Módulos de alinhamento para o AgreementMakerLight

Ricardo Ferreira Guimarães

Trabalho de formatura apresentado
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Bacharel em Ciência da Computação

Orientador: Prof. Dra. Renata Wassermann

São Paulo, dezembro de 2014

Módulos de alinhamento para o AgreementMakerLight

Agradecimentos

Sinceros agradecimentos à todos aqueles que tornaram este trabalho possível de forma direta ou indireta. Em especial, à minha família, amigos, colegas e professores.

Resumo

GUIMARÃES, R. F. **Módulos de alinhamento para o AgreementMakerLight**. Trabalho de formatura (Bacharelado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2014.

Este trabalho tem por objetivo estudar o alinhador de ontologias *AgreementMakerLight*, para entender cada etapa existente na resolução do problema de emparelhar ontologias e formular novos implementos que possam melhorar o desempenho do programa. Para isto, faz-se um estudo teórico dos atuais algoritmos e métodos empregados, analisa-se as funcionalidades já implementadas, e finalmente, a partir de informações obtidas do estudo teórico, adicionam-se algumas funções ao *AgreementMakerLight*, que melhorem seu desempenho em alguns dos casos citados na bibliografia. Ao final, observa-se que algumas camadas tem mais espaço para desenvolvimento que outras e discutem-se algumas possíveis melhorias no *AgreementMakerLight* e nos módulos desenvolvidos.

Palavras-chave: alinhamento de ontologias, heterogeneidade semântica, ontologias, web semântica.

Sumário

Capítulo 1

Introdução

1.1 Contextualização

As ontologias consistem em documentos estruturados escritos em linguagens capazes de representar lógicas descritivas como OWL e RDF (linguagens ontológicas), utilizadas para definir um domínio de conhecimento formalmente, podendo empregar definições de classes, domínios, propriedades, relações e indivíduos. A partir delas, por meio de algum mecanismo de inferência, extraem-se novas informações com base apenas na descrição estática do domínio e respeitando a semântica da lógica definida. Entre os casos de uso mais notáveis estão a documentação de dados genéticos e descrição de serviços web [4, 5].

Mais especificamente adota-se, neste documento, a definição formal dada em [4]:

Ontologia é uma tupla $o = (C, I, R, T, V, \sqsubseteq, \perp, \in, =)$, tal que:

- C é o conjunto de classes;
- I é o conjunto de indivíduos;
- R é o conjunto de relações;
- T é o conjunto de tipos de dados;
- V é o conjunto de valores (C, I, R, T, V sendo disjuntos dois a dois);
- $\sqsubseteq$ é uma relação em $(C \times C) \cup (R \times R) \cup (T \times T)$ denominada especialização;
- $\perp$ é uma relação em $(C \times C) \cup (R \times R) \cup (T \times T)$ denominada disjunção;
- $\in$ é uma relação sobre $(I \times C) \cup (V \times T)$ denominada instanciação;
- $=$ é uma relação sobre $I \times R \times (I \cup V)$ denominada atribuição.

Em muitas aplicações, existe a necessidade de se trabalhar com mais de uma ontologia, como sistemas multiagentes, integração de dados, composição de serviços web, sistemas *peer-to-peer*, e a própria realização da Web Semântica (nesta última, tendo em vista a manipulação automática de informações) [1, 4, 13]. Nestes casos, fica evidente o problema da heterogeneidade das representações, o qual decorre das diferentes variações possíveis ao se especificar um mesmo conjunto de conhecimentos: seja na escolha dos nomes das entidades, diferenças de abordagem do domínio (foco, abrangência, granularidade), linguagem da ontologia (OWL, RDF, SKOS), entre outros [3, 4].

Por exemplo, em duas ontologias que descrevem acervos bibliográficos, uma pode escolher o termo *Paper* e outra o termo *Article* para a mesma entidade, ou ainda, os termos podem estar em linguagens naturais distintas. Também pode ocorrer que uma delas represente o autor e a instituição como entidades, e a outra não.

Neste trabalho, define-se mapeamento ou correspondência como uma relação entre duas entidades, a partir de uma variação da definição dada por Euzenat e Shvaiko [10] (explicita-se, neste

documento, a especialização), representando um alinhamento entre duas ontologias O e O' como uma quíntupla $(id, e, e', n, \Re)$, tal que:

- id é um identificador do mapeamento;
- e é uma entidade de O e e' de O';
- n é uma medida de confiança, que possibilite comparação entre dois alinhamentos quanto a este aspecto (para escolher o mais confiável, por exemplo);
- $\Re$ uma relação dentre: equivalência ($\equiv$), generalização ($\geq$), especialização ($\leq$), disjunção($\perp$) ou sobreposição($\cap$)

Denomina-se um conjunto de tais mapeamentos, obtidos a partir de ontologias definidas, um alinhamento. Por fim, o programa ou algoritmo responsável pela obtenção deste conjunto para ontologias de entrada determinado, chama-se alinhador de ontologias. Por exemplo, na figura 1.1, temos dois trechos de ontologias, sendo que uma delas contém a classe `Veículo` e a outra `Item` e tais que as setas cheias que ligam as duas árvores representam mapeamentos entre as entidades.

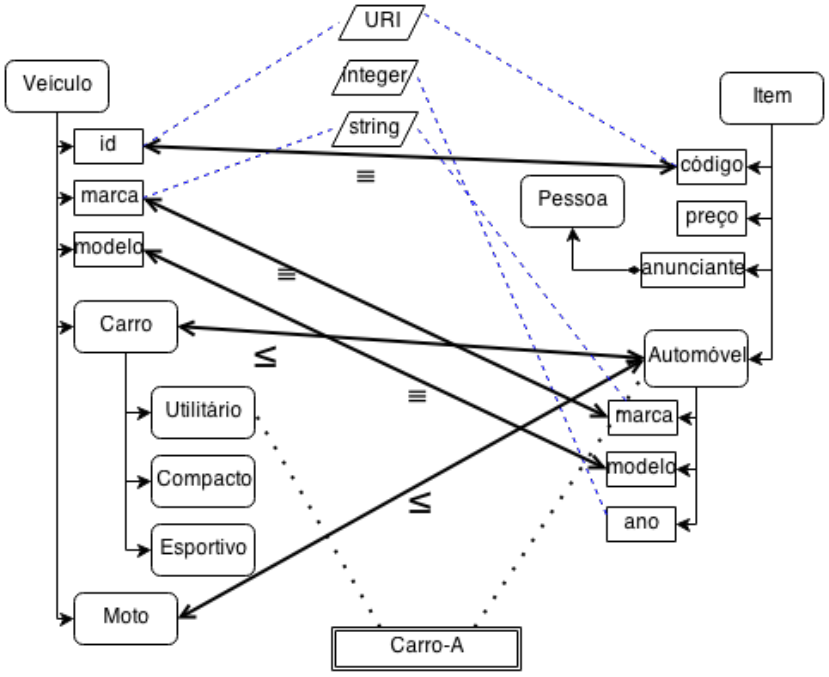

Figura 1.1: *Representação gráfica de um trecho de duas ontologias e com alguns mapeamentos*

Cada ontologia define um conjunto de axiomas, dos quais derivam-se modelos que os satisfazem. Desta forma, o processo de alinhamento consiste em encontrar um modelo que satisfaça ambas ontologias, mapeando as entidades de uma ontologia na outra (em particular, utilizando as relações em $\Re$).

Cabe ao alinhador lidar com diversos tipos de heterogeneidades entre as ontologias, tais como: diferença na granularidade da representação, diferentes perspectivas de um mesmo aspecto, escolha dos termos, padrão e linguagem natural de escrita dos nomes do elementos, formatos distintos, sinonímia, entre outros. Para muitos destes problemas, não há uma solução única, ou que seja geral o suficiente. Por conta disto, existem diferentes alinhadores de ontologia, diferindo não somente nas técnicas, algoritmos e recursos empregados, como também nos requisitos externos aos quais se adequam (tempo de execução, área de conhecimento, recursos de terceiros, interação com o usuário, entre outros).

O AML[1] (*AgreementMakerLight*) é um dos atuais alinhadores de ontologia que tem produzido

[1] http://somer.fc.ul.pt/aml.php

resultados promissores, segundo avaliações recentes da OAEI[2] (*Ontology Alignment Evaluation Initiative*), uma organização responsável por concentrar informações sobre alinhamento de ontologias e os alinhadores [5, 6]. Ele se destaca por ter um tempo de execução razoável e possuir um bom desempenho em termos de precisão e cobertura, mesmo não utilizando naquela competição nenhuma estratégia que fizesse análise das estruturas das ontologias (definida pelas relações entre as entidades), nem métodos para lidar com diferentes linguagens naturais.

O AML utiliza diversas técnicas, cada uma bem compartimentada em "*matchers*" ou módulos de alinhamento, e cada estratégia configurada determina (entre outros aspectos) uma sequência destes módulos para produzir o alinhamento final. Outros fatores determinantes na escolha deste projeto para estudo foram a linguagem (Java) conhecida, o ambiente de desenvolvimento conhecido (Eclipse), e ser um dos poucos alinhadores com código fonte aberto. Em particular, o AML usa a Licença Apache 2.0.

1.2 Objetivo

Tendo em vista os problemas de heterogeneidade, as características do AML e com o objetivo também de explorar as diferentes técnicas empregadas para encontrar mapeamentos, propõe-se a implementação de módulos de diferentes naturezas para o *AgreementMakerLight*, aproveitando-se da sua estrutura modular e fácil composição das estratégias de casamento. Estudando, em particular, soluções em algoritmos de distância de edição, exploração de recursos externos e análise estrutural ou semântica. Para isto, estudam-se em especial: as diferentes técnicas de alinhamento existentes e o funcionamento interno do AML (descrito com mais detalhes no próximo capítulo).

1.3 Organização do Trabalho

No capítulo 2 introduz-se o AML, no estágio em que este estava no início do projeto e quando competiu na OAEI de 2013, com objetivo de analisar as técnicas já empregadas. Em seguida, no capítulo 3, descrevem-se as implementações realizadas para este trabalho, e como elas se integram ao projeto original.

O capítulo 4 mostra os resultados obtidos e uma primeira avaliação dos mesmos. Finalmente, tem-se no capítulo 5, as principais considerações finais deste trabalho e propostas de trabalhos futuros relacionados ao mesmo projeto.

[2] http://oaei.ontologymatching.org/

Capítulo 2

AgreementMakerLight

O *AgreementMakerLight* (AML) foi desenvolvido com o objetivo de construir um alinhador similar ao *AgreementMaker*, entretanto, que fosse mais eficiente e modular, em especial, para ontologias com um grande volume de dados. Para tal, ele se concentra em técnicas usualmente classificadas como elementares, que analisam as entidades das ontologias isoladas [4]; também possui como característica marcante o uso de recursos externos [6]. Além disso, utiliza estruturas de dados eficientes e evita aplicar algoritmos cujo gasto de recursos prejudique a sua escalabilidade.

2.1 Estruturas básicas do arcabouço

Para melhor compreensão do trabalho, a seguir apresentam-se algumas das principais estruturas do AML 2.05 (versão corrente na data de início do trabalho):

- `Lexicon`: Estrutura responsável por mapear uma classe a todos os seus possíveis nomes, sejam eles obtidos durante o processo de carregamento das ontologias de entrada (além do próprio, os possivelmente extraídos de anotações), ou por extensão de vocabulário a partir de recursos externos. Além desta associação, também guarda a fonte de onde o nome foi obtido, isto com o objetivo de determinar melhor a confiabilidade da associação. Praticamente todos os algoritmos de alinhamento utilizam este recurso, visto que, em geral as técnicas elementares se envolvem, de forma direta ou indireta com os nomes das entidades (um exemplo claro são os algoritmos que se utilizam de distância entre *strings*); lembrando que as técnicas elementares representam a grande maioria das técnicas presentes no AML, até pelo menos a versão 2.20.

- `Relationship Map`: Em vez de armazenar a ontologia como uma matriz, grafo ou árvore, para representar as relações entre classes (formas mais comuns em outros alinhadores pesquisados [6, 7, 8, 9]), o AML utiliza um conjunto de tabelas de espalhamento que permite acesso bastante eficiente a partir de ancestrais e descendentes. Estas tabelas contêm as relações do tipo "é um" (equivalência) e "parte de" (inclusão) com fecho transitivo, e as de disjunção sem fecho transitivo. Para evitar a perda de informação, indica-se na relação a distância entre as classes; com isso pode-se inclusive obter de forma relativamente eficiente os ancestrais e descendentes em uma vizinhança de certa classe (disto podem se beneficiar certos algoritmos estruturais mais especializados).

- `Aligment`: A cada algoritmo de alinhamento executado, o conjunto de mapeamentos do processo (representado por uma instância da classe `Aligment`), tem de ser atualizado. Atribui-se à esta estrutura a responsabilidade de inserir novos mapeamentos considerando a confiabilidade associada quando há conflitos, o mais eficientemente possível, e também descartando aqueles que não atinjam o nível mínimo de confiança.

- `Matchers`: Cada um dos algoritmos de alinhamento (também chamados de *matchers*), cria, estende ou altera o conjunto de mapeamentos durante a execução. Os autores do AML

categorizam-nos em dois tipos principais, os menos complexos (tendem a gastar tempo $O(n)$, onde n é o número total de entidades) como primários, com os quais pretendem-se criar alinhamentos do zero, ou trabalhar com todas as entidades sem nenhum tipo de filtragem; e os mais complexos, como secundários, que em geral tendem a gastar tempo quadrático, os quais para certos tamanhos de ontologia, não são eficientes, sem antes limitar o número de entidades disponibilizadas (estas limitações incluem: trabalhar somente com parte da ontologia, ou apenas estender ou recalcular o alinhamento).

- `Lexical Matcher`: Neste algoritmo, comparam-se os termos dos `Lexicons` das ontologias de entrada. Primeiramente, toma-se o menor dos conjuntos de nomes (para minimizar o número de iterações), então, para cada termo, se há classes com este nome, em ambas as ontologias, inclui-se um mapeamento de equivalência entre elas, com confiança igual ao produto das confianças das atribuições do nome às classes envolvidas.
- `Mediating Matcher`: Este algoritmo utiliza uma ontologia externa como forma de estender os `Lexicons` com sinônimos. Ambos os `Lexicons` são passados para o `Lexical Matcher`, junto com a ontologia mediadora, obtendo-se dois alinhamentos parciais. Toma-se o menor deles (em número de mapeamentos), e para cada mapeamento, verifica-se se existe no outro alinhamento, um deles com mesma classe envolvida na ontologia mediadora. Nos casos em que isso ocorre, adiciona-se um mapeamento entre as classes da ontologia de entrada usando o mínimo das confianças nos alinhamentos parciais.
- `Word Matcher`: Este método analisa similaridade entre as palavras que compõe o nome das classes, e com base em um índice de Jaccard, determina possíveis correspondências entre classes. Como possui um gasto elevado de memória (requer uma estrutura similar ao `Lexicon`), o AML não o utiliza para ontologias grandes.
- `Parametric String Matcher`: Este algoritmo, classificado como secundário por exigir tempo $O(n^2)$, compara todos os termos entre duas classes dadas para obter a maior similaridade possível utilizando uma das métricas de distância de edição implementadas: ISub (o padrão), Levenshtein, Jaro-Winkler e K-gramas. Ressalta-se também que esse algoritmo busca mapeamentos usando a mesma técnica para os pais, filhos e irmãos de uma dada classe, usando a mesma métrica.
- `WordNet Matcher`: Este módulo utiliza o tesauro *WordNet* para, primeiro estender os `Lexicon` das ontologias de entrada com os sinônimos obtidos, e depois, procura correspondências literais de forma similar ao `Lexical Matcher`, mas considerando apenas os casos em que o termo possui pelo menos um sinônimo advindo do *WordNet*.
- `Property Matcher`: A especialidade deste módulo de alinhamento consiste em descobrir mapeamentos entre propriedades, usando como critério de seleção: tipos, domínios, intervalos e similaridade entre os nomes. Pode usar o *WordNet*, se habilitado e ainda consultar o alinhamento entre domínios, caso estes sejam classes.

2.2 Funcionamento

O primeiro passo consiste em carregar as ontologias e obter as entidades relevantes, para guardá-las em estruturas próprias, em particular: o `URI Map`, o `Relationship Map`, e o `Lexicon` de cada ontologia; estas estruturas serão detalhadas posteriormente, junto ao seu uso mais evidente no processo de alinhamento.

Em seguida, procede-se para o alinhamento propriamente dito, segundo alguma estratégia dentre as enumeradas como `Matching Algorithm`; cada uma delas define qual sequência de algoritmos a ser executada, podendo variar algumas configurações destes. Optar pelo reparo ao final do processo, entre outros detalhes.

Vale notar que a `OAEI2013` implementada na classe `OAEI2013 Matcher`, aparece como uma das estratégias mais importantes e, também como a melhor documentada (até a versão 2.05). Ela

compõe-se de algumas etapas bastante particulares no que diz respeito à configuração do processo, conforme documentado em [5]. Segue uma breve descrição das etapas da estratégia `OAEI2013`:

- **Casamento básico e configuração**: Faz configurações referentes ao tamanho da ontologia, determina o tipo de seleção, decide se haverá casamento de propriedades (observando se em ambas a taxa entre propriedades e classes é maior que 10 %) e executa o `Lexical Matcher` a fim de obter um alinhamento inicial.
- **Alinhar usando recursos externos**: Durante esta fase o AML seleciona o melhor recurso externo dentre: Uberon, UMLS e o WordNet, a partir do ganho em cobertura e do tamanho obtido na etapa anterior, cria `Lexicons` estendidos usando a fonte mais apropriada, e adiciona novos mapeamentos ao alinhamento corrente. De fato, ressalta-se que, como os recursos externos podem induzir erros mais facilmente, eles só trabalham com classes ainda não mapeadas.
- **Métodos extensivos e seleção**: Nesta etapa são utilizados o `Word Matcher` (para ontologias com menos de 60000 classes, dados os requisitos de memória e propensão a erros) e o `Parametric String Matcher` usando como padrão o algoritmo ISub. Os seletores usados nesta etapa buscam reduzir a cardinalidade (executados logo após o `Word Matcher`).
- **Casamento de propriedades**: Esta fase resume-se em executar o `Property Matcher`, se assim decidido na etapa de configuração.
- **Reparo**: Nesta etapa opcional, o AML utiliza heurísticas com o intuito de verificar se as disjunções são respeitadas, e remover o menor número de mapeamentos para resolver os conflitos.

Em testes, avalia-se o resultado com base em um alinhamento de referência. As informações de saída contém: a precisão (do total encontrado, a proporção de mapeamentos corretos), a cobertura (das correspondências corretas, quantos foram encontrados), a medida F1 ($F1 = \frac{2 \times \texttt{precisão} \times \texttt{cobertura}}{\texttt{precisão} + \texttt{cobertura}}$), e a quantidade de mapeamentos encontrados, corretos, e listados no alinhamento de referência.

Capítulo 3

Alterações

Para estudar melhor o *AgreementMakerLight* e aplicar possíveis melhorias, foi desenvolvido o pacote "Extras4AML", cujo objetivo resume-se em concentrar as principais mudanças realizadas no código fonte do AML, diminuindo ao máximo a intervenção nas classes originais. As modificações mais intrusivas no código original foram apenas em classes que integravam os módulos e as estratégias de alinhamento em si, isto foi possível graças à modularidade do AML, cada uma das técnicas descritas neste capítulo foi implementada por uma ou mais classes, as quais foram facilmente integradas ao código existente.

3.1 Level 2 Jaro-Winkler

Ao estudar o artigo [2], o qual compara diversas métricas, nota-se que as estratégias recursivas conseguem um desempenho razoável sem aumentar demasiadamente o tempo de processamento utilizado e nem precisando de uma etapa anterior de pré-processamento, ou treinamento. Por estas razões, foi implementado no Extras4AML, o método recursivo denominado pelos autores como *Level 2 Jaro-Winkler*. Para melhor compreensão, explica-se a construção desta métrica a partir da medida de Jaro.

Dadas duas *strings* s e t, define-se que um caractere s_i casa com t_j, se são o mesmo caractere, e se: $|i-j| \leq \frac{\max(|s|,|t|)}{2} - 1$; seja c_s o número de caracteres de s que casam com algum caractere de t e c_t o análogo. Seja T, a metade da quantidade de caracteres comuns entre s e t, mas que não respeitam a fórmula anterior. Define-se, então:

$$Jaro(s,t) = \frac{1}{3} \cdot (\frac{c_s}{|s|} + \frac{c_t}{|t|} + \frac{c_s - T}{c_s}) \tag{3.1}$$

A medida de Jaro-Winkler utiliza também o comprimento do maior prefixo comum entre s e t, identificado por P, assim tem-se:

$$JaroWinkler(s,t) = Jaro(s,t) + \frac{\max(P,4)}{10} \cdot (1 - Jaro(s,t)) \tag{3.2}$$

Finalmente, considerando uma divisão em *tokens* de $s = a_1...a_N$ e $t = b_1...b_M$, tem-se a medida:

$$Level2JaroWinkler(s,t) = \frac{1}{N} \sum_{i=1}^{N} \max_{j=1}^{M} JaroWinkler(a_i, b_j) \tag{3.3}$$

Ele foi implementado na classe `Level2JaroWinkler`. Entretanto, em vez de construir o algoritmo desde a medida de Jaro, usa-se uma versão do algoritmo Jaro-Winkler, já acessível pela biblioteca

Simmetrics[1], utilizada pelo projeto original para obter as métricas de Jaro-Winkler, K-gramas e Levenshtein.

3.2 Dictionary Matcher

Para melhorar o desempenho do AML quando as ontologias foram escritas em diferentes linguagens naturais, tentou-se utilizar o *Wikcionário*[2], como fonte de dados a exemplo do que faz o *Dbnary* [14]. Apesar de ser uma alternativa bastante cabível por devolver informações extraídas do *Wikcionário* no formato de triplas RDF, o *Dbnary* não atendia aos requisitos, pois gerava arquivos de saída muito grandes, já que havia muito mais informações que o necessário.

Para o Extras4AML, decidiu-se então, utilizar o *wikt2dict* [15], um programa que obtém traduções a partir de dados do *Wikcionário* e cria dicionários em formato bem simples. Também existe a opção de triangularização dos dicionários que era o objetivo de destaque do programa, e que consiste em usar mais de um dicionário para aumentar os pares de tradução entre duas línguas. Entretanto observa-se que a quantidade de ruído afeta significativamente a qualidade do dicionário, logo o uso desta técnica foi descartado.

A implementação parte de objetos da classe `Dictionary Word`, que contém a língua e a forma escrita da palavra representada, e uma tabela de espalhamento mapeando códigos do idioma (`en`, para o inglês, por exemplo) em uma lista de `Dictionary Words` representando as traduções da palavra representada. A classe `Dictionary` (não confundir com a classe de mesmo nome do projeto original, presente em versões mais recentes do AML) responsabiliza-se de tratar os arquivos texto de dicionário, e criar um conjunto de `Dictionary Words` para representá-lo. Além disso, nesta classe estão os métodos responsáveis por obter as traduções de uma palavra desejada. O módulo que propriamente faz o uso do dicionário para obter os mapeamentos é o `Dictionary Matcher`. O princípio assemelha-se com o do *WordNet Matcher*: primeiro estendem-se os `Lexicons`, com as traduções disponíveis, depois procuram-se correspondências usando como métrica o ISub.

Algorithm 1: Extensão do `Lexicon`

```
Entrada: Lexicon L, Dicionário D, Classes C, Real limiar
Saída: Lexicon L
para cada nome n ∈ L faça
    se n não é uma fórmula então
        traduções ← ∅;
        para cada língua l associada ao nome n faça
            traduções ← traduções ∪ traduções de n na língua l em D;
            confiança ← 0.7 − 0.01·|traduções|;
            para cada classe c associada ao nome n faça
                w ← confiança · peso(n, c);
                se w >= limiar então
                    L ← L ∪ traduções;
                fim
            fim
        fim
    fim
fim
```

No algoritmo acima, como em outros módulos de alinhamento, ignoram-se as fórmulas (como $x = y + z$, por exemplo), a fórmula do peso w deriva-se da utilizada no WordNet Matcher, com intuito de equilibrar o valor da fonte de dados, mas considerando que quanto mais traduções, maior

[1] http://sourceforge.net/projects/simmetrics/
[2] https://www.wiktionary.org/

o risco de imprecisões serem inseridas, o valor de confiança é uma constante atribuída de acordo com a confiabilidade da fonte de informação. A função $peso(n, c)$, obtém do `Lexicon` a confiança da associação do nome n à classe c.

Supondo um conjunto de classes C = {onto1#carro, onto2#coche}, um `Lexicon` L = {[onto1#carro, carro (português), 0.8], [onto2#coche, coche (espanhol), 0.75]}, um dicionário D = {[car, inglês] $\iff$ [carro, português] $\iff$ [coche, espanhol)]} e um limiar l = 0.5; temos o conjunto de nomes N = {carro (português), coche (espanhol)}, com n = carro (português), tem-se um único valor para a língua l, que é português, obtém-se então as traduções: *car* (inglês) e *coche* (espanhol). A onto1#carro é a única classe associada ao nome n, calcula-se confiança = $0.7 - 2 * 0.01 = 0.68$ o peso $w = 0.68 \cdot 0.8 = 0.54$, como $0.68 > 0.5$, adicionam-se as entradas [onto1#carro, car (inglês), 0.54] e [onto1#carro, coche (espanhol), 0.54] ao `Lexicon` L. O procedimento é análogo para a classe onto2#coche.

Algorithm 2: `Dictionary Matcher` após extensão

```
Entrada: Ontologia S, Ontologia T, Real limiar
Saída: Alinhamento
Alinhamento ← ∅;
maxSim ← 0.0;
para todo par de classes de ontologias distintas (classeS, classeT) faça
    para cada nome ns nos nomes em S faça
        para cada língua l possível para o par (ns, classeS) faça
            nomesT ← nomes em T, associados à classeT, tal que a língua é l;
            para cada nome nt em nomesT faça
                sim ← peso(ns, classeS) · peso(nt, classeT) · ISub(ns, nt);
                maxSim = max(sim, maxSim);
            fim
        fim
    fim
    se maxSim ≥ limiar então
        Alinhamento ← Alinhamento ∪(classeS, classeT, ≡, maxSim)
    fim
fim
```

O algoritmo ISub calcula distância de edição entre duas cadeias de caracteres, segundo a fórmula:

$$ISub(s,t) = Similaridade(s,t) - Dissimilaridade(s,t) + MelhoriaDeWinkler(s,t,Similaridade(s,t)) \tag{3.4}$$

Calcula-se a similaridade entre s e t iterativamente: a cada passo encontra-se a maior sequência de caracteres comum, remove-se esta das cadeias originais, e obtém-se a maior sequência para estas novas cadeias. Somam-se os comprimentos das sequências encontradas em cada etapa, e finalmente, calcula-se a similaridade com a seguinte equação:

$$Similaridade(s,t) = \frac{2 \cdot SomaDosComprimentos}{|s| + |t|} \tag{3.5}$$

Já a dissimilaridade depende de um parâmetro p, ao qual foi atribuído o valor de 0.6, seguindo a publicação original do algoritmo [12], e é definida pelas equações a seguir:

$$vs = \max(|s| - SomaDosComprimentos, 0)/|s|$$
$$vt = \max(|t| - SomaDosComprimentos, 0)/|t|$$

$$Dissimilaridade(s,t) = \frac{vs \cdot vt}{(p \cdot (1-p)) \cdot ((vs + vt) - vs \cdot vt)} \tag{3.6}$$

A melhoria de Winkler, consiste numa forma de aumentar a importância da similaridade sobre a dissimilaridade, definida na equação a seguir, onde P é o comprimento do maior prefixo comum entre s e t:

$$MelhoriaDeWinkler(s, t, similaridade) = \max(4, P) \cdot 0.1 \cdot (1 - similaridade(s, t)) \quad (3.7)$$

Como exemplo, suponha um par de classes de ontologias distintas: (onto1#carro, onto2#coche), sendo os nomes associados à primeira: carro (português), *coche* (espanhol) e *car* (inglês), e à segunda apenas *coche* (espanhol) e *car* (inglês). Se l = português, o conjunto $nomesT$ é vazio, mas se l = inglês ou l = espanhol, calcula-se a similaridade, como é descrito no algoritmo anterior. Supondo que com l = inglês tenha-se similaridade 0.6 e quando $l = espanhol$ similaridade 0.71, para um limiar de 0.65 (limiar menor ou igual à similaridade), cria-se um mapeamento (onto1#carro, onto2#coche, $\equiv, 0.71$).

Vale ressaltar que qualquer fonte pode servir de dicionário, basta apenas que o arquivo esteja em um dos formatos aceitos:

- Todas as traduções em um único arquivo em quatro colunas, separadas por espaço, na ordem: língua da primeira palavra, forma escrita da primeira palavra, língua da segunda palavra, forma escrita da segunda palavra.
- Um arquivo cujo nome esteja no formato `<língua de origem>-<língua de destino>.txt`, contendo duas colunas, separadas por espaço, sendo a primeira de palavras na língua de origem, e a segunda das correspondentes na língua de destino.

Outro fato importante foi o desenvolvimento de um módulo de tradução no projeto original do AML[3], cuja ideia também consiste em usar dicionários, mas estes são mais específicos, com um nível de ruído muito menor, e o mais notável, caso uma palavra não seja encontrada nos dicionários locais, faz-se uma consulta usando a interface pública do *Bing* (da *Microsoft*), a fim de obter uma lista de traduções.

3.3 Descendant Analyser Matcher

Muitos alinhadores como ASMOV, S-Match, entre outros [4, 7, 9], possuem estratégias estruturais ou baseadas em inferência, como complemento das técnicas elementares ou mesmo como algoritmo principal. No caso do AML, entretanto, até a versão 2.05, a única técnica parcialmente estrutural estava localizada no `Parametric String Matcher`, aplicando a mesma métrica de similaridade elementar nas redondezas (pais, filhos e irmãos na hierarquia) de um alinhamento já existente. Isto se deve ao fato de que as estratégias estruturais e semânticas tem custo, em geral, bastante elevado, visto que podem exigir multiplicações de grandes matrizes, resolver problemas de casamento de grafos ou ter de realizar muitas inferências.

Tendo isto em mente e seguindo uma ideia similar de localidade usada no `Parametric String Matcher`, implementou-se um algoritmo que, para ter tempo de computação condizente com o alinhador, também trabalha sobre um conjunto previamente fornecido de mapeamentos, mas sem fazer uso de técnicas que meçam distância entre cadeias de caracteres. Vale ressaltar também duas outras motivações para o desenvolvimento deste módulo: aproveitar mais os recursos do `Relationship Map` e ter algum algoritmo capaz de inserir mapeamentos com relações que não fossem equivalências.

Este algoritmo encontra-se na classe `Descendant Analyser Matcher`, o qual tem funcionamento geral descrito pelo pseudocódigo a seguir:

[3] https://github.com/AgreementMakerLight/AML-Project

Algorithm 3: Visão geral do `Descendant Analyser Matcher`

Entrada: Ontologia S, Ontologia T, Real limiar, Alinhamento A
Saída: Alinhamento
maxSim ← 0.0;
para *cada mapeaemento (classeS, classeT) in A* **faça**
  relação ← avaliaçãoEstrita(A, irmãos(classeS), irmãos(classeT));
  mapeamentos ← (i, j) *in* A tal que similaridade(i, j) > 0.0, i *in* irmãos(classeS), j *in* irmãos(classeT);
  sim ← $\sum_{(i,j)\in mapeamentos} similaridade(i, j)$;
  N ← |mapeamentos|;
  se $N \neq 0$ **então**
    sim ← sim/N;
  fim
  para cada *paiS em pais(classeS)* **faça**
    para cada *paiT em pais(classeT)* **faça**
      se *relação = desconhecida* **então**
        relação ← avaliaçãoRelaxada(paiS, paiT, N);
      fim
      se *relação ≠ desconhecida* **então**
        A ← A ∪ (paiS, paiT, sim, relação);
      fim
    fim
  fim
  se *maxSim ≥ limiar* **então**
    Alinhamento ← Alinhamento ∪$(classeS, classeT, rel, maxSim)$;
  fim
fim

A função de similaridade retorna do alinhamento dado como entrada, a maior medida de confiança entre os mapeamentos que existem entre as duas classes passadas como argumento. A avaliação estrita tenta determinar uma possível relação a partir da veracidade de duas afirmações, considerando x e y classes tal que x e y estão em ontologias distintas:

1. todas as classes filhas de x tem algum mapeamento com uma classe filha de y.
2. todas as classes filhas de y tem algum mapeamento com uma classe filha de x.

Se ambas as afirmações são verdadeiras, ela infere que x e y, são equivalentes, se só a primeira é verdadeira considera que x é uma especialização de y, se apenas a segunda estiver válida, x é uma generalização de y. Caso contrário a relação é dada como desconhecida. Já a avaliação relaxada, tenta abrir espaço para alguns erros, podendo piorar a precisão, mas com maiores chances de aumentar a cobertura (a qual tem-se por prioridade no algoritmo). Ela trabalha com margens nos valores em vez de considerar afirmações completas como as vistas anteriormente.

Considerando o exemplo da figura 3.1, ao escolher qualquer um dos mapeamentos de equivalência existentes entre as subclasses de `Usuário` e `Pessoa`, faz-se a avaliação estrita (que, neste caso, deve devolver que `Usuário` é subclasse de `Pessoa`), calcula-se a similaridade por $similaridade = \frac{0.98+0.75+0.98+0.55}{4} = 0.8125$. No exemplo, cada conjunto de irmãos tem apenas um pai em comum, assim se o limiar for menor ou igual à 0.8125, é inserida uma relação $\leq$ entre `Usuário` e `Pessoa`. Se por acaso, a avaliação estrita não tenha identificado nenhuma relação, utiliza-se a relaxada, descrita anteriormente.

Este módulo trabalha apenas com as relações do tipo "é um". Diversas complicações foram encontradas com outras relações como a "parte de" ao se tentar aplicar o mesmo raciocínio, ainda

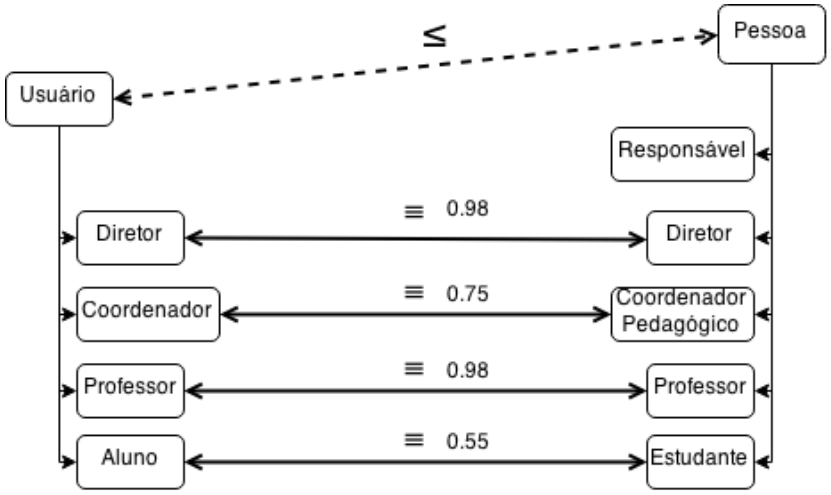

Figura 3.1: *Representação gráfica de classes e subclasses de ontologias distintas*

que também fossem transitivas, dentre elas o fato de não possuírem um forma padrão de serem descritas, ao contrário das relações "é um" que possuem indicação clara de classes e subclasses, e de possuir diferentes interpretações.

Capítulo 4

Resultados/Avaliação

Para avaliar se o pacote desenvolvido neste trabalho, o Extras4AML, conseguiu melhorar o desempenho do alinhador, foram criadas versões das estratégias originais na versão 2.20, acrescidas de extensão do léxico com o `Dictionary Matcher`, usando o dicionário extraído do Wikcionário, obtido com o *wikt2dict* usando apenas o idioma inglês, e tendo como último módulo de alinhamento o `Descendant Analyzer Matcher`. Além disso, para testar o `Level 2 Jaro-Winkler`, foi criado um arquivo jar especial, trocando todas as chamadas ao `ISub`, por equivalentes do `Level 2 Jaro-Winkler`.

A classe responsável pela execução dos testes encontra-se em `AMLExtrasTest.java`. A partir dela cria-se um arquivo jar, o qual deve estar localizado na pasta raiz do projeto (no mesmo diretório que as pastas `src` e `store`). A saída pode ser processada posteriormente usando o *script* parser.py que gera um arquivo csv, o qual pode ser analisado das mais diversas formas. Os testes utilizados foram obtidos na página oficial da OAEI (http://oaei.ontologymatching.org/tests/), na qual também pode-se ler a descrição de cada um dos casos disponibilizados.

Analisando a influência do uso dos Extras4AML nas tabelas acima, nota-se que há uma troca de precisão por cobertura, e melhorias na medida F1, ainda assim os ganhos não foram tão expressivos quanto o esperado. A precisão caiu significativamente, enquanto a cobertura nem sempre teve um ganho equivalente e em nenhum caso o aumento em medida F1 foi maior que 0,2%.

Outra observação importante reside na combinação da estratégia `OAEI2013`, com e sem o pacote desenvolvido neste projeto, e o Jaro-Winkler de nível 2. As estratégias `AML` e `OAEI2013` diferem essencialmente nos parâmetros de configuração utilizados, visto que executam os mesmos algoritmos de alinhamento. Enquanto a `OAEI2013` possui uma etapa específica de autoconfiguração (para decidir limiares, casamento de propriedades, entre outras) mais sofisticada, a estratégia `AML` utiliza valores fixos pré-determinados.

A estratégia `Lexical` não usa de fato nenhuma distância de edição, justificando os resultados iguais em ambos os casos, exceto quando executada com o Extras4AML, já que o `Dictionary Matcher` utiliza o algoritmo corrente para calcular similaridades.

Os casos de teste 301, 302, 303 e 304, mostrados na tabela 4.3, destacam-se por terem como ontologias de entrada, a referência, comum à todos os casos, e uma ontologia obtida de casos reais. Verifica-se a repetição da tendência das tabelas gerais, mas com a compensação entre precisão e cobertura mais acentuados e ganhos um pouco mais visíveis na medida F1.

Estratégia	Precisão (%)		Cobertura (%)		Medida F1 (%)	
	Média	*Desvio Padrão*	*Média*	*Desvio Padrão*	*Média*	*Desvio Padrão*
AML	81,9414	36,7147	31,4981	28,8612	41,4333	29,9261
AML + Extras4AML	80,1252	36,0491	32,0072	29,2459	41,5081	29,7345
Lexical	81,3432	36,9003	31,2108	29,0192	41,0081	30,2148
Lexical + Extras4AML	79,3090	36,2140	31,7180	29,4045	41,0297	29,9712
OAEI2013	66,1414	28,3022	44,7126	34,3230	49,8153	31,3844
OAEI2013 + Extras4AML	65,3027	27,9303	45,2252	34,5523	49,8765	31,2894
Geral	75,6938	34,5281	36,0620	31,5236	44,1118	30,5836

Tabela 4.1: *Resultados usando ISub como distância de edição*

Estratégia	Precisão (%)		Cobertura (%)		Medida F1 (%)	
	Média	*Desvio Padrão*	*Média*	*Desvio Padrão*	*Média*	*Desvio Padrão*
AML	82,1324	36,7036	31,6945	28,7219	41,7450	29,6853
AML + Extras4AML	80,3675	36,0629	32,2027	29,1033	41,8261	29,4877
Lexical	81,3432	36,9003	31,2108	29,0192	41,0081	30,2148
Lexical + Extras4AML	79,3090	36,2140	31,7180	29,4045	41,0297	29,9712
OAEI2013	42,2603	21,6618	45,5270	34,2314	40,8396	24,3892
OAEI2013 + Extras4AML	42,0792	21,5910	46,0333	34,4556	40,9927	24,4215
Geral	67,9153	36,9800	36,3977	31,5162	41,2402	28,0427

Tabela 4.2: *Resultados usando Level 2 Jaro-Winkler como distância de edição*

Estratégia	Precisão (%)		Cobertura (%)		Medida F1 (%)	
	Média	*Desvio Padrão*	*Média*	*Desvio Padrão*	*Média*	*Desvio Padrão*
AML	89,3500	4,18369	27,9500	7,4065	42,1000	8,3538
AML + Extras4AML	83,1250	9,19759	30,2250	5,9382	43,8250	5,566
Lexical	89,9750	2,3796	26,1500	6,8690	40,1250	7,9399
Lexical + Extras4AML	83,0250	11,6021	28,3750	5,7488	41,6750	4,4806
OAEI2013	88,1500	5,2080	31,6250	12,3126	45,3500	12,6452
OAEI2013 + Extras4AML	82,9250	9,4711	33,8750	10,8910	46,9750	9,7592
Geral	86,0916	7,5476	29,7000	8,0136	43,3416	7,9280

Tabela 4.3: *Resultados para os testes 301, 302, 303 e 304 (ISub)*

Capítulo 5

Considerações Finais

Por meio deste trabalho, pode-se entender algumas das dificuldades relacionadas ao problema de alinhamento de ontologias. Dentre elas, as explicitadas em [3], como a necessidade de soluções criativas para obter ganhos expressivos, a importância de um alinhador bem configurado, a dificuldade em aumentar a qualidade do alinhamento dentro de um tempo aceitável, entre outros.

Com os estudos que foram feitos concorrentemente ao desenvolvimento do Extras4AML, nota-se que as estratégias baseadas em recursos externos são bastante promissoras, principalmente quando estes são mais formais, como ontologias já reconhecidas, dicionários e tesauros bem construídos, entre outros. Isto se deve ao fato de serem fontes de informação com ruído reduzido e uma solução bastante eficaz para aumentar a cobertura dos alinhadores, sem ter de recorrer à custosos métodos estruturais. Além disso, considerando o objetivo de integrar dados e informações com uso de ontologias, mostra-se como uma estratégia natural a ser explorada.

Trabalhando sobre o código do AML, e com a análise da documentação de outros alinhadores, observa-se que apesar de algumas soluções serem bastante comuns (como uso do *WordNet* e distância de edição), alguns alinhadores retém particularidades interessantes, resultando em uma gama de técnicas que vão desde algoritmos de casamento de grafos, ao uso de motores de inferência, aprendizagem de máquina entre outras [3, 8].

Como proposta de trabalhos futuros, já de forma imediata, tem-se a necessidade de melhorar a eficiência do `Descendant Analyzer Matcher`, para evitar recalcular relações de elementos que já tenham sido processados em etapas anteriores, e utilizar técnicas mais conhecidas para determinar os parâmetros da avaliação relaxada, possivelmente tornando os valores utilizados como margem mais dinâmicos, variando de acordo com as características das ontologias de entrada.

Outra proposta consiste no desenvolvimento de técnicas estruturais secundárias, ainda com o objetivo de melhorar a cobertura, em especial para lidar com as relações que não sejam de equivalência, como também é proposto em [11].

Para continuidade do projeto, tem-se como uma importante tarefa a integração do Extras4AML com a plataforma SEALS, utilizada atualmente pela OAEI e pelos desenvolvedores de alinhadores para testar e avaliar resultados dos programas. Com este mesmo propósito, uma meta futura resume-se em manter este módulo atualizado com o projeto original, a fim de servir como um complemento ou variação consistente e com mínimo esforço de integração possível.

Finalmente, acredita-se que a área ainda tenha muito em que avançar para que o uso de alinhadores de ontologias se torne uma alternativa utilizada em grande escala em ambientes tão diversos e com tantos requisitos, tendo como exemplo a internet. Além disso, os módulos desenvolvidos deixam claro que ainda há espaço para melhorias. Entretanto, para conseguir avanços significativos podem ser necessárias soluções que interfiram mais no arcabouço do *AgreementMakerLight*.

Referências Bibliográficas

[1] **Caimi(2012)** Federico Caimi. *Ontology and Instance Matching for the Linked Open Data Cloud*. Tese de Doutorado, University of Illinois at Chicago. Citado na pág. 1

[2] **Cohen** *et al.***(2003)** W. Cohen, P. Ravikumar e S. Fienberg. A comparison of string distance metrics for name-matching tasks. *Proceedings of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*. Citado na pág. 9

[3] **Euzenat e Shvaiko(2013)** Jérôme Euzenat e Pavel Shvaiko. Ontology Matching: State of the Art and Future Challenges. *Knowledge and Data Engineering, IEEE Transactions on*, 25 (1):158–176. ISSN 1041-4347. doi: 10.1109/TKDE.2011.253. Citado na pág. 1, 17

[4] **Euzenat e Shvaiko(2007)** Jérôme Euzenat e Pavel Shvaiko. *Ontology matching*. Springer-Verlag, Heidelberg (DE), 2nd edição. Citado na pág. 1, 5, 12

[5] **Faria** *et al.***(2013)** Daniel Faria, Catia Pesquita, Emanuel Santos, Isabel F. Cruz e Francisco M. Couto. AgreementMakerLight Results for OAEI 2013. *ISWC International Workshop on Ontology Matching*. URL http://disi.unitn.it/~p2p/OM-2013/oaei13_paper1.pdf. Citado na pág. 1, 3, 7

[6] **Faria** *et al.***(2013)** Daniel Faria, Catia Pesquita, Emanuel Santos, Matteo Palmonari, Isabel F. Cruz e Francisco M. Couto. The AgreementMakerLight Ontology Matching System. Em *On the Move to Meaningful Internet Systems: OTM 2013 Conferences*, volume 8185 of *Lecture Notes in Computer Science*, páginas 527–541. Springer Berlin Heidelberg. ISBN 978-3-642-41029-1. doi: 10.1007/978-3-642-41030-7_38. URL http://dx.doi.org/10.1007/978-3-642-41030-7_38. Citado na pág. 3, 5

[7] **Giunchiglia** *et al.***(2004)** Fausto Giunchiglia, Pavel Shvaiko e Mikalai Yatskevich. S-match: an algorithm and an implementation of semantic matching. *In Proceedings of ESWS*. Citado na pág. 5, 12

[8] **Hu** *et al.***(2005)** Wei Hu, Ningsheng Jian, Yuzhong Qu e Yanbing Wang. GMO: A Graph Matching for Ontologies. *K-Cap 2005 Workshop on Integrating Ontologies 2005*. Citado na pág. 5, 17

[9] **Jean-Mary** *et al.***(2009)** Yves R. Jean-Mary, E. Patrick Shironoshita e Mansur R. Kabuka. Ontology Matching with Semantic Verification. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):235 – 251. ISSN 1570-8268. doi: http://dx.doi.org/10.1016/j.websem.2009.04.001. URL http://www.sciencedirect.com/science/article/pii/S1570826809000146. Citado na pág. 5, 12

[10] **Shvaiko e Euzenat(2005)** Pavel Shvaiko e Jérôme Euzenat. A Survey of Schema-Based Matching Approaches. *Journal on Data Semantics IV*, páginas 146–171. doi: 10.1007/11603412_5. URL http://dx.doi.org/10.1007/11603412_5. Citado na pág. 1

[11] **Spiliopoulos** *et al.***(2010)** Vassilis Spiliopoulos, George A. Vouros e Vangelis Karkaletsis. On the discovery of subsumption relations for the alignment of ontologies. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(1):69 – 88. ISSN 1570-8268.

doi: http://dx.doi.org/10.1016/j.websem.2010.01.001. URL http://www.sciencedirect.com/science/article/pii/S1570826810000028. Citado na pág. 17

[12] **Stoilos *et al.*(2005)** Giorgos Stoilos, Giorgos Stamou e Stefanos Kollias. A string metric for ontology alignment. *ISWC International Workshop on Ontology Matching*, 3729:624–637. doi: 10.1007/11574620_45. URL http://dx.doi.org/10.1007/11574620_45. Citado na pág. 11

[13] **Sunna e Cruz(2007)** William Sunna e Isabel F. Cruz. Structure-Based Methods to Enhance Geospatial Ontology Alignment. Em *GeoSpatial Semantics*, volume 4853 of *Lecture Notes in Computer Science*, páginas 82–97. Springer Berlin Heidelberg. ISBN 978-3-540-76875-3. doi: 10.1007/978-3-540-76876-0_6. URL http://dx.doi.org/10.1007/978-3-540-76876-0_6. Citado na pág. 1

[14] **Sérasset(2012)** Gilles Sérasset. Dbnary: Wiktionary as a LMF based Multilingual RDF network. *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*. Citado na pág. 10

[15] **Ács(2014)** Judit Ács. Pivot-based Multilingual Dictionary Building using Wiktionary. Em *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland. European Language Resources Association (ELRA). ISBN 978-2-9517408-8-4. Citado na pág. 10