*. O *ID* é um identificador do produto e *Category* é a categoria em que o produto se insere no domínio do *e-commerce*, como "guarda-roupas" no domínio de móveis ou "faróis automotivos" no domínio de componentes automotivos.

*ConsumerItem* é uma classe abstrata que tem o propósito de identificar itens em posse do consumidor que possam ter alguma integração com produtos do *e-commerce*. Uma subclasse de *ConsumerItem* deve ter atributos que adequadamente identifiquem o item. Esses atributos são organizados em duas categorias:

- **Atributos Essenciais**: Conjunto de atributos que unicamente identificam um item. Um item tem todos os atributos definidos neste conjunto.
- **Atributos Acessórios**: Conjunto de atributos opcionais que permitem diferenciar dois itens com o mesmo conjunto de atributos essenciais. Um item poderá ter qualquer subconjunto deste, inclusive o nulo, isto é, apenas os atributos essenciais em sua instância.

Uma instância do item do consumidor é única para cada conjunto de valores dos atributos que ele possui. Como um exemplo, apresentamos uma subclasse de *ConsumerItem* nomeada *Car*. Os atributos essenciais de um carro que permitem identificá-lo

unicamente são o *modelo* e o *ano*. Os atributos acessórios escolhidos foram *marca* (isto é, o nome da fabricante do carro), *design* (como *sedan* ou *hatch*), *especificação do modelo* (nome atribuído pela fabricante, como "Fire" e "ELX" atribuídos ao "Palio" pela marca *Fiat*) e ano de fabricação. Este ano pode diferir do ano do modelo, como previsto pelo Certificado de Registro e Licenciamento do Veículo no Brasil.

O conceito de compatibilidade entre produtos é armazenado através de relações envolvendo a classe *Compatibility*. A ontologia diferencia as compatibilidades propriamente ditas através do tipo de compatibilidade. O tipo indica se a compatibilidade é afirmativa ou negativa; por exemplo, que são respectivamente representadas na ontologia como *FullCompatibility* e *NoCompatibility*. Mais especificamente, esses tipos são subclasses disjuntas da classe *Compatibility*, que é usada como uma classe abstrata. Então quaisquer objetos de *Compatibility* devem ser objetos de algum dos tipos de compatibilidade. Outros tipos de compatibilidade, como *ConditionalCompatibility* e *UniversalCompatibility* são previstos na ontologia apresentada por Sant'Anna *et al.*, porém apenas os tipos mais objetivos são apresentados na Figura 1.

## 3. Computando Similaridade entre Instâncias em Grafos de Conhecimento

A Figura 2 apresenta uma visão geral do serviço projetado para consultar triplas de um *RDF store* para responder às perguntas de compatibilidade. O serviço da *GoBots* é responsável por gerenciar as respostas automáticas para lojas de *e-commerce*. Ao receber uma pergunta do usuário (1 na Figura 2), o serviço faz o processamento do texto em linguagem natural, mapeando a intenção da pergunta e as entidades dela (2 na Figura 2). Uma intenção representa o propósito da sentença de entrada, sendo comumente substantivos ou verbos que descrevem o que se entende da sentença. Uma entidade representa um termo ou uma expressão com um significado relevante para a compreensão da sentença, sendo muitas vezes uma palavra-chave ou um padrão a ser mapeado na frase.

Quando uma pergunta de compatibilidade é identificada a partir da intenção, as entidades, junto com os identificadores do produto, são enviados à API responsável por consultar o *RDF store* em produção, chamada de *Knowledge Graph Service* (3 na figura 2). Nesta API, as entidades e os IDs do produto são utilizados para formular uma *query* de consulta ao GC (a); e os resultados retornados passam por um processo de seleção para elaborar uma resposta final ao usuário (b). Caso a seleção encontre um melhor candidato, a API retorna a resposta final ao usuário (4 na Figura 2); caso contrário, o sistema se abstém de responder a pergunta de compatibilidade em específico, permitindo que um atendente humano da loja virtual o faça posteriormente.

A recuperação do tipo de compatibilidade entre o produto e o item do consumidor é feita através de *queries* SPARQL. Para a formulação de uma *query*, o conjunto de entidades extraídas da pergunta são mapeados para os predicados definidos na ontologia da Figura 1 para a classe *Car*. O mapeamento das entidades ocorre da seguinte forma: A entidade *modelo* para *hasModel*; *spec* para *hasModelSpec*; *design* para *hasDesign*; e *ano* para *hasModelYear*. Em particular, na presença de uma pergunta contendo dois anos consecutivos, como "2020/2021", os números são separados nos predicados *hasFabricationYear* e *hasModelYear*, sendo o ano de fabricação o menor.

Os valores extraídos da sentença em linguagem natural são colocados na *query* como objetos da tripla de consulta. Considere a entidade "modelo" e o respectivo valor

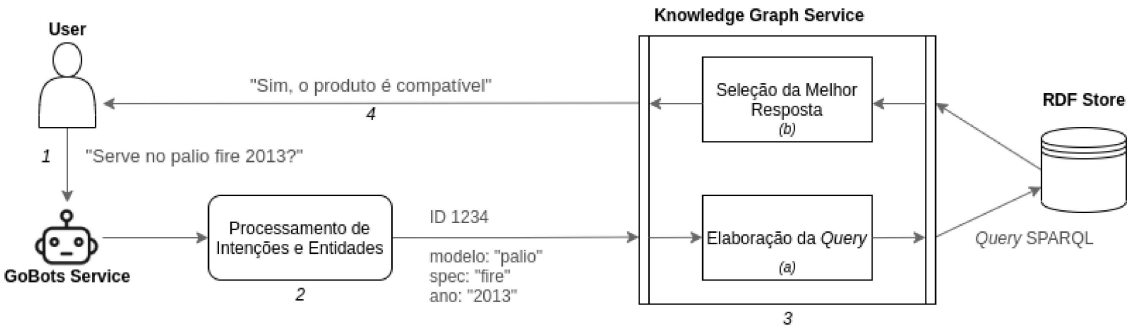


**Figura 2. Solução para responder perguntas de compatibilidade usando GC.**

"fiesta". A tripla de consulta por uma instância de carro que contém o modelo indicado pela entidade é:

```
?car _:hasModel "fiesta" .
```

A consulta pelo produto é feita com o predicado *hasID*, definido na ontologia, e o objeto com o valor do ID recebido pela API:

```
?product _:hasID "1234" .
```

As triplas de consulta com predicados mapeados pelas entidades, em conjunto com a tripla de consulta pelo ID do produto são combinadas a outras triplas de consulta definidas pela ontologia, de modo a obter o tipo da compatibilidade que relaciona o produto e o item do consumidor. A forma mais direta de se elaborar a consulta, é inserir na *query* todos os mapeamentos das entidades recebidas pela API. Esta abordagem mais direta possui dois problemas em potencial:

1. Mais de um item do consumidor pode conter os atributos mapeados na *query*, diferenciando-se por outros atributos acessórios. Como apenas um item deve ser selecionado para formular a resposta, vê-se a necessidade de estabelecer critérios que permitam ordenar os resultados retornados da *query*;
2. Uma *query* com muitas entidades mapeadas desta forma pode restringir os resultados retornados, pois a possibilidade de uma instância de item do consumidor conter todos os atributos da pergunta é baixa, e possivelmente uma outra instância com menos atributos pode servir para formular a resposta ao usuário.

Nossa investigação trata exatamente estes problemas. Elaboramos um método, apresentado na subseção 3.2 que implementa a construção de *queries* em SPARQL (3-a na Figura 2) e trata os resultados retornados (3-b na Figura 2) para selecionar a melhor resposta ao usuário com base nas instâncias de carros e nós de compatibilidades presentes no GC. O método se baseia em uma métrica de distância obtida a partir dos atributos acessórios das instâncias de itens do consumidor registradas no GC, bem como dos atributos acessórios mapeados a partir das entidades extraídas da pergunta do usuário. A subseção 3.1 apresenta definições matemáticas que fundamentam o método proposto.

### 3.1. Cálculo da Distância entre Dois Itens do Consumidor

Uma distância é uma aplicação entre dois vetores de um espaço que resulta em um número real satisfazendo propriedades de simetria, positividade e desigualdade triangular

[Pulino 2012]. A distância entre duas instâncias similares de um item do consumidor é obtida com base em uma representação vetorial dos valores de atributos que cada instância guarda. Essa representação é apenas um suporte matemático e não corresponde ao modo com que a instância é codificada no GC. Este cálculo de distância só é considerado entre dois itens do consumidor da mesma classe que contenham os mesmos valores de atributos essenciais. No caso do domínio automotivo, por exemplo, ambos os carros devem ter o mesmo modelo e o mesmo ano para que o valor da métrica faça sentido.

Para cada atributo acessório do item do consumidor, consideramos a codificação em *one-hot vectors*. A representação vetorial do *k-ésimo* atributo acessório é dada por:

$$V_k \in \mathbb{B}^{n_k}$$

em que $\mathbb{B}$ representa o conjunto binário $\{0, 1\}$ e $n_k$ representa o número de palavras distintas de possíveis valores para o *k-ésimo* atributo acessório. Nesta representação, $V_k$ consiste de um vetor de zeros em todas as dimensões, exceto de um único "um", cuja posição indica uma palavra do vocabulário do atributo em questão. Considere, por exemplo, o atributo *design* para uma instância de carro. Suponha que o vocabulário deste atributo seja composto apenas de três valores possíveis: *hatch*, *sedan* e *coupé*. Para este caso, temos $n_1 = 3$, em que cada palavra pode ser codificada da forma:

$$\text{“}\textit{hatch}\text{”} \mapsto (1, 0, 0) \qquad \text{“}\textit{sedan}\text{”} \mapsto (0, 1, 0) \qquad \text{“}\textit{coupé}\text{”} \mapsto (0, 0, 1)$$

É possível estender a representação de *one-hot vectors* para cobrir a existência de um vetor nulo **0**, tendo em vista que os atributos acessórios são opcionais. Por exemplo, um carro que não tenha em sua instância o *design* especificado teria como vetor $V_1 = (0, 0, 0)$ para o espaço do exemplo.

A utilização de uma **norma** é comum como base da distância em espaços vetoriais. Adotamos a **norma da soma** [Shimakawa et al. 2003] para o atributo acessório $V_k$ como a soma do valor absoluto das entradas do vetor. Portanto, a distância entre dois valores, $V_k$ e $U_k$, para o *k-ésimo* atributo acessório é dada pela norma da diferença entre as representações vetoriais:

$$d(V_k, U_k) = \|V_k - U_k\| = \sum_{j=1}^{n_k} |v_{jk} - u_{jk}|, \qquad V_k, U_k \in \mathbb{B}^{n_k} \tag{1}$$

A distância entre duas instâncias de um item do consumidor é obtida diretamente das distâncias entre os atributos acessórios. Sejam $C$ e $D$ duas instâncias similares de uma mesma classe de item do consumidor, com $C$ definida por $N$ atributos acessórios $V_1, ..., V_N$ e $D$ definida por $N$ atributos acessórios $U_1, ..., U_N$:

$$d(C, D) = \sum_{k=1}^{N} d(V_k, U_k), \qquad V_k, U_k \in \mathbb{B}^{n_k} \tag{2}$$

As escolhas de codificação do atributo por vetores *one hot* e da norma da soma permitem simplificação do cálculo da distância entre dois valores de um atributo a três casos:

1. A distância de dois atributos de valores iguais é zero. Esta é uma condição necessária e suficiente. Exemplo: $d(\text{“}hatch\text{”}, \text{“}hatch\text{”}) = 0$
2. A distância entre um atributo com valor definido e um atributo sem valor definido é unitária. Exemplo: $d(\text{“}hatch\text{”}, \mathbf{0}) = 1$
3. A distância de dois atributos de valores definidos e distintos é igual a 2. Exemplo: $d(\text{“}hatch\text{”}, \text{“}sedan\text{”}) = 2$

Essa simplificação permite abstrair o vocabulário que compõe os valores de cada tipo de atributo acessório previsto, possibilitando o cálculo de distância de duas instâncias de itens do consumidor em grafos extensos, sem necessariamente se conhecer previamente todas as ocorrências de valores de atributos existentes no grafo.

Considere os atributos acessórios *design*, *especificação do modelo* e ano de fabricação. Considere as instâncias de carro no exemplo da Figura 3, ambas versões diferentes do Palio 2013. Com base nos três casos indicados, as distâncias para cada um dos atributos acessórios é: *design*, $d(\mathbf{0}, \mathbf{0}) = 0$; *especificação do modelo*, $d(\text{“}fire\text{”}, \text{“}elx\text{”}) = 2$; e ano de fabricação, $d(\text{“2012”}, \mathbf{0}) = 1$. Ao fim, a distância entre as duas instâncias é igual a 3.

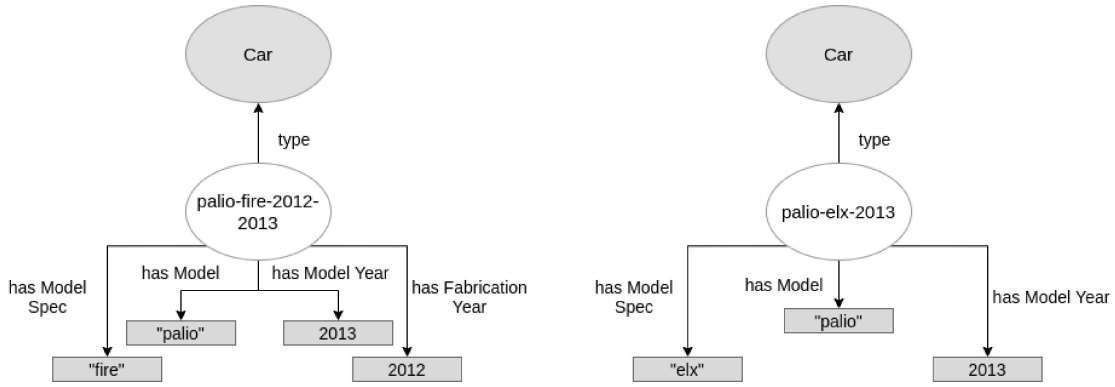


**Figura 3. Exemplos de instâncias de carros que compartilham dos mesmos valores de atributos essenciais: “Palio” e “2013”.**

### 3.2. *Queries* SPARQL Recuperando Resultados Ordenados pela Distância

Esta solução consiste em elaborar uma *query* SPARQL que calcule a distância entre o item do consumidor perguntado e os itens instanciados no GC com base em seus atributos acessórios, e retorne os resultados já ordenados com base nesta métrica. O Código 1 apresenta um *template* de *query* SPARQL utilizado para uma pergunta de compatibilidade do domínio automotivo. Este *template* é preenchido da seguinte forma:

- O identificador do produto recebido pela API, é inserido como objeto da tripla de consulta pelo produto, sucedendo o predicado *hasID*;
- Os atributos essenciais, mapeados a partir das entidades da pergunta do consumidor, são inseridos diretamente como objetos sucedendo os predicados que definem estes atributos. No domínio automotivo, *hasModel* e *hasModelYear* são os predicados para o “modelo” e “ano” extraídos da pergunta, respectivamente;

- Os atributos acessórios, mapeados das entidades recebidas pela API, são utilizados para construir a *distance_sub_query*, uma *query* mais interna, vinculando triplas de consulta pelos atributos a um valor de distância.

A proposta é que cada domínio tenha um *template* semelhante, bastando substituir os predicados e objetos relacionados aos atributos que fazem consulta ao item do consumidor voltado ao domínio em específico.

```
1  prefix onto:<http://graph.edu/ontology#>
2  select ?compatibility_type ?car ?distance
3  where {
4      ?product rdf:type onto:Product;
5               onto:hasID "PRODUCT_ID";
6               onto:hasCompatibility ?compatibility.
7      ?compatibility onto:compatibleWith ?car;
8                      rdf:type ?compatibility_type.
9      ?car rdf:type onto:Car;
10          onto:hasModel "CAR_MODEL" ;
11          onto:hasModelYear CAR_YEAR .
12     bind({{ distance_sub_query }}
13     as ?distance) }
14 order by ?distance
```

**Código 1. Query levando em conta modelo e ano da pergunta, retornando os resultados ordenados por distância. *distance_sub_query* é um template para o conjunto de expressões com objetivo de calcular a distância.**

A construção da *distance_sub_query* depende de três funções SPARQL:

- **exists**: recebe um conjunto de padrões de sentenças, retornando *verdadeiro* quando estão contidas no grafo e *falso* caso contrário.
- **filter**: recebe uma expressão lógica para ser validada, eliminando triplas cujo resultado da expressão retorna *falso*.
- **fn:not**: faz a negação lógica de uma expressão.

Essas funções são combinadas e os valores das expressões lógicas são convertidos para números inteiros, sendo *verdadeiro* equivalente a 1 e *falso* a zero. Este procedimento foi realizado para a versão 1.1 da linguagem SPARQL sobre o Virtuoso *Open Source* 7.20, o servidor de banco de dados para o GC construído.

O Algoritmo 1 apresenta o procedimento que constrói a *sub-query* a partir dos atributos acessórios mapeados das entidades da pergunta. A distância inicialmente é mapeada com zero e para cada atributo acessório previsto para o carro, são avaliados dois casos:

1. Caso a pergunta contenha a entidade para o atributo acessório, soma-se 1 para o carro do grafo que não contenha o atributo (linha 7), e 2 para o carro que contenha o atributo e tenha um valor diferente do encontrado na pergunta (linhas 7 e 8).
2. Caso a pergunta não contenha a entidade para o atributo acessório, soma-se 1 para o carro que contenha qualquer valor para este atributo (linha 12).

A *query* do Código 1 retorna uma lista de URIs referentes aos tipos de compatibilidade entre o produto e cada carro registrado no GC, ou uma lista vazia, caso nenhuma

**Algoritmo 1:** Construção da *sub-query* para cálculo de distância. O símbolo "$" indica o valor da variável que deve ser inserido na *string*.

**input :** $entities$
**output:** $query\_string$

```
1  query_string ← emptyString()
2  foreach attribute ∈ accessoryAttributes(Car ) do
3  |  value ← entities.get(attribute)
4  |  predicate ← attribute.predicate
5  |  if value ≠ 0 then
6  |  |  query_string.addLines(
7  |  |  + fn:not (exists {?car onto : $predicate "$value"})
8  |  |  + exists {?car onto : $predicate ?acs
   |  |    filter (?acs != "$value") }
9  |  |  )
10 |  else
11 |  |  query_string.addLines(
12 |  |  + exists {?car onto : $predicate ?acs}
13 |  |  )
14 |  end
15 end
16 return query_string
```

relação de compatibilidade entre o produto e os carros com modelo e ano definidos na *query* esteja registrada no GC. Quando encontrado, o primeiro resultado da lista é retornado para elaboração da resposta final ao usuário.

## 4. Avaliação

Considere o GC de exemplo da Figura 4. O *Fiesta Sedan 2012* é compatível com o produto de $ID$ 123, enquanto o *Fiesta Hatch 2012* é incompatível com o mesmo produto. O *Fiesta 2012*, possui uma compatibilidade condicional com o produto, isto é, existe alguma possível restrição na descrição do anúncio que permite a compra para este modelo em específico, como medidas do produto, peso ou alteração complementar para realizar instalação. Apresentamos cenários de consulta com base nesse GC para avaliar as *queries* construídas por nossa solução, as distâncias calculadas e respostas retornadas ao usuário.

### 4.1. Consulta a uma Instância de Item do Consumidor Existente no GC

Este cenário apresenta o processo de consulta ao GC em um caso em que há uma correspondência exata entre o veículo descrito na pergunta do usuário e a instância do carro na base de conhecimento. Considere a pergunta: "Posso comprar o produto pro fiesta sedan 2012?". As entidades extraídas da sentença são: **modelo** "fiesta"; **design** "sedan"; **ano** "2012". Nesta pergunta, o usuário não informa ano de fabricação, ou um especificador do modelo do veículo. O Código 2 apresenta a *query* SPARQL resultante da aplicação do método apresentado na Seção 3.2, pelo Algoritmo 1.

```
1 prefix onto:<http://graph.edu/ontology#>
```

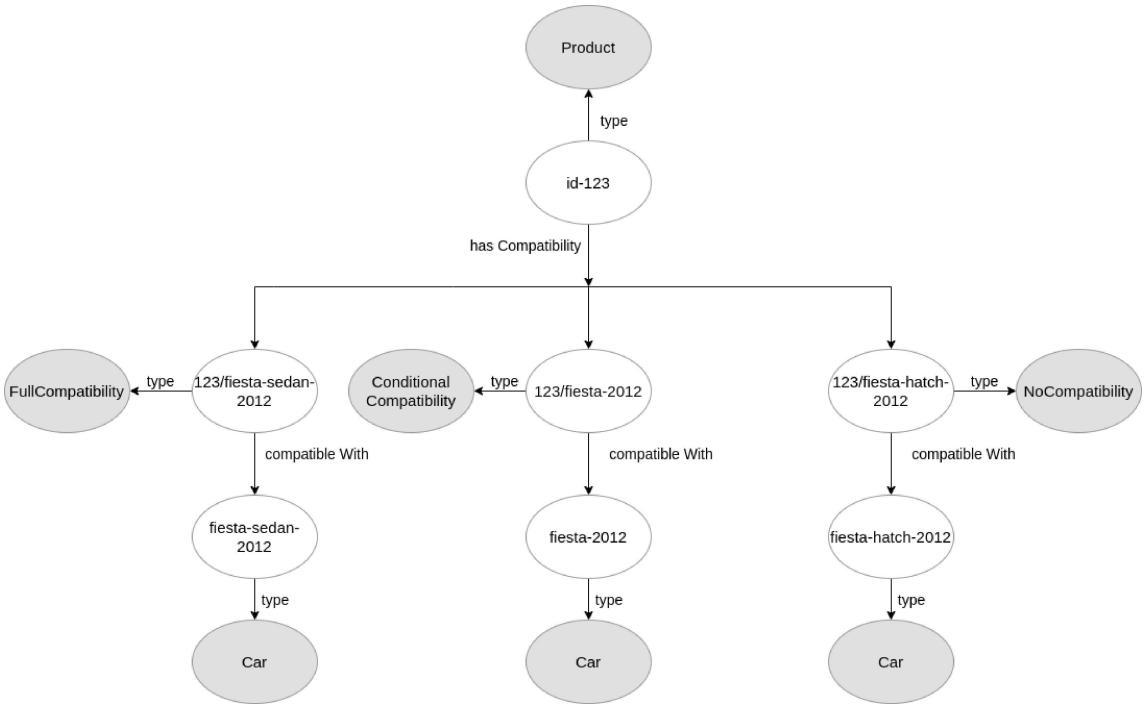


**Figura 4. Grafo de exemplo com carros similares com informações de compatibilidade sobre um mesmo produto. Os carros diferenciam-se pelo *design*. Os atributos foram omitidos para simplificar o conteúdo do diagrama.**

```
select ?compatibility_type ?car ?distance
where {
  ?product rdf:type onto:Product;
           onto:hasID "123";
           onto:hasCompatibility ?compatibility.
  ?compatibility onto:compatibleWith ?car;
                 rdf:type ?compatibility_type.
  ?car rdf:type onto:Car;
       onto:hasModel "fiesta" ;
       onto:hasModelYear 2012 .
  bind(0
   + fn:not(exists{?car onto:hasModelDesign "sedan" })
   + exists{?car onto:hasModelDesign ?acs filter(?acs!="sedan")}
   + exists{?car onto:hasModelSpec ?acs}
   + exists{?car onto:hasFabricationYear ?acs}
  as ?distance) }
order by ?distance
```

**Código 2. *Query* levando em conta modelo, ano e design da pergunta "Posso comprar o produto pro fiesta sedan 2012?", retornando os resultados ordenados por distância.**

A Tabela 1 apresenta os resultados da *query*. Essa tabela apresenta o tipo de compatibilidade, URI do carro e o valor calculado de distância; e os valores dos atributos acessórios considerados para a elaboração da *query*. A compatibilidade que melhor responde à pergunta do usuário neste cenário é *FullCompatibility*, em que ocorre uma

correspondência exata dada pela distância zero. Esse tipo de compatibilidade permite elaborar uma resposta final como: "Este produto é compatível. Aproveite!".

**Tabela 1. Resultados da *query* do Código 2 para o grafo da Figura 4.**

| compatibility_type | car | design | spec | year | distance |
|---|---|---|---|---|---|
| FullCompatibility | fiesta-sedan-2012 | sedan | - | - | **0** |
| ConditionalCompatibility | fiesta-2012 | - | - | - | **1** |
| NoCompatibility | fiesta-hatch-2012 | hatch | - | - | **2** |

### 4.2. Consulta a uma Instância de Item do Consumidor Inexistente no GC

Este segundo cenário ilustra o processo de consulta ao GC em um caso em que não existe a instância de carro referenciada na pergunta do usuário. Logo, uma outra instância deve ser selecionada como alternativa dentre as similares. Nesse caso, considere a seguinte pergunta exemplo: "Este produto dá no Fiesta Rocam 2012?". Nesta pergunta são detectadas as entidades: **modelo** "fiesta"; **spec** "rocam"; e **ano** "2012". O atributo "Rocam" representa uma especificação do modelo "Fiesta", e é um acessório ausente em todas as instâncias apresentadas na Figura 4. O código 3 apresenta a parte da *query* SPARQL que difere da apresentada no código 2.

```
bind(0
  + exists{?car onto:hasModelDesign ?acs}
  + fn:not(exists{?car onto:hasModelSpec "rocam" })
  + exists{?car onto:hasModelSpec ?acs filter(?acs!="rocam")}
  + exists{?car onto:hasFabricationYear ?acs}
as ?distance)
```

**Código 3. *Sub-query* para cálculo do valor de distância levando em conta *spec* da pergunta "Este produto dá no Fiesta Rocam 2012?".**

A Tabela 2 apresenta os resultados da consulta. Para responder a pergunta deste cenário, é utilizada a compatibilidade *ConditionalCompatibility*, associada ao "Fiesta 2012" instanciado no grafo (resultado com menor distância na Tabela 2). Entende-se que uma instância mais genérica fornecerá uma resposta mais genérica ao consumidor, mas que ainda possa esclarecer sua dúvida, permitindo um maior volume de perguntas respondidas com informação recuperada pelo GC. Ao final, a resposta "O produto é compatível, mas verifique as possíveis restrições e medidas na descrição do anúncio antes da compra!" pode ser fornecida ao usuário como representação em LN da compatibilidade *ConditionalCompatibility*.

**Tabela 2. Resultados da *query* do Código 3 para o grafo da Figura 4.**

| compatibility_type | car | design | spec | year | distance |
|---|---|---|---|---|---|
| ConditionalCompatibility | fiesta-2012 | - | - | - | **1** |
| FullCompatibility | fiesta-sedan-2012 | sedan | - | - | **2** |
| NoCompatibility | fiesta-hatch-2012 | hatch | - | - | **2** |

## 5. Discussão

Os cenários apresentados demonstraram como a solução proposta resolve os problemas de consulta em um grafo com instâncias de itens do consumidor que compartilham um conjunto de atributos. Diante do grafo com três instâncias similares (cf. Figura 4), o algoritmo proposto permite encontrar no grafo uma correspondência exata com o item perguntado, desde que a instância esteja representada no grafo e relacionada por uma compatibilidade com o produto em questão. Quando a correspondência não é exata, nosso procedimento determina uma medida de proximidade com o que é perguntado, com base no número e qualidade de atributos, o que permite selecionar a instância de carro mais genérica capaz de fornecer uma resposta correta e imediata à questão do usuário.

Por outro lado, a medida de distância pode ser uma medida de dispersão. Quando são mapeados na pergunta do usuário atributos acessórios que não são registrados em nenhuma das instâncias similares do GC, os resultados retornados podem conter distâncias arbitrariamente altas. Quanto maior o número de atributos definidos e quanto mais diferentes forem entre si, maiores as distâncias obtidas, dando menor relevância à ordenação de candidatos. É necessário ter um GC com alto número de relações entre produtos e carros registradas no grafo para fornecer um alto número de respostas a consumidores, como também fornecer respostas com maior qualidade e corretude. No entanto, existe certa dificuldade em registrar todas as combinações possíveis de atributos acessórios dentre instâncias similares. Nesses cenários, pode-se determinar um limiar para o valor de distância, permitindo que o sistema de GC se abstenha de responder ao usuário final para valores maiores que o determinado. O valor adequado para tal limiar pode ser investigado em um trabalho futuro.

O método desenvolvido é igualmente aplicado para outras subclasses de *ConsumerItem*, desde que na ontologia sejam registrados os atributos essenciais, necessários para a representação do item no GC, e os atributos acessórios possíveis para o item. Outros domínios além do automotivo são de grande interesse para registrar informações de compatibilidade, bem como os de componentes para computadores e de acessórios de telefonia. É de grande interesse da GoBots expandir o conhecimento do grafo para permitir responder mais perguntas de diferentes domínio e atender a mais clientes interessados em aprimorar a experiência dos consumidores em compras *online*.

## 6. Conclusão

O uso de GCs em sistemas de respostas automáticas em plataformas de *e-commerce* beneficiam tanto a experiência do usuário consumidor da plataforma quanto a conversão de vendas do lojista. Este trabalho propôs uma solução visando um alto grau de assertividade ao efetuar uma resposta ao cliente e baixo comprometimento da taxa de resposta do sistema que gerencia o GC. Definimos métricas que relacionam elementos da ontologia (itens do consumidor) com o que se é perguntado pelo usuário. Os procedimentos propostos foram avaliados em diferentes cenários. Implementamos e empregamos esta proposta junto ao sistema de respostas automáticas da empresa *GoBots* para responder às perguntas de clientes sem a ajuda direta de um assistente humano. Como trabalhos futuros, consideramos a possibilidade de extensão e aplicação da solução em outros domínios além do automotivo.

## Referências


Abujabal, A., Yahya, M., Riedewald, M., and Weikum, G. (2017). Automated template generation for question answering over knowledge graphs. In *Proceedings of the 26th international conference on world wide web*, pages 1191–1200.

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2007). Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer.

Chen, S., Li, C., Ji, F., Zhou, W., and Chen, H. (2019). Driven answer generation for product-related questions in e-commerce. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 411–419.

Fu, L., Wang, H., and Yu, Y. (2012). Towards better understanding and utilizing relations in dbpedia. *Web Intelligence and Agent Systems*, 10:291–303.

Paulheim, H. (2016). Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web*, 8:489–508.

Pulino, P. (2012). *Álgebra Linear e suas Aplicações*. Instituto de Matemática, Estatística e Computação Científica, Unicamp.

Sant'Anna, D. T., Caus, R. O., dos Santos Ramos, L., Hochgreb, V., and dos Reis, J. C. (2020). Generating knowledge graphs from unstructured texts: Experiences in the e-commerce field for question answering. In *Advances in Semantics and Linked Data: Joint Workshop Proceedings from ISWC 2020*, pages 56–71.

Shimakawa, K., Plato, R., Borne, R., and Borne, S. (2003). *Concise Numerical Mathematics*. Graduate studies in mathematics. American Mathematical Society.