INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Detecção de Duplicados em Bases de Dados XML

Luís Miguel Gomes dos Santos Reis Leitão

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática e de Computadores

Júri

Presidente: Prof. Pedro Sousa
Orientador: Prof. Pável Calado
Co-orientador: Profª. Helena Galhardas
Vogal: Prof. Arlindo Oliveira

Setembro 2007

Resumo

A detecção de duplicados consiste em identificar múltiplas representações de objectos do mundo real armazenados numa fonte de dados. Esta é uma tarefa de grande relevância prática na limpeza de dados, *data mining* ou integração de dados, possuindo um longo percurso no que diz respeito a dados relacionais armazenados numa única tabela (ou em múltiplas tabelas com o mesmo esquema). Os algoritmos para a detecção de duplicados em estruturas mais complexas como, por exemplo, hierarquias de uma *data warehouse*, dados XML ou dados em grafos só recentemente emergiram. Estes algoritmos usam medidas de similaridade que consideram se os seus vizinhos directos são duplicados, por exemplo, os filhos em modelos de dados hierárquicos, para melhorar a eficácia da detecção de duplicados. Neste trabalho, é proposto um novo método para a detecção de duplicados em dados XML hierárquicos e semi-estruturados. Ao contrário de abordagens anteriores, este método não considera apenas se os filhos são duplicados, mas considera também a probabilidade de todos os descendentes serem duplicados. As probabilidades são calculadas eficientemente usando uma rede Bayesiana. Testes mostram que o algoritmo proposto é capaz de manter valores altos de precisão e *recall*, mesmo perante dados que contenham uma grande quantidade de erros e informação em falta. O método aqui proposto é ainda capaz de apresentar melhores resultados do que um sistema de detecção de duplicados, que expressa o estado da arte neste domínio, em três bases de dados XML diferentes.

Palavras-Chave: XML, Detecção de duplicados, Redes Bayesianas, Bases de dados

Abstract

Duplicate detection aims at identifying multiple representations of real-world objects stored in a data source, and is a task of critical practical relevance in data cleaning, data mining or data integration. It has a long history for relational data stored in a single table (or in multiple tables with equal schema). Algorithms for duplicate detection in more complex structures, e.g., hierarchies of a data warehouse, XML data or graph data have only recently emerged. These algorithms use similarity measures that consider the duplicate status of their direct neighbours, e.g., children in hierarchical data, to improve duplicate detection effectiveness. In this document, a novel method for duplicate detection in hierarchical and semi-structured XML data is proposed. Unlike previous approaches, it not only considers the duplicate status of children, but rather the probability of descendants being duplicates. Probabilities are computed efficiently using a Bayesian network. Experiments show the proposed algorithm is able to maintain high precision and recall values, even when dealing with data containing a high amount of errors and missing information. This proposal is also able to outperform a state-of-the-art duplicate detection system on three different XML databases.

Keywords: XML, Duplicate detection, Bayesian networks, Databases

Índice

Lista de Abreviações

BD – Base(s) de Dados

HTML – *Hypertext Markup Language*

IDF – *Inverse Document Frequency*

IMDB – *Internet Movie Database*

PC – Probabilidade Condicional

RB – Rede(s) Bayesiana(s)

RI – Recuperação de Informação

XML – *Extensible Markup Language*

Lista de Figuras

Lista de Tabelas

1 Introdução

Nesta secção é dado a conhecer ao leitor o contexto em que se insere o problema abordado neste trabalho, bem como as suas aplicações práticas (Secção 1.1). Na Secção 1.2 é apresentado o problema base estudado neste projecto e, seguidamente, na Secção 1.3, é indicada a abordagem proposta para a resolução do mesmo. Finalmente, na Secção 1.4, é feita uma breve descrição da forma como o documento está organizado.

1.1 Contexto

A limpeza de dados é um processo que visa detectar e corrigir anomalias em fontes de dados. Tais anomalias podem ser motivadas por diversos tipos de situações, como erros tipográficos, diferenças na formatação ou existência de várias representações de uma entidade. Existem actualmente diversas abordagens relativas ao processo de limpeza de dados em bases de dados relacionais [1, 2, 3, 4]. No entanto, devido à rápida popularização do XML, em particular devido ao seu uso na Internet, são cada vez mais procuradas soluções para a resolução deste problema neste modelo de dados específico.

Uma sub-tarefa integrante do processo de limpeza de dados é a detecção de duplicados. A detecção de duplicados tem por objectivo identificar múltiplas representações inconsistentes do mesmo objecto do mundo real. Este problema tem sido amplamente explorado e é referido por diversos autores através de variadas designações, tais como *record-linkage* [5], *merge/purge* [6], *entity resolution* [7], *reference reconciliation* [8], *hardening soft databases* [9], *reference matching* [10], *entity-name clustering and matching* [11], *de-duplication* [12], *object identification* [13], *identity uncertain* [14], *object matching* [1], *object consolidation* [15] e *duplicate detection* [16,17] entre outras.

A detecção de duplicados possui grande relevância e tem sido adoptada em diversas áreas onde é necessário determinar que entidades correspondem ao mesmo objecto. No campo da visão, por exemplo, é frequente a necessidade de determinar se duas formas, que aparecem em tempos diferentes de um segmento de vídeo, são, de facto, o mesmo objecto. No processamento e extracção de informação em língua natural, uma acção fundamental consiste em identificar que frases se referem à mesma entidade. Na criação de bases de dados bibliográficas digitais é fundamental determinar que citações correspondem ao mesmo artigo. Na agregação de bases de dados, uma prática bastante corrente no meio científico e empresarial, em que bases de dados distintas devem ser unidas numa só, os objectos

duplicados provenientes das diversas fontes têm de ser identificados, de modo a evitar redundâncias na base de dados final.
De facto, o domínio que mais requer uma aplicação eficaz da detecção de duplicados é o das bases de dados. Tal é especialmente verdade no meio empresarial, onde a necessidade de proceder a migrações e integrações de dados, muitas vezes provenientes de diferentes fontes, é uma constante. Muitas empresas assentam o seu negócio na relação com os clientes. O facto de existir informação duplicada nas suas bases de dados pode resultar numa deficiência no funcionamento do negócio, gerando problemas tais como duplicações no envio de correspondência, ou até tributações levadas a cabo mais de uma vez. Outro exemplo de uma situação indesejável no meio empresarial está relacionado com actividades de *data-mining*. Inúmeras empresas possuem dados dos quais o seu modelo de negócio depende. Esses dados podem representar, por exemplo, informação comercial de produtos adquiridos por clientes, datas de aquisição, etc. Antes que os dados possam ser analisados para fins estatísticos, estes têm de ser tratados de modo a corrigir anomalias como as mencionadas anteriormente. Assim, a única maneira das empresas garantirem o rigor dos resultados é através de métodos de pré-processamento de dados que devem incluir, entre outros, a detecção de duplicados e eliminação de duplicados. Outros domínios exigentes neste aspecto são a bioinformática ou, fundamentalmente, qualquer outro em que dados obtidos independentemente necessitem ser integrados.

O problema da detecção de duplicados tem sido alvo de um estudo alargado por parte da comunidade científica e empresarial. Porém, esta investigação intensiva foca maioritariamente dados armazenados em tabelas, como é o caso das bases de dados relacionais, onde um objecto é definido como um tuplo, sempre com a mesma estrutura. Este modelo de dados linear torna o problema bastante mais simples de lidar do que, por exemplo, num modelo hierárquico, ou que possua uma estrutura menos rígida. Tal discrepância na estruturação entre os dois modelos inviabiliza a aplicação directa das abordagens convencionais a modelos de dados como o XML. Os motivos para a complexidade adicional do modelo de dados hierárquico são apresentados posteriormente neste documento.

1.2 Definição do Problema

O objectivo da detecção de duplicados em XML é identificar elementos que representem o mesmo objecto no mundo real. Como exemplo, podemos considerar a representação em árvore, de dois elementos XML, ilustrada na Figura 1. Ambas as árvores representam elementos XML com o nome *mv* (movie). Estes elementos têm dois atributos, *year* e *title*. Estes aninham mais elementos XML que representam realizadores (*dr*) e elenco (*cst*). Um elenco é composto por vários actores (*ac*), representados por elementos XML filhos de *cst*. Os

elementos *year*, *title*, *dr* e *ac* contêm um valor textual. Por exemplo, *year* contém o valor "1983". Neste exemplo, o objectivo da detecção de duplicados é verificar se os dois objectos se referem ao mesmo filme, apesar do nome do realizador e dos actores estar representado de modo diferente.

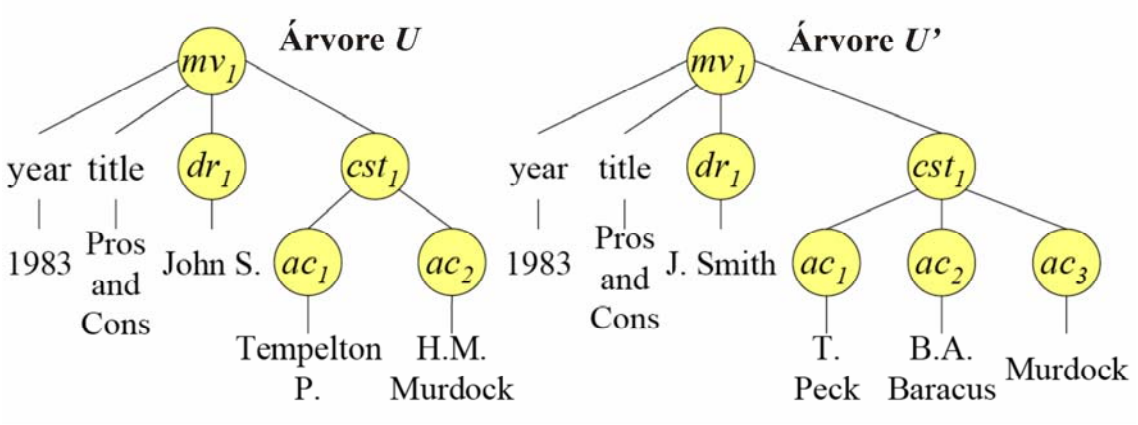


Figura 1: Duas árvores XML. Cada uma representa um filme (*mv*) contendo realizadores (*dr*), um elenco (*cst*) e actores (*ac*).

1.3 Abordagem Proposta

A abordagem proposta consiste num novo método para detecção de duplicados em dados XML. Embora a detecção de duplicados em XML represente um maior desafio do que a detecção de duplicados em bases de dados relacionais devido a, por exemplo, instâncias do mesmo tipo de objecto poderem possuir uma estrutura diferente, por outro lado, a detecção de duplicados em XML permite que a sua estrutura hierárquica seja utilizada para alcançar uma maior eficiência e/ou eficácia.

O método de detecção de duplicados proposto usa um modelo de rede Bayesiana para calcular a probabilidade de dois objectos XML serem duplicados. Para tal é considerado o seguinte pressuposto: ***o facto de dois nós serem duplicados depende apenas do facto dos seus valores serem duplicados e dos seus filhos serem duplicados***. Isto significa que filmes representados por nós com a etiqueta *mv* são ou não duplicados, dependendo dos valores dos seus filhos (*dr* e *cst*) e dos valores dos atributos *year* e *title* serem ou não duplicados. Adicionalmente, os nós identificados como *dr* são duplicados ou não, dependendo dos seus valores serem ou não duplicados. Os nós identificados como *cst* são duplicados ou não, dependendo dos seus filhos (*ac*) serem ou não duplicados. Este processo continua, recursivamente, até os nós terminais (folhas) serem alcançados. Se considerarmos as árvores *U* e *U'* da Figura 1, este processo pode ser representado por uma Rede Bayesiana.

O modelo é construído automaticamente, baseando-se na estrutura dos objectos comparados. Como a rede construída não contém ciclos, a probabilidade de duplicação pode ser determinada eficientemente. Esta abordagem não só fornece uma forte base formal para a solução proposta, como é também suficientemente flexível para se adaptar facilmente a outras bases de dados de diferentes domínios. Adicionalmente, para além de poder funcionar automaticamente, o modelo permite também a introdução de conhecimento humano por parte do utilizador, permitindo assim parametrizar e refinar o modo como os objectos são comparados.

1.4 Organização do Trabalho

Ao longo da Secção 2 é dada uma perspectiva do trabalho existente sobre detecção de duplicados. De seguida, na Secção 3 são apresentados alguns conceitos gerais sobre o tema da detecção de duplicados, bem como XML e redes Bayesianas, com o objectivo de fornecer uma base de compreensão mais sólida para a leitura deste documento. Na Secção 4 é ilustrado o modo como o modelo proposto é construído na prática e são indicadas as suas potencialidades e limitações. Testes experimentais ao modelo são apresentados na Secção 5. Nesta secção, juntamente com os testes realizados para testar a eficácia do modelo proposto, é também efectuado um teste comparativo com um sistema de detecção de duplicados já existente. Finalmente, conclusões e trabalho futuro a desenvolver no seguimento deste projecto são apresentados nas Secções 6 e 7.

2 Trabalho Relacionado

Nesta secção são apresentados alguns estudos desenvolvidos no âmbito da detecção de duplicados. Em 2.1 são referidos os trabalhos dos quais este mais se aproxima e, em seguida, na Secção 2.2 é dada uma perspectiva mais global de todo o trabalho que tem sido desenvolvido sobre o tema da detecção de duplicados. Na Secção 2.3 são mencionadas algumas abordagens e modelos de dados adoptados na resolução deste problema.

2.1 Redes Bayesianas

A abordagem seguida neste trabalho está relacionada com aplicações anteriores de redes Bayesianas na área da Recuperação de Informação (RI). Os modelos de redes Bayesianas foram inicialmente introduzidos em RI por Turtle e Croft [18], com o propósito de ordenar os resultados de procuras. No seu modelo, termos de documentos, consultas e utilizadores são vistos como eventos e são representados como nós de uma rede Bayesiana. O modelo adopta um ponto de vista no qual a observação de um documento induz conhecimento no seu conjunto de termos indexados, e a especificação de tais termos induz conhecimento numa consulta de utilizador ou necessidade de informação. Posteriormente, um segundo modelo foi proposto por Ribeiro-Neto e Muntz [19], demonstrando que as redes Bayesianas podem ser eficazmente usadas para combinar diferentes tipos de informação com vista a melhorar os resultados de buscas. Mais recentemente, Acid *et al*. [20] apresentaram um terceiro modelo no qual a topologia da rede é definida de modo a que um algoritmo de propagação exacto possa ser usado para calcular eficientemente as probabilidades relevantes dos documentos. As redes Bayesianas têm também sido aplicadas a outros problemas de RI, que vão desde *relevance feedback* [21] a *clustering* e classificação de documentos [22]. Neste trabalho, as redes Bayesianas são aplicadas para detectar duplicados em dados XML hierárquicos e semi-estruturados de forma eficaz.

2.2 Detecção de Duplicados

A detecção de duplicados, originalmente definida por Newcombe *et al*. [23] e formalizada por Fellegi e Sunter [24], tem sido extensamente estudada sob vários aspectos. De um modo geral, a investigação em detecção de duplicados recai em duas categorias: técnicas para melhorar a eficácia e técnicas para melhorar a escalabilidade, tanto no espaço (dimensão dos dados processados) como no tempo. Na Tabela 1, estão resumidos alguns métodos de detecção de

duplicados que são classificados segundo duas dimensões, modelo de dados e abordagem utilizada. Os modelos de dados estão diferenciados entre (i) dados numa tabela de relações, (ii) dados em árvore e (iii) dados representados por um grafo. A segunda dimensão da tabela distingue entre três abordagens utilizadas para a detecção de duplicados: (i) aprendizagem (*machine learning)*, onde os modelos e as medidas de similaridade são aprendidos, (ii) uso de técnicas de agrupamento (*clustering)* e (iii) algoritmos que iterativamente detectam pares de duplicados. Na Tabela 1 é também referido se um artigo se foca, essencialmente, na escalabilidade (S) ou na eficácia (E).

	Tabela	**Árvore**	**Grafo**
Aprendizagem	Bill.03[25](E) Sar.02[12](E,S) Doan.03[26](E)		Sin.05[27](E) Bhat.06[28](E)
Agrupamento			Kala.06[29](E,S) Yin.06[30](E,S) Chen05[15](E) Lee05[31](E,S) Bhatt.05[7](E,S)
Iterativos	Her.95[6](S) Mon.97[16](S) Chau.05[32](E,S) Jin.03[33](E,S)	Ana.02[34](E,S) Puh.06[35](S) Mil.06[36](E) Weis.04[37](E,S) Weis.05[17](E)	Dong.05[8](E) Weis.06[38](E,S) Bhat.04[39](E)

Tabela 1: Resumo das abordagens seguidas para detecção de duplicados.

O trabalho aqui apresentado pode ser classificado como um algoritmo iterativo, utilizando uma estrutura de dados em árvore. Apesar das técnicas de aprendizagem poderem ser utilizadas para melhorar o modelo de probabilidades, tal estudo está fora do âmbito deste trabalho. A novidade deste algoritmo, quando comparado com os restantes, é considerar a probabilidade dos descendentes de qualquer nó serem duplicados, em lugar de considerar apenas o estado dos nós mais próximos (filhos).

O único método que considera na totalidade a sub-estrutura e considera algo mais do que apenas o estado dos descendentes é a medida de similaridade proposta por Milano *et al*. [36]. Para comparar dois elementos XML candidatos, é calculada a sobreposição máxima entre as árvores correspondentes. Nesta sobreposição, dois nós que não sejam folhas, podem ser correspondentes se forem antecessores de duas folhas correspondentes. Uma vez determinada uma sobreposição máxima, o seu custo é calculado. O custo de um par de nós folhas na sobreposição é definido por uma função de distância para *strings*. Para os nós não folha, o custo é definido como sendo zero. O custo total da sobreposição é a soma dos custos dos pares de nós e é igual à distância entre duas árvores XML. Uma limitação desta medida de similaridade é o facto do peso dos nós não terminais ser zero, o que faz com que o resultado final dependa essencialmente da similaridade dos nós folha. Esta situação não acontece na

medida de similaridade aqui apresentada, onde as probabilidades também são calculadas para os nós não terminais.

2.3 Modelos de Dados e Abordagens na Detecção de Duplicados

Nesta secção são apresentados diversos métodos utilizados na detecção de duplicados. Deste modo, é dada uma perspectiva ao leitor dos diferentes caminhos que podem ser tomados na resolução de um problema deste tipo, à medida que são ilustradas as exigências dos diferentes modelos de dados. Aqui são também expostos alguns dos métodos referenciados na Tabela 1, apresentando, paralelamente, o seu modelo de dados e a abordagem seguida na respectiva detecção.

2.3.1 Modelo de Dados em Tabela

O modelo em tabela é o mais comum na maioria dos estudos realizados até à data. Tal se deve ao facto deste modelo representar a estrutura de dados presente nas tradicionais bases de dados relacionais. No âmbito das bases de dados relacionais, as principais abordagens empregues são baseadas em aprendizagem e métodos iterativos.

Estratégias baseadas em Aprendizagem

Dois trabalhos representativos de estratégias baseadas em aprendizagem são os apresentados em [25] e [26].

Em [25] são desenvolvidas técnicas para melhorar funções de similaridade textual. É proposta a utilização de medidas de similaridade aprendidas para cada campo da base de dados e, deste modo, é demonstrado que estas medidas são capazes de se adaptar à similaridade pretendida para o domínio de cada campo. São apresentadas duas medidas para realizar esta tarefa de aprendizagem. Uma variante do método comum de similaridade entre *strings* (*edit distance*), com uma componente de aprendizagem, e uma medida baseada no modelo vectorial [40], que faz uso de *Support Vector Machines* [41] para o treino das medidas. Este algoritmo assenta em dois níveis de aprendizagem. Primeiro as medidas de similaridade são treinadas para todos os campos da base de dados e, em seguida, uma medida para detectar duplicados é aprendida a partir das medidas de similaridade empregues anteriormente em cada campo.

No trabalho desenvolvido em [26], os autores abordam um problema mais alargado da detecção de duplicados, no qual os objectos podem possuir atributos disjuntos, como acontece nos casos em que os dados provêm de tabelas relacionais com esquemas diferentes. Nesta solução, o algoritmo começa por comparar dois tuplos diferentes. Em seguida, aplica-lhes um conjunto de perfis que contêm informação sobre o modo como é constituído um dado objecto. Estes perfis indicam se o par de tuplos pode ser um duplicado. Os perfis utilizados podem ser definidos manualmente ou aprendidos através dos dados construídos a partir de uma fonte de dados externa. Este é um algoritmo que se foca na optimização da eficácia das comparações entre os objectos, baseando-se no pressuposto de que atributos disjuntos estão muitas vezes relacionados, e que essa ligação pode melhorar a performance das comparações realizadas entre eles.

Estratégias Iterativas

As soluções apresentadas em [6, 16, 32, 33] utilizam uma abordagem iterativa e procuram, através de vários passos, obter tanto eficiência como eficácia nos algoritmos.

O trabalho apresentado em [6] engloba duas alternativas para atingir um algoritmo o mais eficiente possível. As alternativas usadas são um método de ordenação de vizinhos, proposto em [44], e um método de agrupamento. Os autores do artigo demonstram uma forma de melhorar estes métodos, baseando-se numa abordagem que efectua várias passagens pelos dados. Este algoritmo opta por analisar uma pequena parte dos dados de cada vez, seguindo a máxima de que efectuar várias passagens baratas pelos dados resulta melhor do que efectuar poucas passagens caras.

Em [16] é criado um algoritmo para reconhecer grupos de registos aproximadamente duplicados. A solução apresentada é baseada em três pilares principais. O primeiro consiste num algoritmo de comparações que funciona relativamente bem para qualquer domínio de dados. O segundo consiste numa implementação eficiente do fecho transitivo, utilizando uma estrutura de dados *union-find*. O último baseia-se na utilização de heurísticas com o objectivo de minimizar o número de comparações dispendiosas.

Em [32], o algoritmo proposto captura propriedades estruturais dos dados para caracterizar grupos de tuplos duplicados.

No trabalho apresentado em [33], os autores sugerem uma solução realizada em dois passos. O seu algoritmo consiste em, numa primeira etapa, mapear registos para um espaço euclidiano multi-dimensional, utilizando para tal técnicas como as descritas em [43]. Na segunda etapa,

são encontrados pares de objectos similares, no espaço euclidiano, cuja distância se encontra dentro de um determinado valor (threshold).

2.3.2 Modelo de Dados em Árvore

No modelo em árvore os dados são representados obedecendo a uma estrutura hierárquica. Sendo assim, os trabalhos apresentados neste tópico aplicam-se normalmente a dois tipos diferentes de repositórios de dados: *data warehouses* e bases de dados XML.

Em [34], os autores descrevem um algoritmo para eliminar duplicados em tabelas dimensionais numa *data warehouse*. Estes exploram a sua hierarquia dimensional, propondo uma estratégia de agrupamento, que apenas compara tuplos dentro de pequenos grupos em cada relação. Esta medida tem por objectivo evitar comparar todos os pares de tuplos presentes em cada relação da hierarquia. Deste modo, os dados são processados iterativamente, em que primeiro são detectados os duplicados para cada relação e, posteriormente, é processado o seu agrupamento com outras relações, considerando os duplicados detectados. Este algoritmo faz ainda uso de uma medida de *threshold* dinâmica. A sua finalidade é atribuir *thresholds* diferentes a cada grupo e, deste modo, poder adaptá-los às características dos dados. Em conclusão, este é um algoritmo que pretende ser escalável, visto lidar com dados de grandes dimensões, e pretende alcançar eficácia e eficiência fazendo uso de uma abordagem iterativa para evitar comparações.

Os dois artigos seguintes [35] e [37] apresentam uma solução direccionada para o XML. Em ambos é oferecida uma solução focada na eficiência do algoritmo.

O algoritmo apresentado em [37] considera que existem quatro questões essenciais na detecção de duplicados em XML. Estas questões representam quatro tarefas essenciais a integrar no algoritmo e que são: selecção dos descritores, classificação de duplicados, estratégia de comparação e escalabilidade.

A primeira tarefa consiste em decidir qual a informação que descreve um elemento XML. Aqui, os autores consideram dois modos de obter descritores de objectos. O primeiro reside na aplicação de heurísticas baseadas na constatação de que os elementos que descrevem um objecto são, normalmente, definidos em proximidade do elemento que o representa. O segundo faz uso de métodos estatísticos, aplicados à estrutura e ao conteúdo dos elementos, com o objectivo de verificar aqueles que possuem maior frequência e, portanto, terão maior relevância em ser incluídos.

A segunda tarefa baseia-se na classificação que é atribuída a um par de objectos (duplicados ou não duplicados) segundo os seus descritores. Neste caso foram consideradas duas medidas para a classificação de duplicados. Uma que considera a heterogeneidade das instâncias e que utiliza uma classificação baseada num valor aproximado de *threshold*, recorrendo a uma medida de similaridade independente do domínio, como a proposta em [17], e outra que considera a heterogeneidade do esquema XML. Nesta última é utilizada uma medida de similaridade para calcular o custo de transformar o esquema de um objecto no de outro. Posteriormente, são pesados os dados e a heterogeneidade das instâncias, e é determinado se os objectos são ou não duplicados.

A terceira tarefa lida com a melhor forma de efectuar as comparações necessárias entre todos os pares de objectos. No trabalho em questão, são adoptadas duas estratégias para evitar um grande número de comparações, uma baseada na aplicação de filtros de *threshold* e outra baseada em heurísticas.

A última tarefa prende-se com a escalabilidade. Para grandes quantidades de informação existe a necessidade de minimizar o número de acessos ao disco e, portanto, o número de passagens pelos dados. Para resolver esta situação, os autores recorrem a estratégias que incluem a utilização de bases de dados relacionais para armazenar os descritores.

No trabalho desenvolvido em [35], é adaptado ao XML um algoritmo conhecido da detecção de duplicados em bases de dados relacionais, proposto em [44]. Numa fase inicial, é gerada uma chave para cada elemento da base de dados sujeito a comparações. Em seguida, numa etapa de detecção de duplicados, os elementos são ordenados a partir dessas chaves e são comparados apenas os objectos mais próximos. Esta medida tem o intuito de melhorar a eficiência do algoritmo, partindo do princípio que a ordenação da fase anterior coloca os duplicados perto uns dos outros. Para tal, utilizam uma medida de similaridade que considera duplicados entre os sucessores dos elementos e informação inserida manualmente para descrever os objectos (descritores).

2.3.3 Modelo de Dados com Grafo

Os autores dos trabalhos apresentados neste tópico baseiam-nos na construção de uma representação interna para os dados, sob a forma de um grafo.

Estratégias baseadas em Aprendizagem

Um exemplo de uma estratégia que se baseia em aprendizagem está presente em [27]. Neste trabalho, os autores propõem resolver o problema de detecção de duplicados recorrendo a algoritmos de aprendizagem e tendo como objectivo principal a eficácia do seu método. Nesta solução, é levada a cabo uma inferência realizada simultaneamente para todos os pares de objectos candidatos, permitindo a propagação de informação entre os mesmos. Essa propagação é efectuada através de atributos comuns entre os objectos. Os parâmetros utilizados neste modelo são aprendidos usando redes neuronais.

Estratégias baseadas em técnicas de Agrupamento

Os trabalhos propostos em [15], [31] e [7] empregam na sua abordagem métodos de agrupamento. A abordagem proposta em [15] vê conjuntos de dados como grafos em que os atributos que representam os objectos são vértices. No seu funcionamento, a abordagem aplica técnicas de particionamento de grafos para agrupar representações de objectos. Faz também uso de métodos que verificam a similaridade entre objectos, consoante o peso das suas relações. Esta solução, em contraste com outras técnicas tradicionais, analisa não só as características dos objectos, mas também as relações entre eles que se encontram implícitas na base de dados.

Em [31] é apresentada uma solução que visa resolver problemas correntes em bibliotecas digitais. Dada a grande dimensão das bibliotecas digitais em geral, a solução investigada pretende principalmente ser escalável.

No trabalho proposto em [7], os autores propõem duas medidas de similaridade diferentes. Uma que actua ao nível dos atributos (devolve um valor entre 0 e 1, consoante o seu grau de similaridade) e outra que considera uma similaridade entre relações. Nesta segunda, é medida a similaridade entre os *clusters* de duas entidades, considerando os *clusters* com que estas estão relacionadas, através das arestas do grafo.

Estratégias Iterativas

Nos artigos [8] e [45] é adoptada uma abordagem iterativa que divide a resolução do problema em diferentes etapas. O algoritmo descrito em [8] é referido pelos autores como sendo eficaz em espaços de informação complexos e em casos onde objectos possuam informação em falta. Este possui três características principais: explora associações entre referências para desenhar novos métodos de comparação, propaga a informação entre as decisões de

duplicidade e enriquece gradualmente os objectos juntando valores de atributos. O algoritmo faz uso de um grafo de dependências para a propagação de decisões. Aqui, os nós representam similaridades entre pares de objectos e as arestas representam dependências entre decisões de duplicidade. Este estudo teve como objectivo final a eficácia do seu algoritmo.

No trabalho desenvolvido em [45] é apresentada uma solução para a detecção de duplicados em dados XML. Este artigo foca-se na exploração eficiente das dependências entre objectos. No algoritmo apresentado, os pares de objectos, em qualquer nível da hierarquia, são comparados segundo uma ordem que depende das suas relações. Objectos com muitas dependências influenciam muitas decisões de duplicidade e, portanto, deve ser decidido *à priori* se eles próprios são duplicados. Para fazerem uso desta premissa, os autores criaram dois algoritmos de modo a obter uma estratégia de ordenação das comparações e, assim, poderem alcançar um compromisso entre eficácia e eficiência. Esta ordenação torna-se mais relevante quando há uma grande interdependência entre os elementos. Os algoritmos desenvolvidos denominam-se *Recona* e *Adama*. O primeiro permite que um objecto seja reexaminado se os vizinhos que o influenciam forem duplicados. Deste modo, a ordenação obtida evita recomparações entre objectos. O segundo não permite recomparações, pois cada par de objectos apenas é comparado uma vez. Isto torna-o mais eficiente, sendo que a ordenação, neste caso, minimiza o número de erros. Adicionalmente, existem duas extensões a estes algoritmos. Uma delas permite que um par de objectos seja classificado como duplicado ou não duplicado, numa fase inicial, sem que seja feita uma computação dispendiosa da sua similaridade. Isto baixa significativamente o custo computacional das comparações efectuadas entre todos os pares. A outra evita comparações entre elementos que não partilhem de um elemento em comum. Os cortes resultantes desta medida podem, na prática, aumentar significativamente a eficácia do algoritmo. O grafo utilizado para contemplar os algoritmos e comparações entre objectos serve-se de três componentes, nomeadamente, vértices de elementos, vértices de texto e arestas de dependências. Assim, é criado um vértice elemento para cada candidato, um vértice texto para cada nó de texto num descritor e as arestas são criadas para representar as dependências entre os vértices. Através de cortes e ordenações, este trabalho pretende alcançar uma boa eficiência na execução do seu algoritmo.

3 Conceitos Gerais

Nesta secção pretende-se transmitir ao leitor uma base de conhecimento que melhor lhe permita compreender os conceitos mencionados ao longo do documento. Na secção 3.1 são apresentadas características do modelo XML. O modo como a detecção de duplicados é abordada e adaptada ao XML está presente na Secção 3.2. Por fim, na Secção 3.3, é descrito em que consiste o modelo teórico das redes Bayesianas e quais as vantagens em empregá-lo na detecção de duplicados em XML.

3.1 XML

O XML (Extensible Markup Language) [46] é uma proposta para uma linguagem de *markup*, semelhante à linguagem HTML [47], que permite aos utilizadores definirem as suas próprias etiquetas. O seu principal objectivo é facilitar a partilha de dados entre diferentes sistemas de informação. Na Figura 2 está representado um exemplo do formato de um ficheiro XML.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<cddb>
  <disc>
    <did>a012320c</did>
    <artist>Various</artist>
    <dtitle>Frankfurt Trance Vol. 04 cd1</dtitle>
    <category>data</category>
    <tracks>
      <title>DJ Tom Stevens VS. Fridge - Outface 2000 (Radio Mix)</title>
      <title>Alice Deejay - Better Off Alone (Signum Remix)</title>
      <title>Tillmann Uhrmacher Feat. Peter Ries - Bassfly (Original Mix)</title>
      <title>DJ 2 L 8 - Too Late</title>
      <title>Time Square - Invisible Girl (Future Breeze Remix)</title>
      <title>Cirillo - Across The Soundline</title>
      <title>DJ Leon &amp;amp; Jam X - Hold It</title>
      <title>Sean Dexter - Synthetica (Extended Mix)</title>
      <title>DJ Bjorn - On A Mission (Original Mix)</title>
      <title>8Voice - Music Hypnotizes 2000</title>
      <title>Alex Apollo - Jahr 2000</title>
      <title>Headroom - Utopia (Radio Mix)</title>
    </tracks>
  </disc>
  <disc>
    <did>a1088c0b</did>
    <artist>NO RETURN</artist>
    <dtitle>Self Mutilation</dtitle>
    <category>data</category>
    <genre>Data</genre>
    <cdextra> ID3G: 254</cdextra>
    <tracks>
      <title>Do or Die</title>
      <title>Truth and Reality</title>
      <title>Lost</title>
      <title>Soul Extractor</title>
      <title>Sadistic Desire</title>
      <title>The True Way</title>
      <title>Fanatic Mind</title>
      <title>Individualistic Ideal</title>
      <title>One Life</title>
      <title>Trail of Blood</title>
      <title>Sect</title>
    </tracks>
  </disc>
</cddb>
```

Figura 2: Exemplo de um ficheiro XML.

Nos dias que correm, o XML assume um papel fundamental na distribuição de dados e é, cada vez mais, um suporte *standard* para a sua divulgação. A popularidade que alcançou deve-se em grande parte ao crescimento da Internet, onde manifestou ser de grande utilidade para a publicação de dados, e à enorme simplicidade na sua utilização como veículo de troca de informação entre empresas. Face a esta evolução, constatou-se que se tornava cada vez mais importante criar ferramentas eficazes para a detecção de dados duplicados em XML, de modo semelhante ao que já existe em outros modelos de dados. Tal tarefa não se revelou tão simples de realizar como em outros modelos, principalmente devido à complexidade da sua estrutura. O XML possui uma organização hierárquica livremente estruturada, que inclui campos opcionais e listas de elementos que possuem dependências entre si. A sua construção pode ser efectuada com um certo grau de liberdade, o que propicia uma não uniformização da estrutura dos objectos e um consequente crescimento na complexidade do problema da detecção de duplicados.

Várias soluções estudadas até à data abordam o problema focando-se em três aspectos essenciais, que advêm da maior complexidade do XML. O primeiro diz respeito à forma de representar um objecto do mundo real de modo a que este possa ser comparado com outros. Isto significa que é necessário decidir que valores presentes nos dados realmente descrevem o objecto, ou seja, que valores considerar aquando da comparação de dois objectos. Este aspecto assume, correntemente, a designação de *object description.* No modelo relacional torna-se muito fácil fazer a correspondência entre o objecto do mundo real e o seu *object descriptor*, visto que este pode ser formado simplesmente pelos seus atributos. No caso do XML a questão torna-se menos óbvia. O objecto é representado por elementos estruturados e dependentes entre si, o que torna difícil fazer uma correspondência linear entre este e o seu descritor.

O segundo aspecto está relacionado com a própria construção que o modelo XML permite edificar. Devido à sua livre organização, este possibilita que dois objectos com diferenças estruturais representem o mesmo objecto no mundo real. Autores, como em [48], dividem ainda este aspecto da chamada *heterogeneidade estrutural* em dois sub-aspectos: *heterogeneidade esquemática* e *heterogeneidade de instâncias*. A primeira acontece quando dois elementos diferentes representam o mesmo objecto do mundo real. Isto é, se tivermos dois elementos chamados "*movie"* e "*film"*, estes representam o mesmo objecto no mundo real e, no entanto, são elementos diferentes no esquema XML. A segunda significa que duas instâncias pertencentes ao mesmo esquema podem apresentar uma representação diferente. Isto significa que estes podem possuir, por exemplo, um número diferente de ocorrências de um elemento. Esta situação está representada na Figura 3.

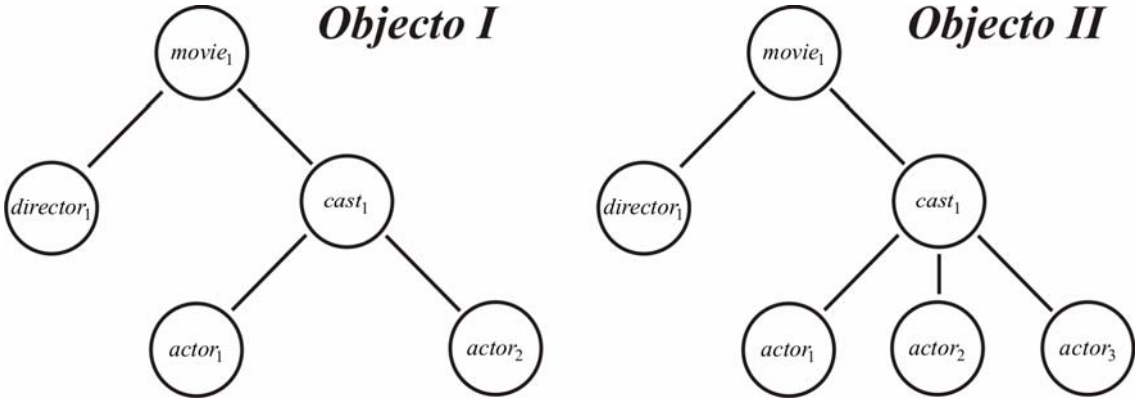


Figura 3: Objectos com diferente número de ocorrências de um elemento.

Por fim, o ultimo aspecto visa considerar as dependências que os elementos possuem. Em XML, os elementos assumem o papel de pais e filhos, relacionados entre si sob a forma de antecessores e predecessores. Trabalho recente, como [8], mostra que a eficácia e eficiência na detecção de duplicados é melhorada substancialmente se estas relações entre entidades forem consideradas.

3.2 Detecção de Duplicados em XML

Uma vez decidida qual a informação que vai representar determinado objecto do mundo real, não basta compará-la directamente com a dos restantes objectos, com o objectivo de determinar quais serão considerados duplicados. Toda a informação contida dentro de um documento XML se encontra sob a forma de *strings*. Logo, comparar directamente o conteúdo de todos os campos dos objectos, com o propósito de localizar replicações, seria muito ineficiente. Além disso, um dos problemas mais comuns nas bases de dados é conterem representações erróneas da informação, devido, por exemplo, a erros tipográficos. Isto significa que a comparação entre dois campos que contenham, por exemplo, a informação “João Silva” e “Joao Silva” resulta numa desigualdade, contrariando a grande probabilidade, que os elementos possuem, de representar a mesma pessoa. A Figura 4 ilustra um exemplo deste problema. Por esta razão, a comparação entre elementos XML faz normalmente uso de medidas de similaridade entre *strings,* problema conhecido como *string matching* [49]. Tais métodos consistem, normalmente, em medir a quantidade mínima de inserções, eliminações e substituições necessárias para converter uma das *strings* na outra de modo a ficarem iguais, e podem ainda ser adaptados ao domínio dos dados em questão. Outros algoritmos mais complexos, como o proposto em [42], exploram medidas de aprendizagem para semelhança de *strings*.

```
<film>
    <title> Big </title>
    <actor>Hanks</actor>

    <actor>Perkins</actor>
    <runtime>
       104
    </runtime>
</film>
```

```
<film>
    <title> Big </title>
    <actor>Hansk</actor>
    <actor>Loggia</actor>
    <actor>Perkins</actor>
    <runtime>
       1044
    </runtime>
</film>
```

Figura 4: Objectos que representam a mesma entidade com dados erróneos, ou em falta.

Para comparar a totalidade dos objectos, é necessário aplicar os métodos descritos anteriormente aos elementos contidos nas respectivas árvores XML. Vários são os critérios utilizados pelas diferentes abordagens existentes. Os mais comuns consideram que dois deles são iguais se os pais forem iguais ou similares, se possuírem o mesmo nome na sua etiqueta, se os dados que contêm forem similares, ou se tiverem dados similares e possuírem estrutura semelhante. Considerados todos estes critérios, entre dois objectos, é comum ser obtido um valor que é representativo de um grau de similaridade que lhes é atribuído. Os algoritmos possuem então, nestes casos, um valor fronteira (*threshold)* que é utilizado para decidir que pares de objectos são realmente considerados duplicados. Esta apreciação é efectuada consoante este valor esteja acima ou abaixo de um valor previamente definido.

Naturalmente, grande parte da despesa computacional efectuada pelos algoritmos é localizada no período de comparação entre todos os pares de objectos pertencentes à base de dados. Contudo, o facto de muitas bases de dados conterem um elevado número de objectos não é a única razão para esta ocorrência. Uma grande parte do processamento efectuado é dispendido nas, anteriormente mencionadas, comparações entre *strings.* Os métodos de *string matching* são computacionalmente bastante caros, o que faz com que os autores de diversas soluções procurem ao máximo evitar a evocação dos mesmos. Se considerarmos uma base de dados que contém 3.000 objectos, o algoritmo utilizado para a detecção de duplicados vai efectuar, pelo menos, 4.498.500 comparações entre objectos (número de pares existentes). Se considerarmos também que cada objecto dessa base de dados possui trinta campos diferentes que têm de ser comparados, facilmente podemos constatar a pertinência de tal preocupação por parte de quem constrói os algoritmos. Por este motivo, a maioria das abordagens aposta na utilização de filtros e de heurísticas, com o objectivo de reduzir o número de cálculos efectuados. Adicionalmente, algumas abordagens, que efectuam mais do que uma comparação entre o mesmo par de objectos, optam por estratégias focadas na ordem pela qual estas são realizadas, de forma a evitar comparações desnecessárias.

3.3 Redes Bayesianas

As redes Bayesianas fornecem um formalismo gráfico para representar, explicitamente, as dependências entre as variáveis de um determinado domínio, fornecendo ao mesmo tempo uma clara especificação de uma distribuição conjunta de probabilidades [50]. Esta representação é baseada num grafo acíclico dirigido, onde um conjunto de variáveis aleatórias compõe os nós da rede e um conjunto de ligações direccionadas ligam pares de nós. Neste grafo, uma aresta entre dois nós significa que o primeiro possui uma influência directa no segundo. Esta influência é quantificada através de uma função de distribuição de probabilidade condicional, que relaciona os estados de cada nó com os estados dos seus pais.

Para ilustrar, vamos assumir que *X* e *Y* são duas variáveis aleatórias, como esquematizado na Figura 5, e *x* e *y* os seus respectivos valores. Vamos usar *X* e *Y* para nos referirmos às variáveis aleatórias, bem como aos nós da rede que estão associados a estas variáveis. Uma aresta dirigida de *Y*, o nó pai, para *X*, o nó filho, representa a influência da variável *Y* sobre a variável *X*, o que é quantificado pela probabilidade condicional $P(x \mid y)$.

Figura 5: Variáveis aleatórias *X* e *Y*.

Generalizando, seja Π o conjunto de todos os nós pais de um determinado nó *X*. Adicionalmente, consideremos que p é o conjunto de valores para todas as variáveis em Π, e que *x* é o valor da variável *X*. A influência de Π em *X* pode ser modelada por qualquer função *F* tal que $\sum_x F(x, p) = 1$ e $0 \le F(x, p) \le 1$. A função $F(x, p)$ fornece um quantificador numérico para $P(x \mid p)$.

A Figura 6 ilustra uma rede Bayesiana para a distribuição de uma probabilidade conjunta $P(x_1, x_2, x_3, x_4, x_5)$. Neste caso, as dependências declaradas na rede permitem a

representação da distribuição da probabilidade conjunta em termos de probabilidades condicionais locais, tal como apresentado em seguida.

$$P(x_1, x_2, x_3, x_4, x_5) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1)P(x_4 \mid x_2, x_3)P(x_5 \mid x_3)$$

A probabilidade $P(x_1)$ é designada de probabilidade *à priori* e pode ser usada para modelar conhecimento anterior.

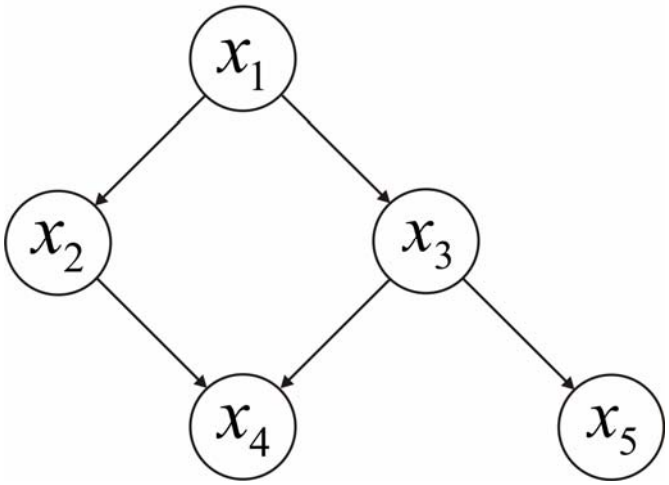


Figura 6: Exemplo de uma rede Bayesiana.

A principal vantagem das redes Bayesianas é o modo sintetizado com que representam as relações probabilísticas. De facto, apenas é necessário considerar as independências conhecidas entre as variáveis de um domínio, em lugar de especificar uma distribuição de uma probabilidade conjunta completa. As dependências declaradas na altura da modelação são usadas para inferir conhecimento para todas as variáveis na rede. O mecanismo de inferência, apesar de exponencial no pior caso, é eficiente na maioria das soluções práticas, particularmente nas que são apresentadas neste trabalho.

4 Detecção de Duplicados usando Redes Bayesianas

Para a detecção de duplicados em XML, é construída uma rede Bayesiana tal como ilustrado na Figura 7. Consideremos em primeiro lugar os nós XML com a etiqueta *mv* nas árvores *U* e *U'* da Figura 1. A rede Bayesiana terá um nó designado mv_{11} que representa a possibilidade do nó mv_1 na árvore XML *U* ser um duplicado do nó mv_1 na árvore XML *U'*. Ao nó mv_{11} é atribuída uma variável binária aleatória. Esta variável assume o valor 1 (activo) para representar o facto dos nós *mv* nas árvores *U* e *U'* serem duplicados.

A probabilidade dos dois nós XML serem duplicados depende de (i) os seus valores serem ou não duplicados, e (ii) os seus filhos serem ou não duplicados, de acordo com o pressuposto descrito na Secção 1.3. Assim, o nó mv_{11} na rede Bayesiana possui dois nós pais: Vmv_{11} e Cmv_{11}. O nó Vmv_{11} representa a possibilidade dos valores nos nós *mv* serem duplicados. Tal como anteriormente, a estes nós é atribuída uma variável binária aleatória que pode estar activa ou inactiva, representando a situação dos valores e os nós filhos serem duplicados ou não-duplicados, respectivamente.

É assumido que a probabilidade dos valores dos nós XML serem duplicados depende de cada atributo, independentemente. Isto é representado na rede através da adição de novos nós para os atributos, como pais do nó Vmv_{11}, representados por rectângulos na Figura 7. Neste caso, os novos nós representam a possibilidade dos valores de *year* nos nós *mv* serem duplicados e dos valores de *title* nos nós *mv* serem duplicados. Analogamente, a probabilidade dos filhos dos nós *mv* serem duplicados depende da probabilidade de cada par de nós filhos ser duplicado. Deste modo, são adicionados mais dois nós como pais do nó Cmv_{11}: o nó dr_{11} representa a possibilidade do nó dr_1 na árvore *U* ser duplicado do nó dr_1 na árvore *U'*; o nó cst_{11} representa a possibilidade do nó cst_1 na árvore *U* ser duplicado do nó cst_1 na árvore *U'*.

Podemos agora repetir todo o processo para estes dois nós. Contudo, é adoptado um procedimento ligeiramente diferente quando se representam múltiplos nós do mesmo tipo, como é o caso dos nós com etiqueta *ac*. Neste caso, pretende-se comparar o conjunto total de nós, em lugar de cada nó isoladamente. Assim, diz-se que o facto do conjunto de nós *ac* ser duplicado depende de cada nó *ac* na árvore *U* ser duplicado de qualquer nó *ac* na árvore *U'*. Isto está representado pelos nós ac_{**}, ac_{1*} e ac_{2*}, na rede Bayesiana da Figura 7. Finalmente, cada nó ac_{ij} representa a possibilidade do nó ac_i na árvore *U* ser duplicado do nó ac_j na árvore *U'*. Como os nós *ac* não possuem filhos, a probabilidade de serem duplicados depende apenas dos seus valores. Deste modo, cada nó ac_{ij} na rede tem apenas um nó pai Vac_{ij}. Sendo que cada nó *ac* possui apenas um valor, cada nó Vac_{ij} na rede tem apenas um nó pai que representa a possibilidade de ambos os nós XML, ac_i e ac_j, terem valores duplicados.

Um algoritmo formalizando o modo como a rede Bayesiana é construída está presente na Secção 4.2.

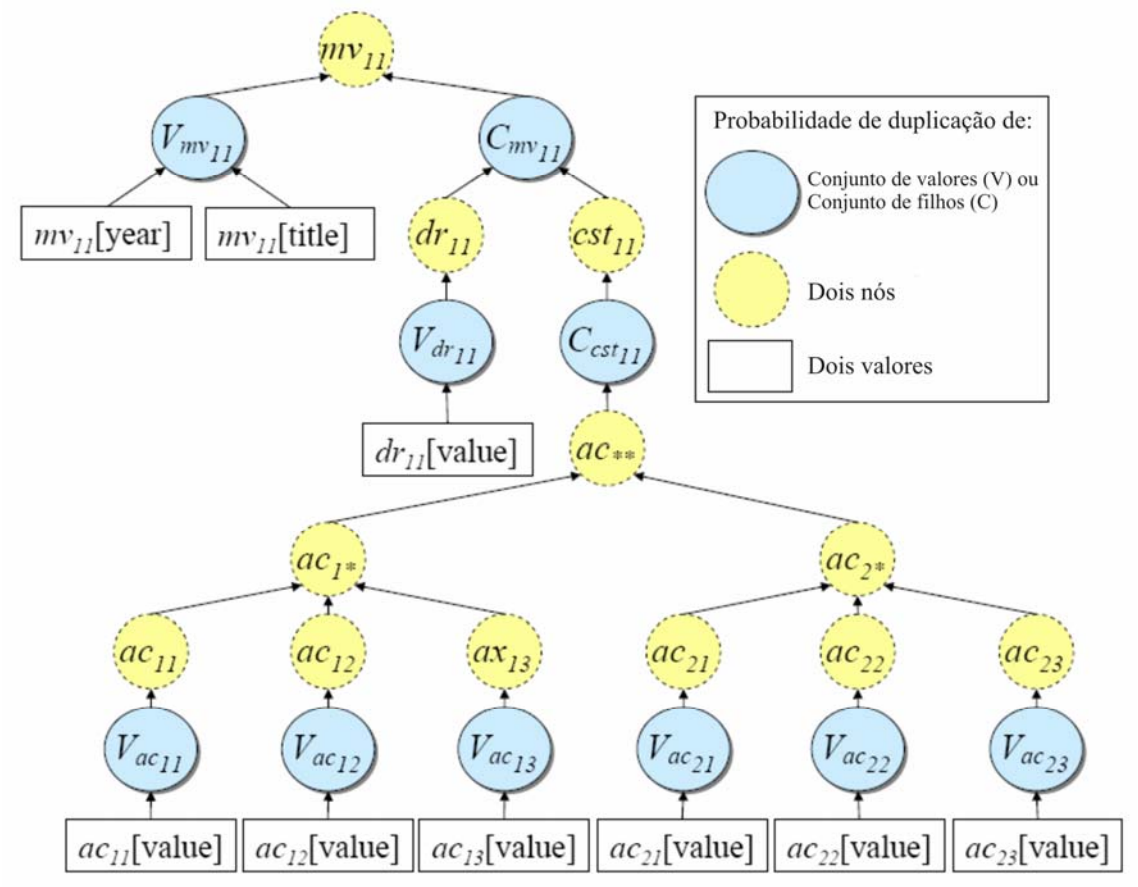


Figura 7: Rede Bayesiana para calcular a similaridade das duas árvores da Figura 1.

Através da rede da Figura 7, podemos agora calcular a probabilidade $P(mv_{11})$ dos objectos *U* e *U'* serem duplicados. De facto, podemos calcular a probabilidade de qualquer par de nós que seja comparável, um de cada árvore, ser um duplicado. Por exemplo, se soubermos que os nós ac_2 (da árvore *U*) e ac_3 (da árvore *U'*) são duplicados, podemos calcular a probabilidade das árvores serem duplicados, $P(mv_{11}|ac_{23})$. Por outro lado, podemos calcular a probabilidade do elenco nos dois filmes ser duplicado se os realizadores forem duplicados, ou seja, $P(cst_{11}|dr_{11})$. O foco deste trabalho é a probabilidade dos nós da raiz serem duplicados $P(mv_{11})$, o que pode ser interpretado como um valor de similaridade entre os dois objectos XML. Para calcular esta probabilidade, é necessário definir, em primeiro lugar, as probabilidades *à priori* associadas aos nós terminais (folhas) da rede e as probabilidades condicionais associadas aos nós internos da rede.

Na rede da Figura 7, é necessário definir as probabilidades *à priori* dos valores serem duplicados no contexto dos seus nós pais, por exemplo, $P(mv_{11}[year])$,

$P(mv_{11}[title])$, $P(dr_{11}[value])$, e $P(ac_{ij}[value])$. Necessitamos ainda de definir quatro tipos de probabilidades condicionais: (i) a probabilidade dos valores dos nós serem duplicados, dado que cada par de valores, individualmente, é duplicado; (ii) a probabilidade dos filhos dos nós serem duplicados, dado que cada par de filhos, individualmente, é duplicado; (iii) a probabilidade de dois nós serem duplicados, dado que os seus valores e os seus filhos são duplicados; e (iv) a probabilidade de um conjunto de nós do mesmo tipo ser duplicado, dado que cada par de nós do conjunto, individualmente, é duplicado. Neste exemplo, os quatro tipos de probabilidades condicionais (denominadas PC1 a PC4) correspondem às probabilidades descritas nas Tabelas 2 (a) a (d).

Probabilidade Condicional 1
$P(Vmv_{11} \mid mv_{11}[year], mv_{11}[title])$
$P(Vdr_{11} \mid dr_{11}[value])$
$P(Vac_{ij} \mid ac_{ij}[value])$

(a) Probabilidade Condicional PC1

Probabilidade Condicional 2
$P(Cmv_{11} \mid dr_{11}, cst_{11})$
$P(Ccst_{11} \mid ac_{**})$

(b) Probabilidade Condicional PC2

Probabilidade Condicional 3
$P(mv_{11} \mid Vmv_{11}, Cmv_{11})$
$P(dr_{11} \mid Vdr_{11})$
$P(cst_{11} \mid Ccst_{11})$
$P(ac_{ij} \mid Vac_{ij})$

(c) Probabilidade Condicional PC3

Probabilidade Condicional 4
$P(ac_{**} \mid ac_{1*}, ac_{2*})$
$P(ac_{i*} \mid ac_{i1}, ac_{i2}, ac_{i3})$

(d) Probabilidade Condicional PC4

Tabela 2: Probabilidades Condicionais.

4.1 Motivação da Abordagem Proposta

Tal como descrito na Secção 3, as redes Bayesianas possuem a capacidade de representar as dependências de um conjunto de variáveis num determinado domínio. Sendo o XML um suporte de dados, onde estes se encontram modelados de forma hierárquica, existe um claro interesse em relacionar os vários elementos dos diversos níveis dessa estrutura, com vista a identificar se dois objectos são duplicados.

Uma abordagem que poderia ser seguida para verificar se dois objectos representam, de facto, um duplicado consiste em extrair todos os nós da árvore XML que têm um valor associado, agrupá-los segundo o seu tipo e, em seguida, compará-los. Esta abordagem significa que todos os nós são encarados como estando no mesmo nível, visto que todos são extraídos e comparados paralelamente, ou seja, a estrutura hierárquica dos elementos não é tida em conta. Relativamente ao exemplo apresentado na Figura 1, seriam comparados actores e título do filme da mesma forma, sem que fosse distinguida a importância que cada um assume na identificação do objecto. Esta situação não está de acordo com o propósito do XML nem com o que se passa na prática no mundo real. Se considerarmos o que ocorre no mundo real, podemos facilmente constatar que temos maior facilidade em dizer se dois filmes são iguais se olharmos para o seu título ao invés de examinarmos os seus actores. Isto advém do simples facto de um actor protagonizar vários filmes e, portanto, poder estar presente em vários filmes diferentes. Por outro lado, o nome do filme é, na grande maioria dos casos, um identificador da sua unicidade. O XML possui a vantagem de poder capturar estas situações ao permitir colocar diferentes elementos em diferentes níveis da sua árvore, como está representado na Figura 1, sendo que, tipicamente, os de maior importância e que melhor caracterizam o objecto estão presentes nos níveis mais perto da raiz.

A utilização de uma rede Bayesiana como a da Figura 7, possui a vantagem de considerar os diferentes níveis de importância entre os elementos da árvore XML e respeitar a sua estrutura hierárquica. O efeito de propagação de probabilidades que ocorre na rede garante que a probabilidade de duplicação entre os nós de níveis inferiores vai afectar directamente a probabilidade de duplicação de um nó do nível imediatamente superior, e assim sucessivamente até chegar à raiz. Nessa altura é garantido que apenas os nós do nível abaixo da raiz influenciam directamente a probabilidade do objecto ser duplicado. Ou seja, como representado na Figura 7, o nó dr_{11} possui maior peso no cálculo de $P(mv_{11})$ do que, por exemplo, o nó ac_{23} que não propaga a sua probabilidade directamente para o nó raiz. O nó que pertence à sub-rede que contém ac_{23} e que propaga directamente a sua probabilidade para o nó mv_{11} é o nó cst_{11}, e a probabilidade de ser duplicado depende de ac_{23}, assim como dos restantes nós que representam comparações entre actores.

A rede Bayesiana possui ainda outra vantagem na detecção de duplicados XML. Esta possui a versatilidade de se poderem aplicar diferentes funções nos nós intermédios da rede e na sua raiz. Estas funções recebem como *input* as probabilidades que são propagadas dos nós inferiores e produzem como *output* uma probabilidade atribuída ao nó superior segundo uma determinada lógica de ponderação.

4.2 Construção da rede

Formalmente, uma árvore XML é definida como um triplo $U = (t, V, C)$, onde

- t é a etiqueta da raiz. Por exemplo, para a árvore U na Figura 1, $t = mv$.

- V é um conjunto de pares (atributo, valor). Se o próprio nó tem um valor, pode ser considerado como um par especial (atributo, valor). Para a árvore U, na Figura 1, tem-se $V = \{$(*year,* '1983'*), (title,* 'prosandcons'*)*$\}$.

- C é um conjunto de árvores XML, isto é, as sub-árvores de U. Para a árvore U, na Figura 1, C contém sub-árvores com raíz em *dr* e *cst*. Estas sub-árvores são, por sua vez, descritas individualmente por um triplo.

Diz-se que duas árvores XML são duplicadas se os seus nós da raiz são duplicados.

Para todos os nós com uma dada etiqueta, pode associar-se um tipo do mundo real. Vamos definir T_t como o tipo do mundo real associado aos nós com etiqueta t. Definimos ainda o tipo de uma árvore XML com o tipo do seu nó raiz. É assumido que duas árvores só podem ser duplicados se forem do mesmo tipo. Adicionalmente, dois nós apenas podem ser comparados se pertencerem ao mesmo tipo. No exemplo aqui presente, os tipos do mundo real são T_{mv} = *movie*, T_{dr} = *director*, T_{cst} = *cast* e T_{ac} = *actor*. Para simplificar, nas definições subsequentes vamos assumir que os nós com o mesmo tipo do mundo real também possuem a mesma etiqueta, ou seja, um passo de reformulação das etiquetas precedeu a construção da rede Bayesiana.

No Algoritmo 1, é fornecido pseudo-código para a construção da rede Bayesiana a partir da estrutura das árvores XML a serem comparadas. Intuitivamente, o algoritmo recebe como *input* dois conjuntos de árvores XML U e U', dos quais os nomes das etiquetas dos nós de raiz, pertencentes a dado tipo do mundo real r, são extraídos (linhas 1-4). Quando os nós de um tipo do mundo real r são processados, é distinguido entre os tipos dos nós que ocorrem no máximo uma vez (l.6) e o tipos dos nós com múltiplas ocorrências (l.26). No primeiro caso, apenas é

necessário construir um único nó V_{ii} e um único nó C_{ii}, ao passo que, no segundo caso, é necessário construir um nó t_{ij} para todas as combinações de nós t_i e t_j do mesmo tipo, bem como nós t_{**}, t_{i*}. Quando esta função é aplicada às árvores XML da Figura 1, obtém-se o grafo da Figura 7.

Entrada: $U = \{(t_1,V_1,C_1),(t_2,V_2,C_2),\ldots\}$,
$U' = \{(t'_1,V'_1,C'_1),(t'_2,V'_2,C'_2),\ldots\}$
Saída: Um grafo dirigido $G = (N,E)$
/* ---------- Inicialização ---------- */
/* Etiquetas dos nós raiz de todas as árvores XML em U e U' */
1 $S \leftarrow \{t_1, t_2, \ldots\}$;
2 $S' \leftarrow \{t'_1, t'_2, \ldots\}$;
/* Etiquetas em S e S' que representam o tipo r do mundo real */
3 $S_r = \{t_i \in S | T_{ti} = r\}$;
4 $S'_r = \{t'_i \in S | T_{t'i} = r\}$;
/* ---------- Construção da Rede Bayesiana ---------- */
5 **para cada** *tipo* $r \in S \cup S'$ **fazer**
/* Nós com uma única ocorrência */
6 **se** $|S_r| \leq 1$ *e* $|S'_r| \leq 1$ **então**
7 Inserir em N um nó t_{ii};
8 **se** $V_i \cup V'_i \neq \phi$ **então**
9 Inserir em N um nó $V_{t_{ii}}$;
10 Inserir em E uma aresta deste nó para o nó t_{ii};
11 **se** $C_i \cup C'_i \neq \phi$ **então**
12 Inserir em N um nó $C_{t_{ii}}$;
13 Inserir em E uma aresta deste nó para o nó t_{ii};
14 **se** *nó* $V_{t_{ii}}$ *foi criado* **então**
15 **para cada** *atributo* $a \in V_i \cup V'_i$ **fazer**
16 Criar um nó $t_{ii}[a]$;
17 Inserir uma aresta a partir deste nó para o nó $V_{t_{ii}}$;
18 **se** *nó* $C_{t_{ii}}$ *foi criado* **então**
19 $G' = (N',E') \leftarrow RBGen(C_i, C'_i)$;
20 **para cada** *nó* $n \in N'$ **fazer**
21 Inserir n em N;
22 **para cada** *aresta* $e \in E'$ **fazer**
23 Inserir e em E;
24 **para cada** *nó* $n \in N'$ *sem arestas de saída* **fazer**
25 Inserir uma aresta em E de n para o nó $C_{t_{ii}}$;
/* Nós com múltiplas ocorrências */
26 **caso contrário se** S_r *ou* S'_r *contêm mais que uma etiqueta cada* **então**
27 Inserir em N um nó t_{**};
28 **para cada** *etiqueta* $t_i \in S_r$ **fazer**
29 Inserir em N um nó t_{i*};
30 Inserir em E uma aresta a partir deste nó para o nó t_{**};
31 **para cada** *etiqueta* $t'_j \in S'_r$ **fazer**
32 Inserir em N um nó t_{ij};
33 Inserir em E uma aresta a partir deste nó para o nó t_{i*};
34 **para cada** *novo nó criado* t_{ij} **fazer**
35 Análogo ao processamento do nó t_{ii} (linhas 8-25), sendo que o segundo índice i é substituido por j ...

Algoritmo 1: *RBGen(XTreeSet U, XTreeSet U')*.

4.3 Definição das Probabilidades

Para completar a rede Bayesiana, construída como descrito na secção anterior, é necessário definir as probabilidades *à priori* associadas aos nós folhas da rede e as probabilidades condicionais associadas aos nós internos da rede, tal como ilustrado na Secção 4. Em ambos os casos, o modelo descrito assume uma abordagem epistemológica do problema, onde as probabilidades são interpretadas como graus de crença que podem ser especificados independentemente da sua experimentação.

Nas próximas secções, para simplificar, é usada a notação $P(x)$ para referir $P(x=1)$. É assumido que todas as probabilidades $P(x=0)$ são definidas como $P(x=0)=1-P(x=1)$.

4.3.1 Probabilidades *à priori*

No modelo aqui apresentado, as probabilidades *à priori* representam a probabilidade de dois valores na árvore XML serem, na verdade, o mesmo. Elas estão associadas aos nós etiquetados como $t_{ij}[a]$, onde *a* é o nome de um atributo. Por exemplo, no caso da rede da Figura 7, é necessário definir a probabilidade *à priori* $P(mv_{11}[year])$ dos dois anos nos nós *mv* serem o mesmo. Estas probabilidades podem ser definidas com base na similaridade entre os valores. Por exemplo, a probabilidade dos atributos *title*, em dois filmes, ser a mesma, pode corresponder a uma similaridade de *strings* entre os dois títulos. Se esta similaridade for normalizada para um valor entre 0 e 1, podemos aplicá-la como um valor de probabilidade. Contudo, por vezes não é possível, ou eficiente, medir a similaridade entre os valores de dois atributos. Neste caso, a probabilidade é definida como uma pequena constante, representando a possibilidade de quaisquer dois valores serem duplicados, antes de terem sido observados. Assim, é definido

$$P(t_{ij}[a]) = \begin{cases} sim(V_i[a], V_j[a]) & \text{se a similaridade foi medida} \\ k_a & \text{caso contrário} \end{cases} \qquad (1)$$

onde $V_i[a]$ é o valor do atributo *a* do *i*-ésimo nó com etiqueta *t* na árvore XML, *sim*(.) é uma função de similaridade, normalizada para estar compreendida entre 0 e 1, e k_a é uma pequena constante que representa a probabilidade de dois valores do atributo *a* serem similares. A constante k_a é denominada *probabilidade por omissão*. Por exemplo, para o atributo *year* nos nós *mv*, pode ser definido $sim(y_1, y_2) = 1$ se $y_1 = y_2$, e $sim(y_1, y_2) = 0$ no caso contrário. A probabilidade por omissão k_{year} pode ser derivada da distribuição dos anos na base de dados, ou simplesmente ser definida como um número pequeno.

4.3.2 Probabilidades Condicionais

Tal como discutido na Secção 4, neste modelo estão apresentados quatro tipos de probabilidades condicionais (PC1 a PC4). Nesta secção é apresentado o modo como é calculado cada tipo de probabilidade condicional.

Probabilidade Condicional PC1

PC1 traduz a probabilidade dos valores dos nós XML serem duplicados, dado que os valores dos seus atributos são duplicados. Formalmente, isto corresponde a $P\left(V_{t_{ij}} \middle| t_{ij}[a_1], t_{ij}[a_2], \ldots\right)$. Esta probabilidade pode ser definida arbitrariamente, desde que esteja em conformidade com os axiomas das probabilidades.

Neste trabalho, a proposta para definir a probabilidade é a seguinte: (a) se todos os valores dos atributos são duplicados, os valores dos nós são considerados duplicados; (b) se nenhum dos valores dos atributos for duplicado, os valores dos nós são considerados não duplicados; (c) se alguns dos valores dos atributos são duplicados, é determinado que a probabilidade dos nós serem duplicados é igual a um dado valor w_a. Este valor representa a importância do atributo respectivo em determinar se os nós são duplicados. Esta definição está representada na Equação (2).

$$P\left(V_{t_{ij}} \middle| t_{ij}[a_1], t_{ij}[a_2], \ldots, t_{ij}[a_n]\right) = \sum_{1 \le k \le n \mid t_{ij}[a_k]=1} w_{a\,k} \qquad (2)$$

$$\text{Em que } \sum_{1 \le k \le n} w_{ak} = 1, w_{ak} > 0 .$$

Por exemplo, na rede da Figura 7, podia usar-se w_{year} = 0.1 e w_{title} = 0.9, o que significa que se os anos forem iguais, há uma probabilidade de 10% dos nós serem duplicados, mas se os títulos forem iguais, então existe uma probabilidade de 90% dos nós serem duplicados.

Probabilidade Condicional PC2

A probabilidade de considerar os filhos dos nós duplicados, dado que cada par de nós filhos comparáveis é um duplicado, é designada PC2 e formalmente definida como $P\left(C_{t_{ij}} \middle| t_{1_{ij}}, t_{2_{ij}}, \ldots\right)$. Intuitivamente, faz sentido dizer que dois nós são duplicados apenas se todos os seus filhos o forem também. Contudo, pode acontecer o caso em que, por exemplo, a árvore XML está

incompleta ou contem informação errónea. Assim, a abordagem proposta relaxa este pressuposto e considera que ***quantos mais nós filhos em ambas as árvores forem duplicados, maior é a probabilidade dos nós pais serem duplicados***. Isto é representado pela Equação (3).

$$P\left(C_{t_{ij}} \left| t_{1_{ij}}, t_{2_{ij}}, ..., t_{n_{ij}} \right.\right) = \frac{1}{n} \times \sum_{k=1}^{n} t_{k_{ij}} \qquad (3)$$

De acordo com a Equação (3), a probabilidade é directamente proporcional ao número de nós filhos que são duplicados. Por exemplo, na rede da Figura 7, a probabilidade $P\left(C_{mv_{11}} \left| dr_{11}, cst_{11} \right.\right)$ é definida como $P\left(C_{mv_{11}} \left| dr_{11}, cst_{11} \right.\right) = \left(dr_{11} + cst_{11}\right)/2$.

Probabilidade Condicional PC3

Para definir a probabilidade de dois nós serem duplicados, dado que os seus valores e os valores dos seus filhos são duplicados, ou seja, $P\left(t_{ij} \left| V_{t_{ij}}, C_{t_{ij}} \right.\right)$, os nós são considerados duplicados se ambos os seus valores ou os seus filhos forem duplicados. Assim, a probabilidade é definida tal como na Equação (4).

$$P\left(t_{ij} \left| V_{t_{ij}}, C_{t_{ij}} \right.\right) = \begin{cases} 1 & se\ V_{t_{ij}} = C_{t_{ij}} = 1 \\ 0 & caso\ contrário \end{cases} \qquad (4)$$

Probabilidade Condicional PC4

Finalmente, são definidas as probabilidades de um conjunto de nós do mesmo tipo ser duplicado, dado que cada par de nós no conjunto é duplicado, isto é, $P\left(t_{**} \left| t_{1*}, t_{2*}, ... \right.\right)$ e $P\left(t_{i*} \left| t_{i1}, t_{i2}, ... \right.\right)$. É definida, em primeiro lugar, a probabilidade $P\left(t_{**} \left| t_{1*}, t_{2*}, ... \right.\right)$, que corresponde à probabilidade do conjunto de nós ser duplicado, dado que cada um dos seus nós é um duplicado. Tal como antes, é assumido que quantos mais nós forem duplicados, maior é a probabilidade de todo o conjunto de nós ser duplicado. A probabilidade pode, assim, ser definida como demonstrado na Equação (5).

$$P\left(t_{**} \left| t_{1*}, t_{2*}, ..., t_{n*} \right.\right) = \frac{1}{n} \times \sum_{k=1}^{n} t_{k*} \qquad (5)$$

Pode agora definir-se $P(t_{i*}|t_{i1}, t_{i2}, ...)$, que reflecte a probabilidade de um nó numa árvore XML ser um duplicado, se este for duplicado de, pelo menos, um nó do mesmo tipo na outra árvore XML. Isto é representado na Equação (6).

$$P(t_{i*}|t_{i1}, t_{i2}, ..., t_{in}) = \begin{cases} 1 & se \exists_j | t_{ij} = 1 \\ 0 & caso\ contrário \end{cases} \qquad (6)$$

4.3.2.1 Probabilidade Final

Uma vez que todas as probabilidades condicionais e *à priori* estão definidas, a rede Bayesiana pode ser usada para calcular a probabilidade de duas árvores XML corresponderem a duplicados, isto é, $P(t_{11})$, onde *t* é a etiqueta do nó raiz de ambas as árvores. Isto pode ser alcançado através de qualquer algoritmo de propagação de probabilidades, tal como os descritos em [50].

O cálculo da probabilidade para a rede da Figura 7 pode ser demonstrado. Neste caso pretende-se calcular a probabilidade das árvores XML *U* e *U'* serem duplicados, isto é, $P(mv_{11})$. De acordo com a rede, e aplicando a Equação (4), a probabilidade é definida como:

$$P(mv_{11}) = \sum_{V_{mv_{11}}, C_{mv_{11}}} P(mv_{11}|V_{mv_{11}}, C_{mv_{11}}) P(V_{mv_{11}}) P(C_{mv_{11}}) = P(V_{mv_{11}}) \times P(C_{mv_{11}}) \qquad (7)$$

Analogamente, aplicando a Equação (2), a probabilidade $P(V_{mv_{11}})$ é definida como:

$$P(V_{mv_{11}}) = w_{year} P(mv_{11}[year]) + w_{title} P(mv_{11}[title]) \qquad (8)$$

Relativamente à probabilidade $P(C_{mv_{11}})$, de acordo com a Equação (3), temos:

$$P(C_{mv_{11}}) = \frac{P(dr_{11}) + P(cst_{11})}{2} \qquad (9)$$

Segue-se o cálculo da probabilidade $P(dr_{11})$, usando as Equações (4) e (2), como em seguida:

$$P(dr_{11}) = w_{value} P(dr_{11}[value]) = P(dr_{11}[value]) \qquad (10)$$

sabendo que $w_{value} = 1$, de acordo com a Equação (2).

Analogamente, para $P(cst_{11})$, usando as Equações (3) e (5):

$$P(cst_{11}) = P(ac_{**}) = \frac{P(ac_{1*}) + P(ac_{2*})}{2} \qquad (11)$$

Usando as Equações (6) e (2) podemos calcular a probabilidade $P(ac_{1*})$ como:

$$P(ac_{1*}) = 1 - \prod_{i=1}^{3} (1 - P(ac_{1i}[value])) \qquad (12)$$

Uma equação semelhante pode ser obtida para $P(ac_{2*})$.

Finalmente, juntando as Equações (7) a (12), obtém-se:

$$\begin{aligned} P(mv_{11}) &= (w_{year} P(mv_{11}[year]) + w_{title} P(mv_{11}[title])) \\ &\times (P(dr_{11}[value]) + \left(1 - \prod_{i=1}^{3} (1 - P(ac_{1i}[value])) \right. \\ &\left. + 1 - \prod_{i=1}^{3} (1 - P(ac_{2i}[value])) \right) \times \frac{1}{2}) \times \frac{1}{2} \end{aligned} \qquad (13)$$

4.3.2.2 Eficiência e Limitações

Pode facilmente ser constatado que, se as árvores XML que estão a ser comparadas não possuírem múltiplos nós do mesmo tipo, o procedimento proposto na Secção 4.2 é linear no número de nós da maior árvore, sendo que é criado um número constante de nós na rede para cada par de nós nas árvores XML. Contudo, se ocorrerem múltiplos nós do mesmo tipo, o procedimento resultante pode ser da ordem de O(*n* x *n'*), onde *n* e *n'* representam o número de nós a ser comparado em cada árvore. Apesar disso, isto apenas aconteceria no pior caso, se todos os tipos nas árvores XML (com a excepção dos nós de raiz) permitissem a repetição dos seus elementos. Assume-se, no entanto, que esta é uma situação muito rara e que a maioria dos dados armazenados em XML permite a repetição de apenas alguns dos seus elementos. Deste modo, o algoritmo proposto deve ser realizável na maioria dos casos. Além disso, note-se que, na prática, o procedimento de construção do modelo é usado apenas uma vez. Como é assumido que todos os objectos do conjunto de dados estão em conformidade com o mesmo esquema, o modelo vai ser igual para todos os objectos, possuindo apenas uma pequena alteração quando calcula as probabilidades para nós repetidos do mesmo tipo, de forma a contemplar o diferente número de nós que podem ocorrer. Finalmente, uma vez que a rede

Bayesiana construída não possui ciclos, calcular as probabilidades é linear em relação ao número de nós da rede. Como pode ser observado nas Equações (1) a (6), todas as probabilidades podem ser calculadas num tempo constante ou linear (se considerarmos o custo de calcular a similaridade entre valores de nós constante).

4.4 Implementação

Esta secção descreve o modo como o algoritmo de detecção de duplicados em XML foi implementado, de maneira a reflectir o modelo enunciado nas secções anteriores. O algoritmo pode ser dividido em dois passos essenciais, a configuração do algoritmo e a comparação dos objectos, descritos em 4.4.1 e 4.4.2, respectivamente. Na etapa da configuração, o utilizador fornece a estrutura dos objectos contidos na base de dados e decide quais os parâmetros que pretende que o algoritmo considere na sua execução. Na fase de comparação, o algoritmo constrói a rede Bayesiana e, em seguida, calcula as probabilidades da rede tendo em conta os parâmetros definidos pelo utilizador na fase anterior. É nesta segunda fase que o algoritmo decide, para cada par de objectos, se estes são ou não considerados duplicados. Toda a implementação do algoritmo foi realizada na linguagem de programação Java [51], recorrendo à plataforma do DOM [52] como *parser* de documentos XML.

4.4.1 Configuração

Antes do algoritmo correr, é necessário que o utilizador defina um conjunto de parâmetros que vão afectar o modo como a comparação dos objectos vais ser efectuada. O utilizador tem possibilidade de configurar o algoritmo escolhendo os seguintes parâmetros: (i) nós do objecto que pretende que sejam avaliados, (ii) valores para a probabilidade por omissão (referida na Secção 4.3.1) e (iii) funções utilizadas para avaliar as probabilidades propagadas para cada nó.

No primeiro parâmetro o utilizador pode marcar os nós da árvore, indicando se pretende que estes sejam considerados pelo algoritmo ou não. Por exemplo, imaginemos o caso em que o utilizador pretende detectar duplicados numa base de dados de filmes. Eventualmente, este não terá grande interesse que o país de origem de cada filme seja avaliado, pois esse campo ajudará muito pouco a diferenciar um filme da maior parte dos demais. Nesta situação o utilizador possui toda a vantagem em marcar este campo para que não seja avaliado, pois não só irá obter uma maior qualidade nos duplicados identificados, mas também poupará tempo computacional na execução do algoritmo, visto que a rede construída será de menor dimensão e menos similaridades serão calculadas. De notar que, se o utilizador marcar um nó que não

deseja ver comparado, e esse nó possuir filhos, nenhum dos seus filhos será considerado na comparação. Como a rede Bayesiana é construída de acordo com a hierarquia dos objectos da base de dados, nós que apareçam abaixo dos não incluídos na rede não estarão presentes na mesma e, portanto, não serão considerados.

No segundo parâmetro, o utilizador tem a possibilidade de definir uma probabilidade por omissão diferente para cada um dos nós do objecto que pretenda que sejam tidos em conta na comparação. Esta probabilidade por omissão é utilizada nos casos em que um ou ambos os objectos do par a ser comparado não possuem um determinado nó que o utilizador pretende que seja incluído na comparação. Neste caso, em lugar da probabilidade dos nós serem duplicados ser produzida a partir da medida de similaridade entre o seu conteúdo, ou propagada através de outros nós da rede, é aplicada a probabilidade por omissão previamente definida pelo utilizador. Através deste parâmetro o utilizador pode, de certa forma, indicar ao algoritmo a importância que representa a falta da informação de determinado nó na identificação de objectos duplicados.

O último parâmetro possibilita ao utilizador empregar diversas funções de avaliação das probabilidades propagadas na rede. Estas funções são definidas para cada nó da rede, podendo ser definidas diferentes funções para diferentes nós. A definição destas funções permite ao utilizador tirar partido do seu conhecimento prévio da base de dados. Assim, se este quiser pode, por exemplo, utilizar num nó uma função *noisy-OR* [53], que equivale, na prática, a dizer que basta que dois dos nós XML filhos (um de cada objecto) sejam duplicados para o nó ser considerado duplicado. Por outro lado, se o utilizador pretender utilizar uma avaliação mais ponderada das probabilidades pode utilizar uma função que calcula uma média entre as probabilidades recebidas. O algoritmo possui a flexibilidade de permitir a fácil implementação de qualquer função deste tipo. Um exemplo da utilização destas funções pode ser observado na Figura 8, que ilustra uma rede criada a partir de um par de objectos. Estes objectos representam uma pessoa p_1, que possui um nome nm_1 e uma morada add_1. Esta morada é representada por dois campos, rua str_1 e número nmb_1. A rede resultante é apresentada de um modo simplificado, apenas com o intuito de fornecer uma visão de como a propagação de probabilidades resulta com a aplicação das referidas funções.

Todos os parâmetros aqui descritos são introduzidos num ficheiro de configuração juntamente com a estrutura hierárquica dos objectos da BD, tal como exemplificado na Figura 9. Quando o algoritmo é iniciado, este ficheiro é lido e a sua informação é copiada para memória sob a forma de uma estrutura que respeita a hierarquia dos campos dos objectos. Tal estrutura é utilizada numa fase posterior de construção da rede, tal como descrito na Secção 4.4.2.

Além dos parâmetros aqui descritos, a utilização de uma rede deste tipo facilita bastante a introdução de novos parâmetros nos seus nós. O facto de qualquer tipo de informação poder

ser armazenado nos nós, para posterior utilização aquando da avaliação da rede, possibilita que novos parâmetros como, por exemplo, atribuição de pesos a nós seja de implementação trivial. Nesta secção apenas foram descritos os parâmetros considerados de maior relevância no contexto dos testes efectuados.

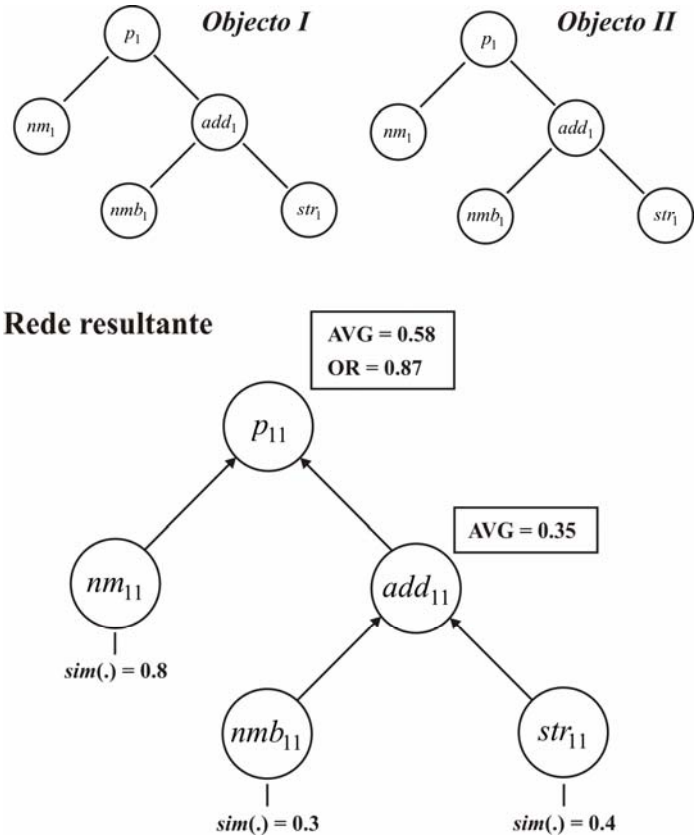


Figura 8: Aplicação de funções para as probabilidades condicionais numa rede Bayesiana simplificada.

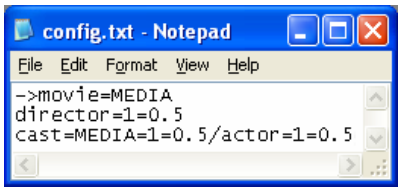


Figura 9: Conteúdo de um ficheiro de configuração para base de dados com objectos com a mesma estrutura dos apresentados na Figura 1.

4.4.2 Comparação e Construção da Rede

A segunda etapa da execução do algoritmo diz respeito à fase da comparação. É neste período que os objectos pertencentes à base de dados são agrupados em pares e comparados entre

si. O modo como funciona esta etapa do algoritmo será explicado sob uma perspectiva do que acontece quando um par de objectos é comparado, sendo que todos os restantes são processados analogamente.

Na seguinte descrição será usada a denominação “objectos Java” para referir a estrutura de dados usada para implementar os nós da rede Bayesiana. Os objectos contidos na base de dados serão designados por “objectos XML”. Igualmente, serão designados por “nós XML” os nós dos objectos XML.

Quando um par de objectos XML é seleccionado para comparação, o primeiro passo consiste em construir a respectiva rede. Esta é, na prática, composta por listas que representam os diferentes níveis da rede. As listas contêm objectos Java que representam os nós da rede e que contêm toda a informação referente aos parâmetros de configuração descritos na secção anterior, assim como os nós dos objectos comparados, o seu tipo, o seu conteúdo e alguma informação adicional. No caso dos objectos Java representarem nós intermédios na rede Bayesiana, estes possuem um apontador para uma lista que caracteriza o nível imediatamente abaixo. Esse nível é obtido a partir das comparações entre os filhos (que representam o mesmo tipo) dos dois nós contidos nesse objecto Java. Um exemplo da construção da estrutura da rede aqui descrita pode ser observada na Figura 10. Quando os objectos Java contêm nós terminais, estes possuem a informação necessária para gerar a probabilidade de duplicação entre eles, ou seja, possuem o valor dos nós XML.

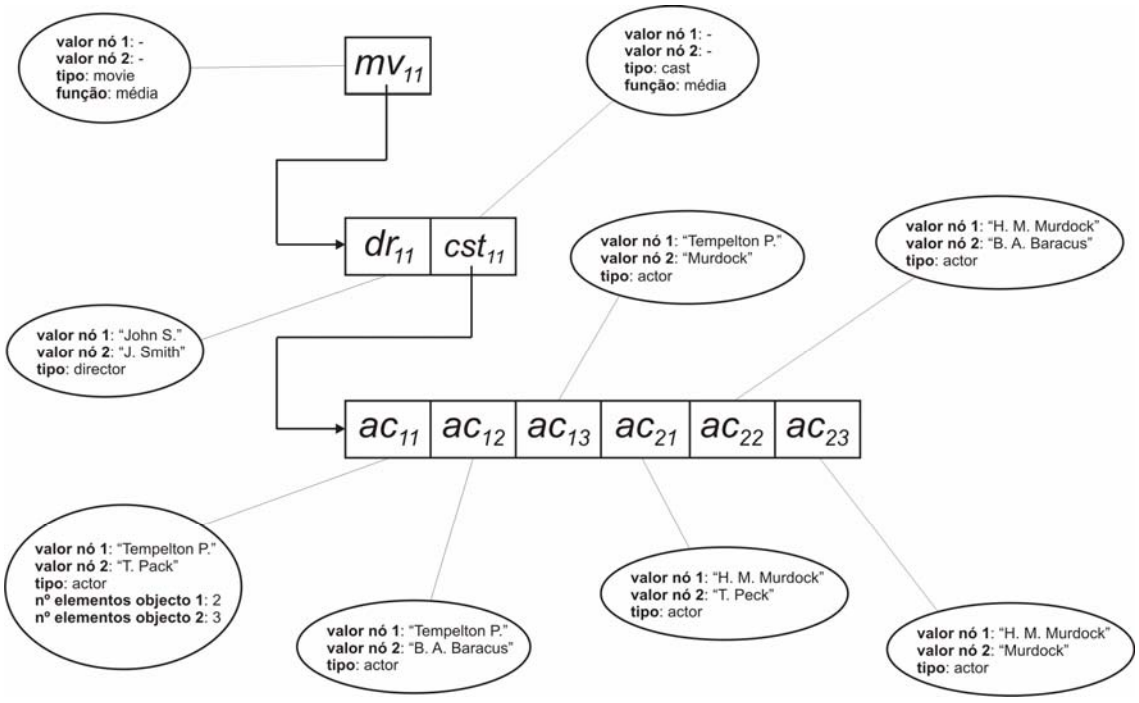


Figura 10: Estrutura e conteúdo de uma rede construída a partir dos dois objectos XML presentes na Figura 1.

A construção da rede é feita agrupando cada combinação possível de nós pertencentes ao mesmo tipo num objecto Java. De facto, cada objecto Java representa a informação de dois nós do mesmo tipo, pertencentes a objectos XML diferentes, e está contido no nível da rede correspondente ao nível onde os nós XML se encontram no objecto XML. A informação de configuração que é adicionada aos objectos Java é proveniente da estrutura mencionada na Secção 4.4.1 que, por sua vez, é obtida através do ficheiro de configuração. Esta estrutura é percorrida simultaneamente à criação da rede, visto que também respeita a estrutura hierárquica dos objectos XML, de modo a que a informação nela contida seja atribuída aos nós correspondentes. Nesta estrutura, a hierarquia dos objectos XML é obtida através da utilização de um conjunto de listas ligadas. Cada posição da lista possui todos os parâmetros de configuração definidos pelo utilizador, assim como o tipo de cada nó. A informação do tipo é importante para garantir que, apesar dos nós manterem a estrutura definida pelo utilizador, o algoritmo suporta ordens de ocorrência diferentes nos objectos XML. Esta informação é utilizada numa fase de normalização dos níveis que ocorre imediatamente antes da construção de cada nível da rede. A normalização visa colocar todos os nós na ordem definida pelo utilizador, nos casos em que tal é necessário. Como exemplo desta situação, considere-se a rede produzida na Figura 8 a partir dos objectos I e II. Poderia suceder o caso em que o objecto XML correspondente ao objecto I apresentasse o nó nmb_1 antes do nó str_1 e o objecto II apresentasse a ordem inversa. Poderia ainda acontecer que um dos nós mencionados não estivesse presente em algum dos dois objectos, sendo por isso necessário assinalar essa ausência. É nestes casos, em que os nós dos objectos XML não ocorrem com a mesma ordem ou que existem nós em falta, que é necessário normalizar os níveis de modo a garantir que os pares de nós comparados pertencem sempre ao mesmo tipo. É importante assinalar que a questão das múltiplas ocorrências de elementos do mesmo tipo no mesmo objecto, tal como descrita na Secção 4, é adiada para a avaliação da rede. Na verdade, no momento da construção da rede todos os objectos Java criados são colocados indiscriminadamente no nível a que pertencem e por ordem de ocorrência, nomeadamente, os que correspondem a comparações entre múltiplos filhos do mesmo tipo. Quando existem, em determinado nível, vários objectos Java resultantes das comparações de múltiplos filhos do mesmo tipo, o primeiro que aparece na lista contém informação sobre quantos elementos deste tipo possui cada um dos objectos que está a ser comparado. Esta informação vai ser utilizada para, na fase da avaliação da rede, inferir quantos destes elementos estão presentes naquele nível.

4.4.3 Avaliação

Depois de construída a rede, prossegue-se para a sua respectiva avaliação. Esta avaliação tem por objectivo produzir um valor numérico entre 0 e 1 que representará a probabilidade do par ser, de facto, um duplicado. Este número, como anteriormente mencionado, é obtido através da propagação das probabilidades encontradas nas folhas da rede através da medida

de similaridade ou, em alguns casos, dada pela probabilidade por omissão definida quer para os nós terminais quer para os não terminais. Assim, a rede é avaliada de forma recursiva, sendo que para cada nó é invocada a função lógica que lhe foi atribuída na fase da configuração. No caso do algoritmo detectar que o objecto Java que está a avaliar na altura é um de vários objectos que advém da ocorrência de vários elementos do mesmo tipo, a probabilidade obtida vai ser a resultante de uma sub-rede construída de acordo com a presente no nó ac_{**} na rede da Figura 7. Para que o algoritmo possa construir a sub-rede, este terá de aceder à informação contida no nó, que indica quantas ocorrências existem em cada objecto. Esses dois valores vão fornecer informação sobre quais os nós que ficam nas duas sub-redes.

Encontrada a probabilidade final, o algoritmo decide, segundo um *threshold* definido pelo utilizador, se o par em questão é classificado como duplicado ou não. Se for, é adicionado o índice dos dois objectos, ou seja, a posição que ocupam na base de dados, a um ficheiro de resultados. Entre cada adição é garantida uma ordenação das probabilidades por ordem decrescente. Assim, quanto mais próximas do topo estiverem os resultados dos pares de objectos, maior é a sua probabilidade de serem realmente duplicados.

4.4.4 Limitações

Apesar da funcionalidade final do algoritmo ter ficado bastante completa, este apresenta algumas limitações que serão apresentadas nesta secção, e estão relacionadas com os seguintes tópicos:

- **Introdução de parâmetros adicionais:** exemplo das limitações mencionadas é o facto do algoritmo não permitir a introdução de mais parâmetros além dos descritos na Secção 4.4.1. Os parâmetros que se encontram à disposição do utilizador foram considerados fulcrais para uma modelação eficaz do modo como os objectos seriam comparados, com base no conhecimento de uma base de dados por parte de um indivíduo. No entanto, existem mais parâmetros que poderiam ser de alguma utilidade para este processo. A atribuição de pesos aos objectos, não implementada, poderia ser um bom modo do utilizador estabelecer uma ordem de importância entre os vários nós de determinado nível. Assim, este poderia atribuir maior peso a campos que considerasse serem de essencial importância para a identificação do objecto. Juntamente com o peso, poderiam ser explorados mais alguns parâmetros que, eventualmente, pudessem ser de alguma utilidade para o método mas, com as limitações no tempo disponível para este projecto e sem garantias de que isso representasse uma mais valia na qualidade do algoritmo, optou-se por apenas incluir os já mencionados. Apesar disso, e considerando o modo como a rede Bayesiana se

encontra implementada, seria necessário apenas um pequeno esforço para a inclusão de parâmetros adicionais.

- **Medidas de similaridade:** outra limitação do algoritmo está directamente relacionada com a aplicação das medidas de similaridade entre *strings*. Este algoritmo apenas usa uma medida de *edit distance* normalizada como medida de similaridade. Isto representa uma limitação se considerarmos que um objecto possui elementos de vários tipos, ou seja, uma medida de similaridade que funciona extremamente bem para comparar *strings* que representam nomes de pessoas, pode não ser a melhor solução para comparar *strings* que representem anos. Tome-se por exemplo o caso em que se pretende comparar duas *strings* que representam anos distintos, por exemplo, “1998” e “1999”. Obviamente, se assumirmos a ausência de erros tipográficos, os campos que as contêm representam anos diferentes. No entanto, o resultado produzido por uma medida de similaridade como um *edit distance* normalizado iria ser muito próximo de 1, visto que a *string* apenas apresenta uma diferença ao nível de um caracter. Tal já não seria um problema no caso de dois nomes como “Rui J. Silva” e “Rui Silva”, pois estes possuem grande probabilidade de representarem a mesma pessoa e, portanto, é desejável que seja produzido um valor probabilístico alto. Assim sendo, o facto do algoritmo não permitir ao utilizador definir uma medida de similaridade para cada campo do objecto, em função das suas características, é um factor limitativo.

- **Avaliação de atributos dos nós:** a única limitação do algoritmo que representa uma não conformidade com o modelo proposto neste projecto é a ausência da avaliação de atributos dos nós. Na verdade, este algoritmo apenas considera na construção da rede os nós e os seus respectivos conteúdos, ignorando os atributos. Esta situação pode ser algo limitativa no caso dos objectos possuírem informação relevante nos atributos dos seus nós. No entanto, para muitas bases de dados, nomeadamente as testadas e que figuram na parte experimental, esta funcionalidade não possui qualquer utilidade pois toda a informação dos seus objectos está representada no conteúdo dos nós. Para que o algoritmo tivesse em conta os atributos dos nós apenas seria necessário alterar um pouco a lógica de avaliação da rede. Tal não seria demasiado complexo, uma vez que os nós da rede possuem toda a informação necessária sobre os nós XML dos objectos que estão a ser comparados, incluindo o valor dos atributos. Na prática, do ponto de vista da rede, os atributos seriam encarados como nós adicionais.

- **Escalabilidade:** a última das limitações aqui citadas é uma limitação funcional que em nada influencia a qualidade pretendida para os resultados do algoritmo. Esta prende-se como a questão do algoritmo, antes da fase de comparação dos pares de objectos, carregar integralmente a base de dados para a memória física do computador. Isto limita a escalabilidade do algoritmo ao nível da dimensão das bases de dados que

processa. No entanto, esta restrição em nada afecta a qualidade do processo de detecção de duplicados e possui pouca relevância dentro do carácter experimental deste projecto, que tem por objectivo apenas demonstrar a eficácia do modelo proposto. A resolução deste problema implicaria um esforço de carácter técnico e programático, sem qualquer influência nos resultados apresentados na Secção 5.

Das várias limitações que aqui foram apresentadas, relativamente ao funcionamento do algoritmo, pode ser constatado que a maioria se prende com limitações de tempo para a concretização do projecto. Todas elas representam o fruto de uma gestão de esforço e de um direccionamento do mesmo para componentes do projecto que foram consideradas de maior importância no âmbito do objectivo inicialmente delineado.

5 Avaliação Experimental

Nesta secção é avaliado o método desenvolvido para a detecção de duplicados em XML. A avaliação é feita em termos de eficácia em dois conjuntos de dados de características diferentes, sendo que um deles contém dados reais e o outro contém dados gerados artificialmente. Esta avaliação cobre o impacto da probabilidade por omissão na eficácia do algoritmo, o impacto da qualidade dos dados, isto é, como erros (tanto erros tipográficos como dados em falta) e a frequência dos mesmos afectam o resultado final. A avaliação inclui também uma comparação com o sistema DogmatiX [17], um algoritmo para detecção de duplicados em XML que representa o estado da arte do que existe neste domínio. No final, na Secção 5.5, e apesar deste não ser o propósito do projecto, é feita uma breve abordagem relativamente à eficiência do algoritmo.

5.1 Ambiente Experimental

Nas secções 5.1.1 e 5.1.2 são descritas as características das bases de dados utilizadas nos testes, bem como as medidas utilizadas como indicadores para os mesmos. Todos os testes foram efectuados num computador com o Windows XP Professional a correr sobre um processador AMD Athlon XP 2600+, com 512MB de memória RAM. A plataforma Java utilizada foi o Java SE 6.

5.1.1 Bases de Dados Utilizadas

Para testar a abordagem proposta foram usados dois conjuntos de bases de dados. O primeiro conjunto continha uma base de dados XML em que parte dos objectos foram gerados artificialmente, recorrendo à ferramenta *DirtyXMLGenerator* [54]. Esta ferramenta permite alterar uma base de dados já existente, gerando objectos duplicados que apresentem diferenças em relação ao original, recorrendo a uma variação de determinados parâmetros da informação contida no objecto. Deste modo foi possível variar estes parâmetros segundo diferentes percentagens, com vista a verificar qual a implicação da variação da qualidade da informação de cada objecto na identificação final dos duplicados. O conjunto de dados artificial, com o nome IMDB, era composto por 4288 objectos distintos que representavam filmes. Estes filmes foram extraídos da *Internet Movie Database* (IMDB) [55]. Para os testes, foi adicionado um conjunto de 4288 objectos duplicados à colecção inicial. Cada objecto gerado é um duplicado de um objecto presente na base de dados inicial, e pode conter os seguintes erros:

(i) erros tipográficos, (ii) dados em falta (por exemplo, faltar o título) e (iii) duplicação de dados com erros (por exemplo o mesmo filme conter dois títulos diferentes). A partir daqui, e caso não seja dada indicação em contrário, erros tipográficos, dados em falta, e duplicação de dados com erros ocorrem com probabilidades de 20%, 10% e 8%, respectivamente.

O segundo conjunto era representado por duas bases de dados distintas contendo dados do mundo real e, portanto, não geradas artificialmente. A primeira, designada IMDB+FilmDienst, foi gerada a partir da integração de 500 filmes extraídos da base de dados da *IMDB*, e os mesmos 500 filmes extraídos do site Web *FilmDienst* [56]. Neste processo não foram gerados quaisquer tipos de dados artificiais. A segunda base de dados, designada FreeDB, continha 10.000 objectos distintos, que representavam CDs, extraídos da base de dados da *FreeDB* [57]. Com muito esforço manual, foram identificados todos os duplicados contidos neste conjunto de dados. Foram encontrados 300 pares de CDs duplicados (cada par foi classificado como duplicado ou não duplicado num processo *double-blind*). Estatisticamente, de acordo com os dados recolhidos pelos utilizadores que participaram no processo de classificação manual, 17.3% das classificações de duplicados foram devido a dados em falta, 12.1% a erros tipográficos e 11.8% devido a duplicação de dados com erros. As restantes classificações foram devido a erros mais particulares como, por exemplo, o caso das pistas de alguns CDs apresentarem como nome a *string* "track" seguida do número da pista. Casos como estes descaracterizam a informação contida no objecto, mas como acontecem com menos frequência possuem bastante menor relevância que os demais. Este processo de identificação de duplicados foi realizado no departamento de informática da Universidade Humboldt de Berlim.

A Figura 11 ilustra os esquemas das bases de dados descritas. O esquema considerado, tanto para a IMDB como para a IMDB+FilmDienst, está identificado apenas como IMBD.

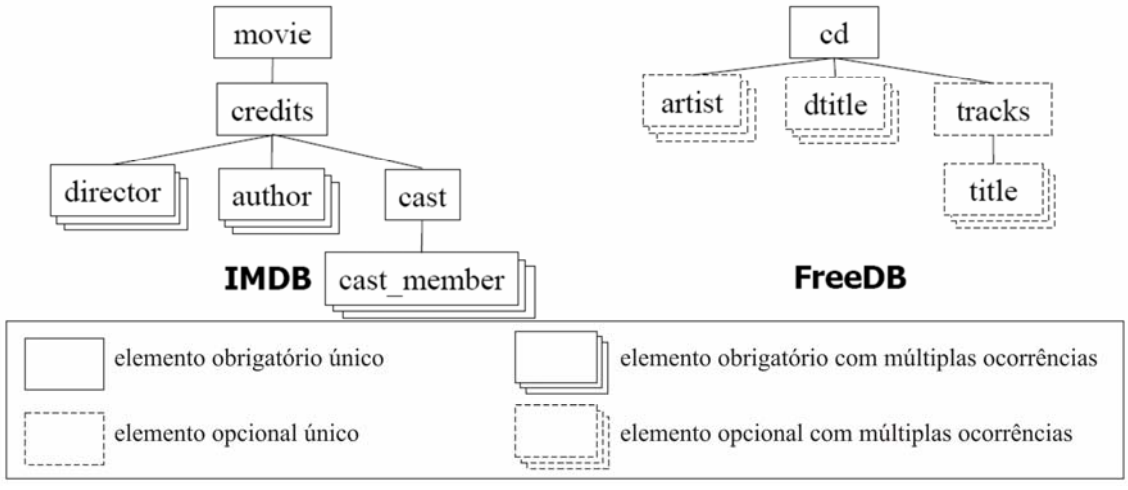


Figura 11: Esquemas XML das bases de dados XML usadas nos testes.

A utilização de bases de dados de diferentes domínios nos testes teve como intuito demonstrar a flexibilidade e adaptabilidade da detecção de duplicados XML aqui proposta aos mais diferentes universos de informação, e também comprovar a sua eficácia em dados obtidos a partir de diferentes tipos de fontes, procurando abranger todas as situações e erros possíveis numa base de dados.

5.1.2 Medidas de Avaliação

As experiências efectuadas em todos os conjuntos de dados procuraram determinar a eficácia do modelo, proposto neste documento, na identificação de pares de objectos duplicados. A rede construída para ambas as colecções (real e artificial) foi parametrizada tal como descrito na Secção 4. Apenas os objectos para os quais a probabilidade era igual ou maior do que 0.6 foram considerados duplicados. Todos os valores dos nós foram considerados como *strings* de texto e foi usado $p = 1 - ed(V_1, V_2) / \max(|V_1|, |V_2|)$ como probabilidade *à priori*, onde $ed(V_1, V_2)$ é a *edit distance* entre os valores V_1 e V_2, e $|V_1|$ é o comprimento da *string* V_1.

Para medir a eficácia do algoritmo, foram usadas as conhecidas medidas de precisão (*precision*) e *recall*. A medida precisão mede a percentagem de pares de objectos identificados como duplicados pelo sistema, que realmente são duplicados, ou seja,

$$precisão = \frac{\left|\{Objectos\ Duplicados\ Existentes\ Na\ BD\} \cap \{Objectos\ Duplicados\ Identificados\ Pelo\ Sistema\}\right|}{\left|\{Objectos\ Duplicados\ Identificados\ Pelo\ Sistema\}\right|}$$

A medida *recall* mede a percentagem de duplicados correctamente identificados pelo sistema em relação à totalidade de objectos duplicados existentes na base de dados. Neste caso,

$$recall = \frac{\left|\{Objectos\ Duplicados\ Existentes\ Na\ BD\} \cap \{Objectos\ Duplicados\ Identificados\ Pelo\ Sistema\}\right|}{\left|\{Objectos\ Duplicados\ Existentes\ Na\ BD\}\right|}$$

Na secção seguinte são discutidas as experiências efectuadas para avaliar a eficácia do algoritmo, utilizando as medidas aqui descritas.

5.2 Parametrização do Algoritmo

A parametrização do algoritmo procura estudar e analisar algumas variáveis utilizadas no modelo com vista a determinar quais os valores que maximizam a eficácia do algoritmo. Nesta

secção são apresentados testes realizados a três parâmetros do algoritmo, e cujos resultados advêm da sua variação. São eles (i) a probabilidade por omissão, (ii) as funções de combinação para filhos múltiplos e (iii) o limite de similaridade. Depois de estabelecidos quais os valores para as variáveis, que permitem ao modelo alcançar melhores resultados, foi testado o impacto da qualidade dos dados no modelo e foram efectuados testes comparativos (Secções 5.3 e 5.4).

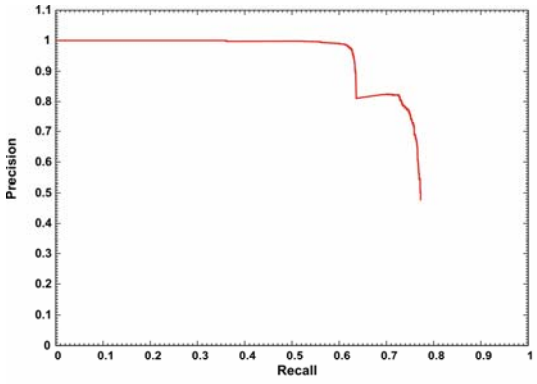
5.2.1 Testes Variando Probabilidade por Omissão

Este conjunto de testes teve por objectivo analisar o impacto da escolha de valores para a probabilidade por omissão no desempenho do modelo. Nesta experiência, fez-se variar a probabilidade por omissão k_a (Equação 1) entre 0 e 0.8. A Figura 12 mostra os resultados das medidas de precisão e r*ecall* na variação de probabilidades por omissão na base de dados IMDB.

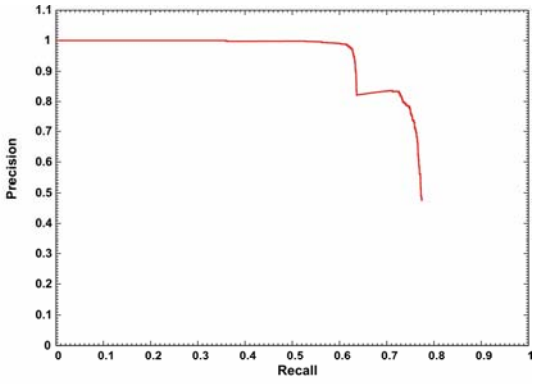
Claramente, a escolha dos valores da probabilidade por omissão afecta a eficácia. Para valores inferiores a 0.2 (Figuras 12(a) a 12(c)) os objectos com dados em falta são muitas vezes considerados como não duplicados, dado que o modelo se comporta como se os elementos em falta fossem completamente diferentes. Isto apresenta um claro impacto no *recall*, que atinge um máximo de 79%. Para valores acima de 0.5 (Figura 12(f)), acontece exactamente o oposto. Neste caso, o modelo é demasiado permissivo, assumindo uma similaridade alta nos elementos que faltam. Assim, apesar de se verificar um aumento significativo no *recall*, a precisão cai abruptamente. Os melhores resultados foram alcançados para valores entre 0.2 (Figura 12(d)) e 0.5 (Figura 12(e)). Nas experiências que se seguem foi usado o valor 0.5 para a probabilidade por omissão, o que maximiza tanto a precisão como o *recall*.

5.2.2 Testes Variando Funções de Combinação para Filhos Múltiplos

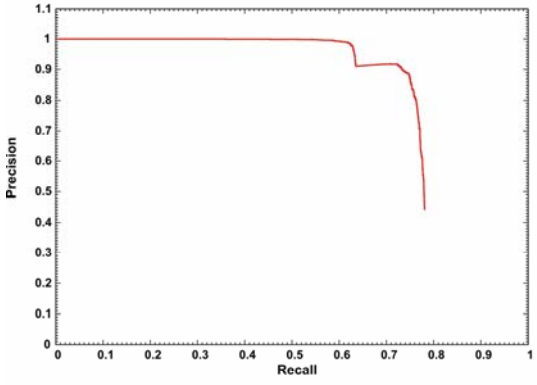
Neste conjunto de testes procurou-se apurar qual a melhor combinação de funções a utilizar no caso de ocorrência de filhos múltiplos do mesmo tipo, como descrito na Secção 4. Tomando por exemplo a Figura 7, isto significa que é necessário definir que função aplicar aos nós ac_{1*} e ac_{2*}, e, por outro lado, que função aplicar em ac_{**}. Nos gráficos da Figura 13 estão apresentados os resultados das diversas combinações de funções. A sua legenda indica em primeiro lugar a função aplicada no nível mais distante da raiz e em segundo lugar no nível mais próximo. Apenas foram realizados testes para estas combinações, pois foram consideradas as de maior relevância.



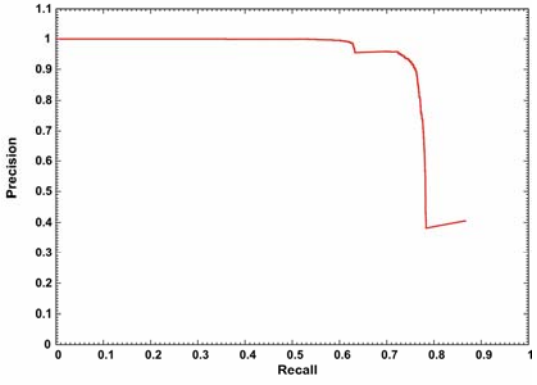
(a) K_a = 0.0



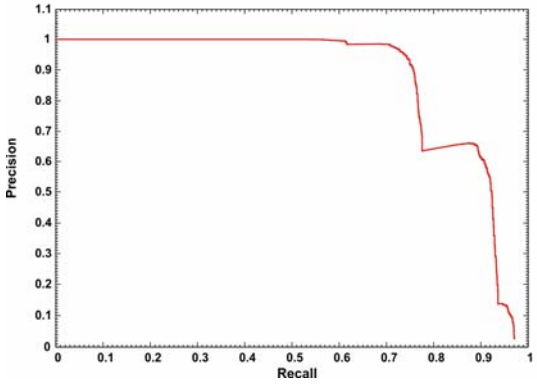
(b) K_a = 0.01



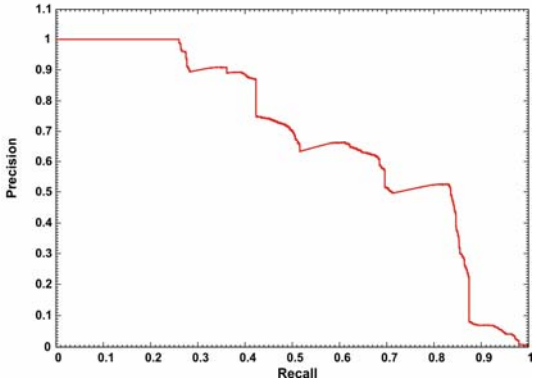
(c) K_a = 0.1



(d) K_a = 0.2

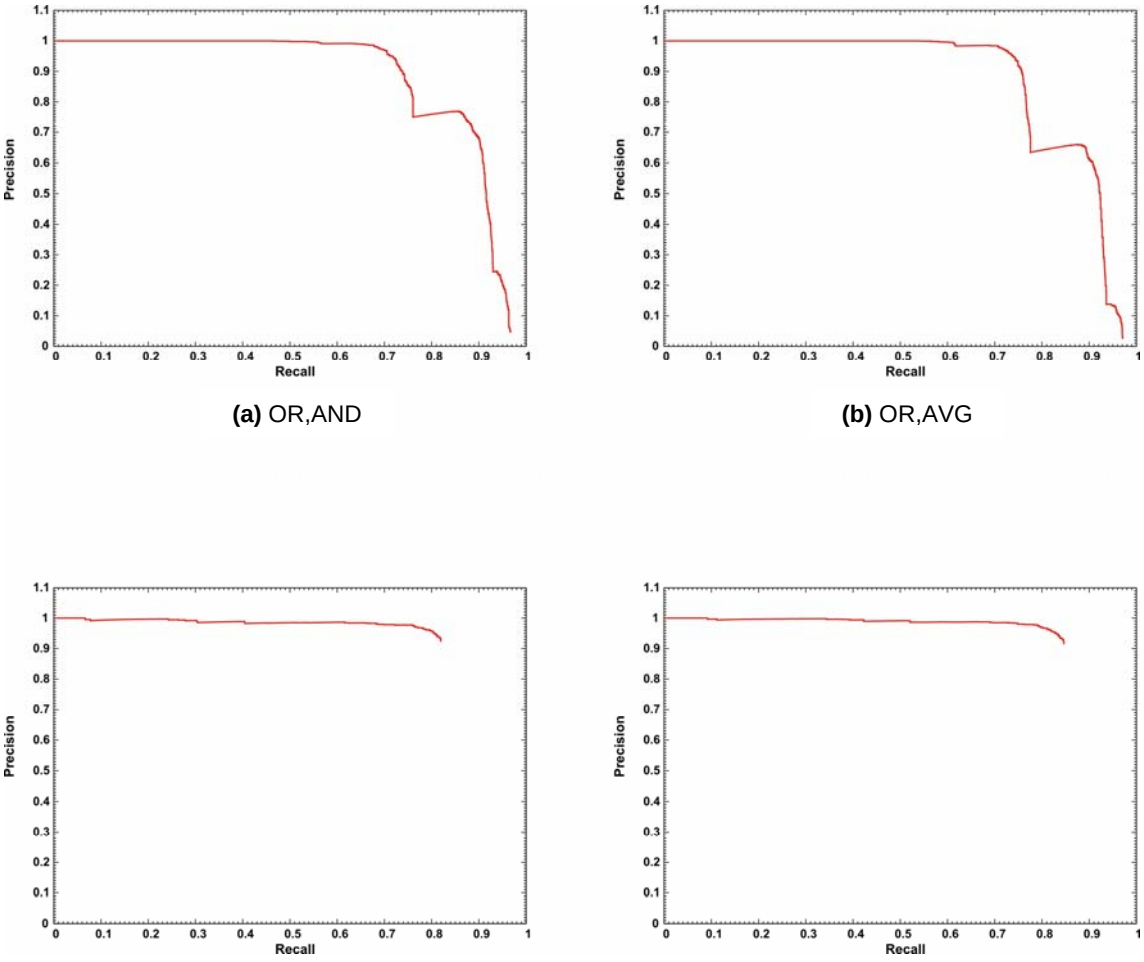


(e) K_a = 0.5



(f) K_a = 0.8

Figura 12: Eficácia variando probabilidade por omissão Ka.



(a) OR,AND **(b)** OR,AVG

(c) XOR,AND **(d)** XOR,AVG

Figura 13: Eficácia variando funções de combinação para filhos múltiplos.

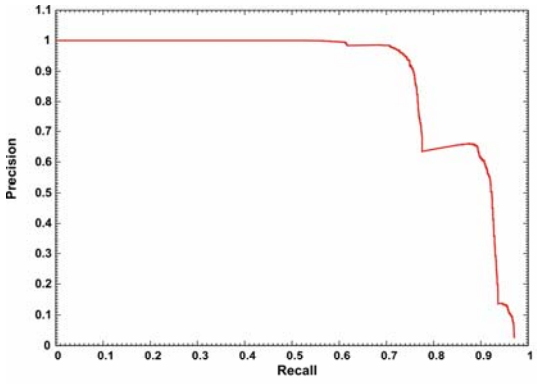
Como se pode observar, as combinações apresentadas nas Figuras 13(c) e 13(d) mostram resultados bastante semelhantes. Ambas possuem valores de precisão muito próximos de 100% para valores de *recall* até cerca de 80% (cerca 85% na Figura 13(d)). No entanto, apesar de possuírem uma precisão perto de 100%, estas combinações revelaram-se demasiado restritivas, nunca ultrapassando valores de *recall* acima dos 85%. Por outro lado, os gráficos das Figuras 13(a) e 13(b) apresentam resultados consideravelmente mais inclusivos, com valores de *recall* que ultrapassam os 95%. A utilização de uma combinação OR,AND apresenta uma precisão de 100% para valores de *recall* até cerca de 55%, ao passo que uma combinação OR,AVG apresenta os mesmos valores de precisão para valores de *recall* até, sensivelmente, 60%.

Optou-se por nos testes seguintes utilizar a combinação OR,AVG, visto ser a que maximiza os valores de precisão e *recall*. No entanto esta é uma escolha que poderá, de futuro, estar ao dispor do utilizador por meio de parâmetros de configuração.

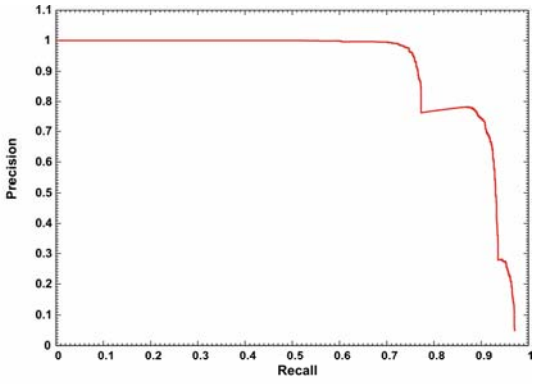
5.2.3 Testes Variando Limite de Similaridade

O propósito deste conjunto de testes foi determinar se existiria alguma vantagem em estabelecer um limite de similaridade para os valores devolvidos pela função *edit distance* utilizada. Este limite consistiria em estabelecer um *threshold t* tal que, se a similaridade entre duas *strings* fosse inferior a *t*, a sua similaridade seria considerada zero. A ideia subjacente a estes testes consistiu em verificar se através deste método se poderiam alcançar resultados mais precisos. Ou seja, o objectivo foi distinguir entre os elementos que apresentavam valores de similaridade realmente altos e, portanto, possuíam uma boa probabilidade de serem duplicados, e os elementos que possuíam uma similaridade com valores médios ou baixos e que poderiam contribuir para a identificação de falsos duplicados. Com esta solução pretendia-se que fosse criado um intervalo significativo entre valores de similaridade obtidos para pares de objectos realmente duplicados e os obtidos para pares de objectos não duplicados. A Figura 14 apresenta a eficácia dos resultados experimentais para diferentes limites de similaridade.

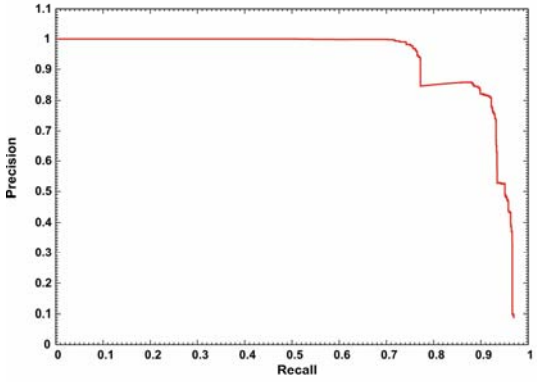
De facto, o limite de similaridade possui alguma influência sobre a qualidade dos resultados obtidos. Pode ser observado que os melhores resultados foram atingidos para limites de similaridade médios, como os apresentados nas Figuras 14(c) e 14(d). Apesar de para qualquer um dos limites de similaridade, definidos entre 0 e 0.8 (Figuras 14(a) a 14(e)), o algoritmo apresentar valores de precisão perto de 100% para valores de *recall* acima de 70%, é para limites entre 0.4 e 0.6 que a precisão mostra uma queda mais suave e, simultaneamente, maximiza os valores de *recall* (superiores a 95%). Os resultados foram mais afectados quando foi aplicado um limite de similaridade alto (Figura 14(f)), onde o modelo apenas atingiu valores de precisão perto de 100% para valores de *recall* até cerca de 50%. Apesar dos resultados obtidos, nos restantes testes efectuados não foi empregue qualquer limite de similaridade, ou seja, utilizou-se um limite de similaridade igual a 0. Esta decisão prende-se com a convicção de que os desempenhos do modelo, usando limites de similaridade, estão fortemente relacionados com o domínio das bases e dados testadas e, portanto, tal só poderia ser confirmado testando mais exaustivamente outro tipo de bases de dados. No entanto, como o valor definido para os limites de similaridade assenta numa variável *hardcoded*, esta poderia ser facilmente transformada em mais um parâmetro de configuração ao dispor de qualquer utilizador, que o deseje adaptar de acordo com o seu conhecimento da base de dados.



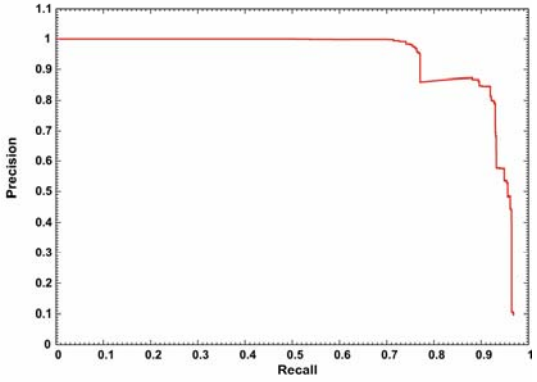
(a) sem limite de similaridade



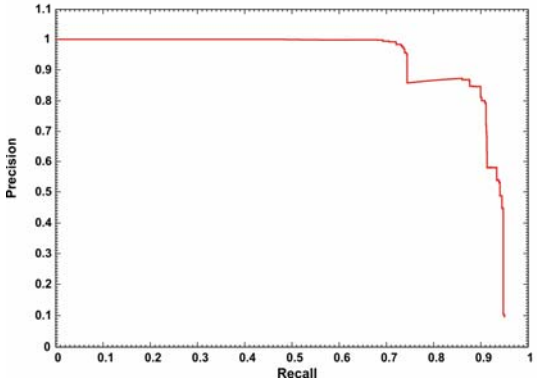
(b) limite de similaridade = 0.2



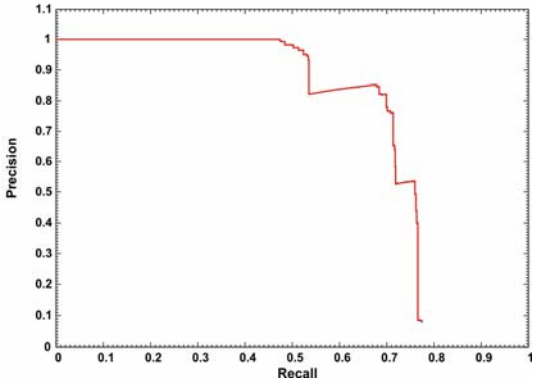
(c) limite de similaridade = 0.4



(d) limite de similaridade = 0.6



(e) limite de similaridade = 0.8



(f) limite de similaridade = 1

Figura 14: Eficácia variando limite de similaridade.

5.3 Impacto da Qualidade dos Dados no Algoritmo

A próxima série de experiências foi realizada para determinar o impacto da qualidade dos dados processados sobre o desempenho do modelo. Para estes testes, a probabilidade de cada um dos três diferentes tipos de erro foi sujeita individualmente a uma variação entre 0% e 50%, aquando da geração de duplicados no conjunto de dados da IMDB, enquanto as restantes probabilidades foram mantidas com um valor fixo. As Figuras 15, 16 e 17 mostram os resultados obtidos com a variação das probabilidades de erros tipográficos, dados duplicados com erros e dados em falta, respectivamente.

5.3.1 Número de Erros Tipográficos

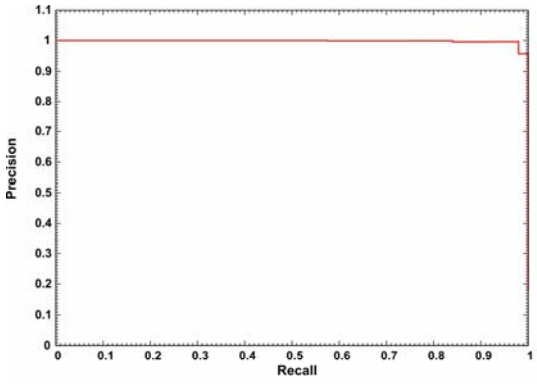
Na Figura 15, pode ser observado que a precisão sofre muito pouco com o aumento do número de erros tipográficos, mantendo os valores perto de 100% para todos os valores expressos. O modelo é, portanto, capaz de lidar adequadamente com a ocorrência de erros nos campos com dados de texto.

5.3.2 Número de Filhos Duplicados com Erros

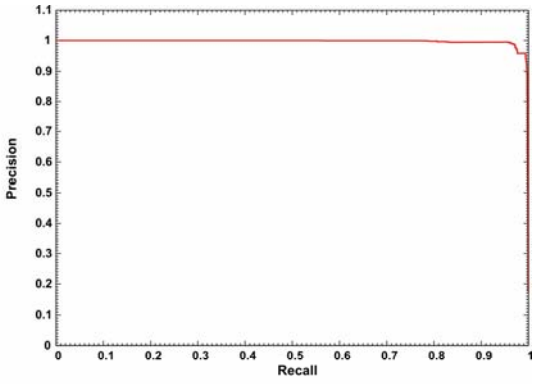
Com a variação do número de filhos duplicados com erros não são verificadas alterações de maior. Como pode ser verificado na Figura 16, apesar do efeito na precisão ser ligeiramente maior, especialmente para probabilidades acima dos 10%, o modelo continua a ser capaz de manter 99% de precisão para valores de *recall* até 84%.

5.3.3 Número de Eliminações

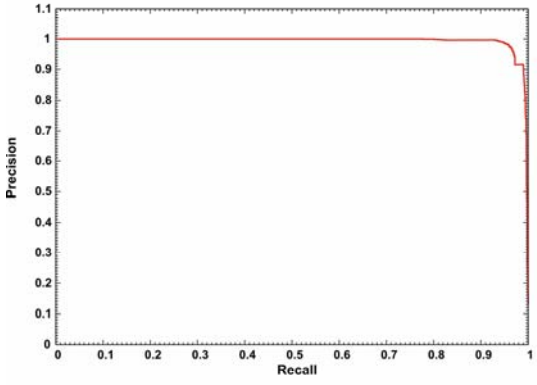
Variar a quantidade de dados em falta revelou ser o teste com maior impacto na qualidade dos resultados, tal como ilustrado na Figura 17. Isto é particularmente evidente no caso da precisão, que decai rapidamente quando os dados se apresentam incompletos. Quando faltam apenas 10% dos elementos XML, a precisão mantém-se acima dos 95% para valores de *recall* até cerca de 74%. Contudo, para 20% de dados em falta, tal apenas pode ser alcançado para valores de *recall* inferiores a 55%. Apesar disso, para valores de *recall* baixos, os valores de precisão permanecem bastante altos, mesmo quando faltam 50% dos dados.



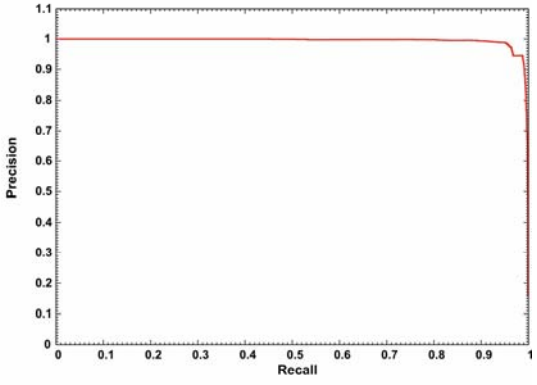
(a) sem erros



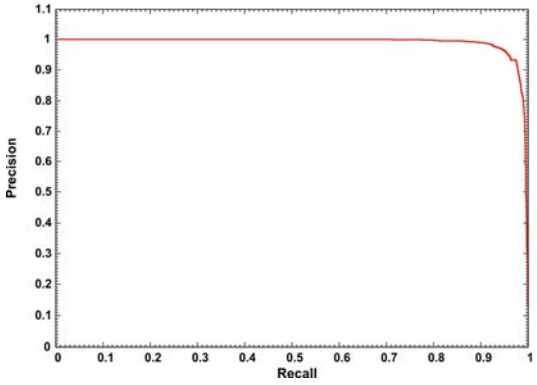
(b) 10% de erros



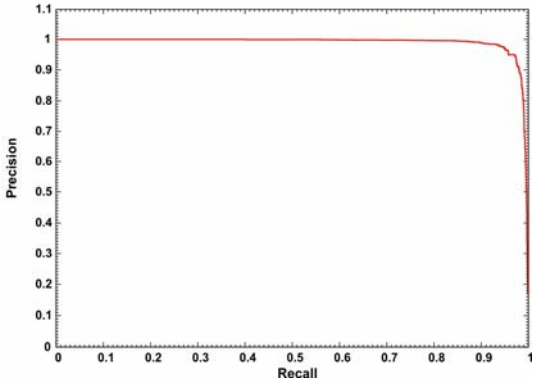
(c) 20% de erros



(d) 30% de erros

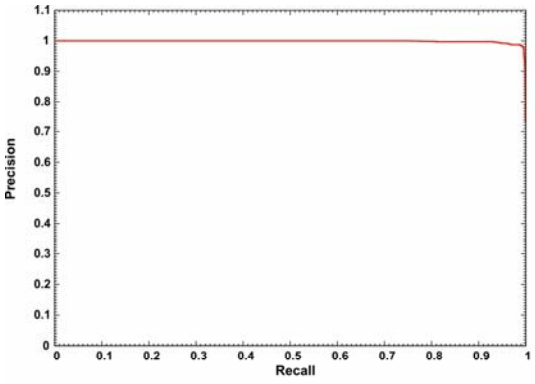


(e) 40% de erros

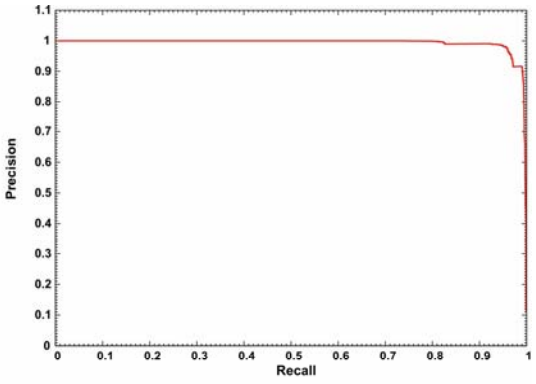


(f) 50% de erros

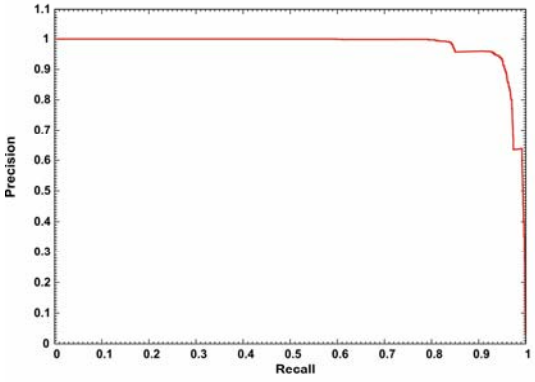
Figura 15: Valores de precisão e *recall* para diferentes probabilidades de ocorrência de erros tipográficos.



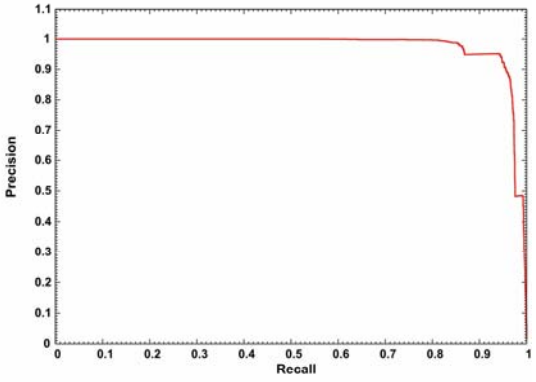
(a) sem dados duplicados com erros



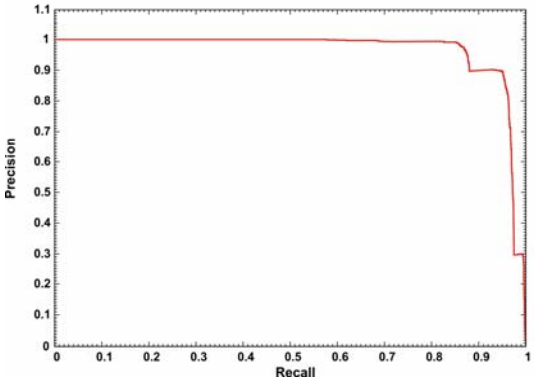
(b) 10% de dados duplicados com erros



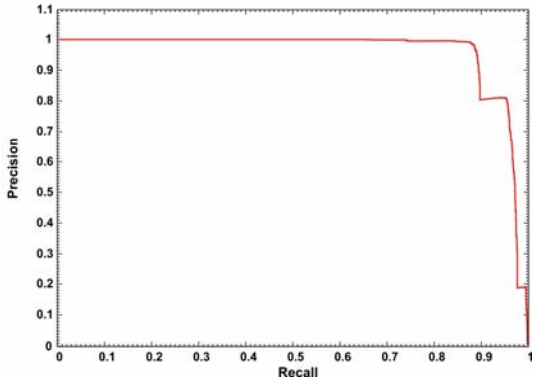
(c) 20% de dados duplicados com erros



(d) 30% de dados duplicados com erros

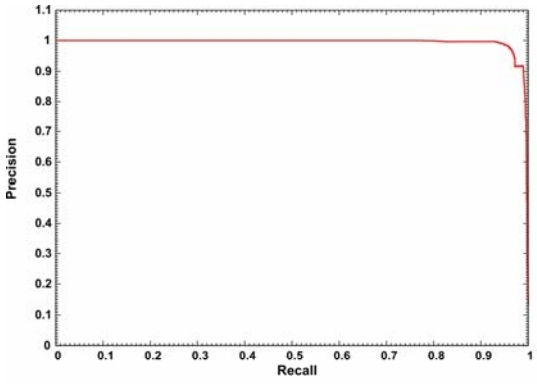


(e) 40% de dados duplicados com erros

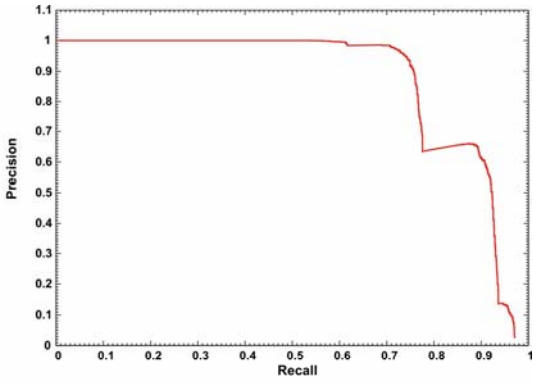


(f) 50% de dados duplicados com erros

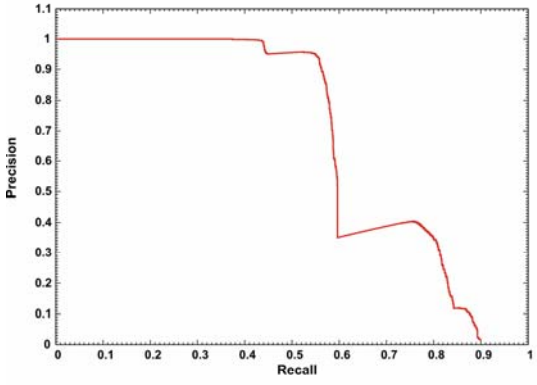
Figura 16: Valores de precisão e *recall* para diferentes quantidades de dados duplicados com erros.



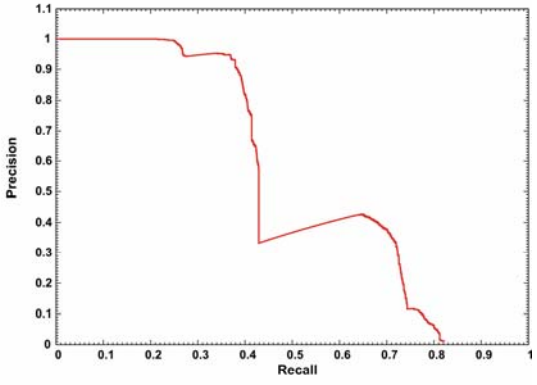
(a) sem dados em falta



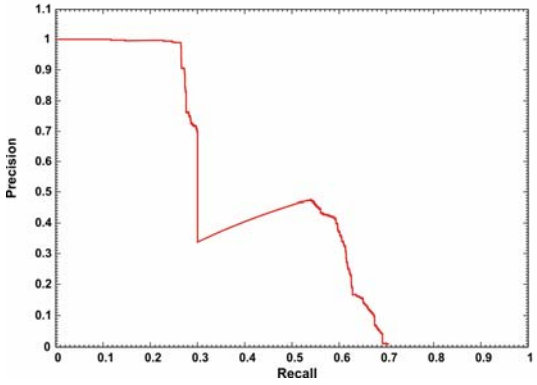
(b) 10% de dados em falta



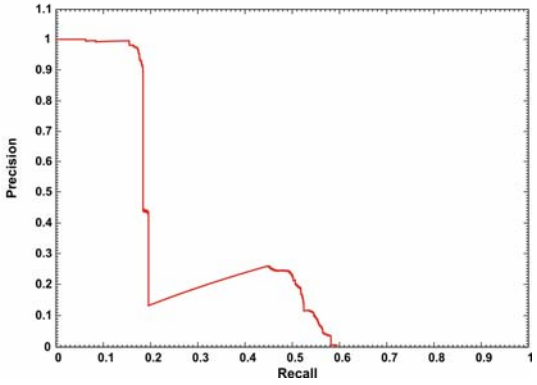
(c) 20% de dados em falta



(d) 30% de dados em falta



(e) 40% de dados em falta



(f) 50% de dados em falta

Figura 17: Valores de precisão e *recall* para diferentes quantidades de dados em falta.

Claramente, o modelo é bastante afectado pela falta de dados quando comparado com os resultados dos outros dois testes. Esta situação acaba por ser espectável, uma vez que o algoritmo funciona com base na informação recolhida, mesmo que errónea e duplicada (erros tipográficos e numero de filhos duplicados com erros). No caso de não ser fornecido qualquer tipo de informação ao algoritmo, no limite, este vai funcionar apenas com base na probabilidade por omissão.

5.4 Comparação com o sistema DogmatiX

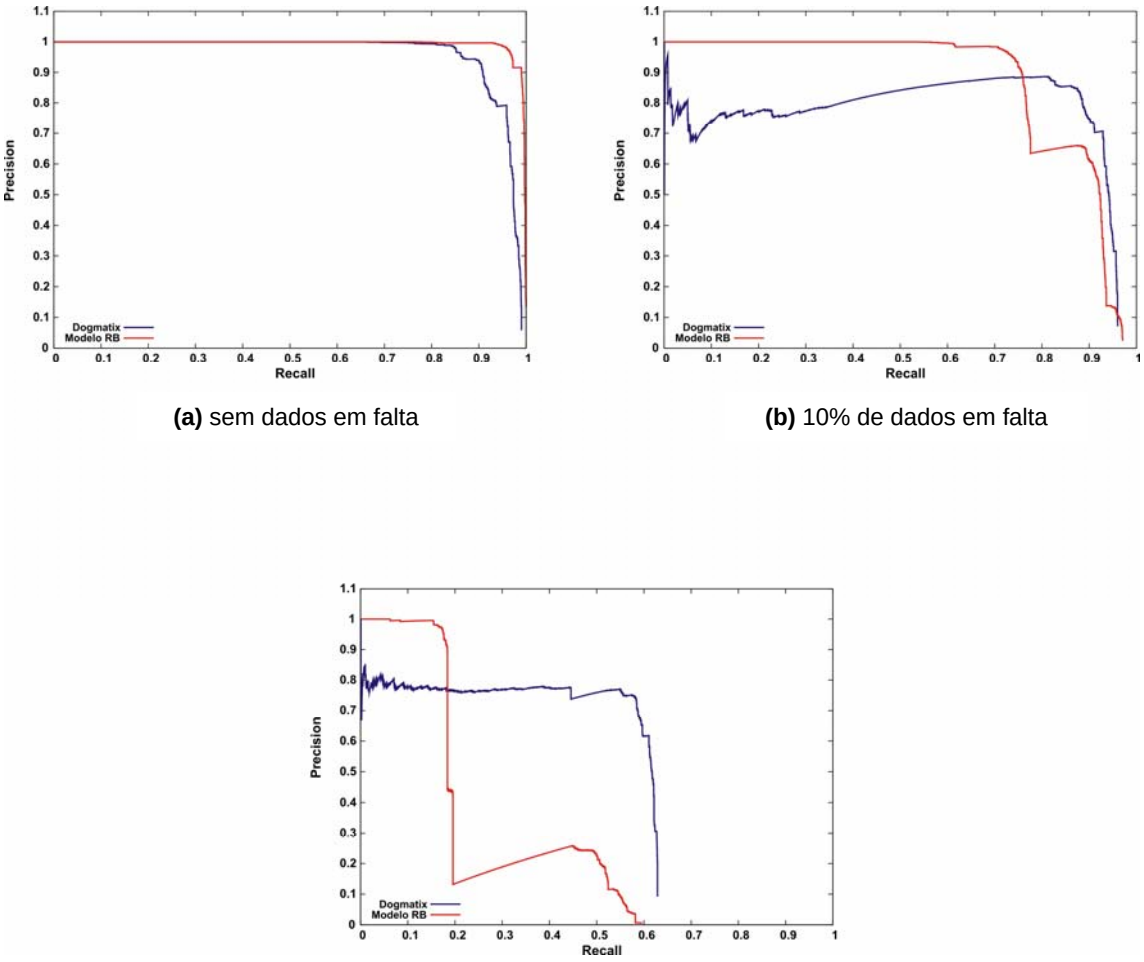
Para fornecer uma perspectiva melhor sobre os resultados obtidos pelo método proposto neste documento, o último conjunto de testes procurou comparar a eficácia da abordagem proposta com outro método de detecção de duplicados em XML, mais especificamente, com o sistema DogmatiX [17]. O sistema DogmatiX usa duas medidas de similaridade diferentes: uma simples *edit distance* normalizada, também utilizada pelo modelo aqui proposto, e uma *edit distance* ajustada por uma medida IDF (Inverse Document Frequency). A utilização de IDF garante que é atribuído um menor peso às palavras que ocorrem em muitos objectos, pois estas contribuem menos para os distinguir. Informação adicional sobre esta medida pode ser encontrada em [17].

Nestes testes comparativos, o modelo proposto, que utiliza redes Bayesianas, vai ser referido como RB. O modelo RB foi comparado com a abordagem do sistema DogmatiX, nas três bases de dados descritas na Secção 5.1.1, em termos de eficácia. Na base de dados da IMDB foram efectuadas as mesmas experiências descritas na Secção 5.3, mas apenas serão mostrados os resultados onde as diferenças se verificaram significativas, isto é, resultados da variação de dados em falta. Nas duas bases de dados do mundo real, o modelo RB foi comparado com duas variantes do sistema DogmatiX, uma que utiliza IDF e outra que não utiliza IDF. Como o modelo RB não utiliza IDF, é mais correcto comparar com a segunda variante. Os resultados das bases de dados IMDB+FilmDienst e FreeDB são apresentados nas Figuras 19 e 20, respectivamente. Ambos os algoritmos usaram um *threshold* de 0.6 na medida de similaridade para distinguir entre duplicados e não duplicados.

5.4.1 Testes na Base de Dados IMDB

Tal como pode ser observado na Figura 18, quando não existem dados em falta, ambos os sistemas apresentam um desempenho semelhante. Contudo, para o DogmatiX, a precisão desce abaixo de 90% quando a *recall* atinge cerca de 91%, ao passo que para o modelo RB

esta queda apenas acontece para valores de *recall* acima dos 99%. Quando 10% dos dados estão em falta, o modelo RB mostra melhorias significativas em relação ao DogmatiX. Apenas para um *recall* superior a 76% é que o DogmatiX apresenta valores de precisão mais altos. Adicionalmente, pode ser verificado que o modelo RB possui uma curva muito mais regular, indicando que este é mais eficaz em evitar falsos duplicados para os resultados com similaridades mais altas. Por outro lado, o DogmatiX possui uma maior resiliência ao aumento de dados em falta. Com 50% de dados em falta, apesar de ocorrer um decrescimento claro no *recall* final, os valores de precisão não sofrem uma grande diminuição, ao passo que para o modelo RB, a precisão cai rapidamente para valores de *recall* tão baixos como 17%. De notar que os resultados apresentados na Figura 18 apenas consideram a variante do DogmatiX que utiliza IDF. Apesar de não terem sido realizadas experiências com a variante do DogmatiX que não utiliza IDF, tudo leva a crer que a eficácia do DogmatiX neste conjunto de dados seria negativamente afectada nessa situação.



(a) sem dados em falta

(b) 10% de dados em falta

(c) 50% de dados em falta

Figura 18: Comparação entre o sistema DogmatiX e o modelo RB, variando quantidade de dados em falta.

5.4.2 Testes na Base de Dados IMDB + Film-dienst

Considerando os resultados da base de dados integrada IMDB+FilmDienst presentes na Figura 19, pode ser observado que o modelo RB é muito eficaz na detecção de duplicados nesta colecção. A precisão permanece acima dos 92% e apenas mostra uma queda significativa quando a *recall* atinge cerca de 80%. Nesta situação o DogmatiX, com a ajuda da IDF, consegue manter uma precisão semelhante apenas até 60% de *recall* mas, sem IDF, os resultados obtidos pelo DogmatiX são bastante mais fracos.

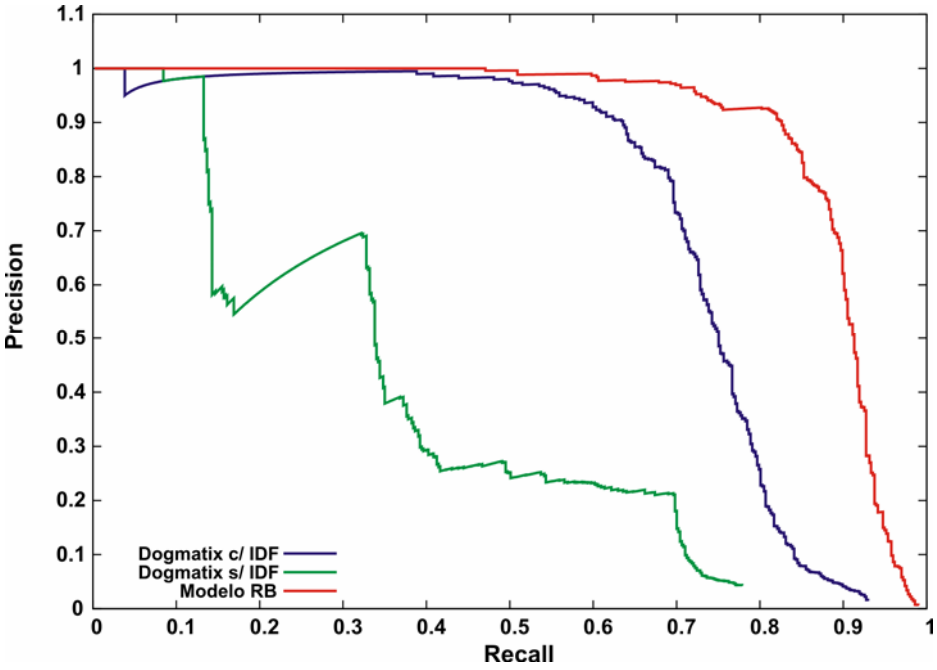


Figura 19: Valores de precisão e *recall* na base de dados IMDB+FilmDienst.

5.4.3 Testes na Base de Dados FreeDB

A suspeita de que o DogmatiX é penalizado com a não utilização de IDF foi reforçada pelos resultados obtidos na colecção de 10.000 CDs da FreeDB, apresentados na Figura 20, onde o DogmatiX usando IDF possui um desempenho bastante superior ao DogmatiX que não usa IDF. Basicamente, o DogmatiX com IDF apresenta os melhores resultados, sendo capaz de devolver 91% dos duplicados conhecidos. Em contraste, o *recall* mais alto atingido pelo modelo RB foi 79%. Por outro lado, o modelo RB apresenta resultados totais mais elevados para a precisão, sempre acima dos 90%, para níveis de *recall* entre 7% e 66%. O facto desta base de dados conter muitos valores indistintos para muitos dos campos nos objectos XML (por exemplo, “track1”,”track2”, etc, nos nomes das faixas dos cds ou “various artists” no nome dos

artistas), explica não só os falsos positivos entre os pares com similaridades mais altas, mas também porque é que uma medida de similaridade que integre IDF pode fornecer resultados mais precisos.

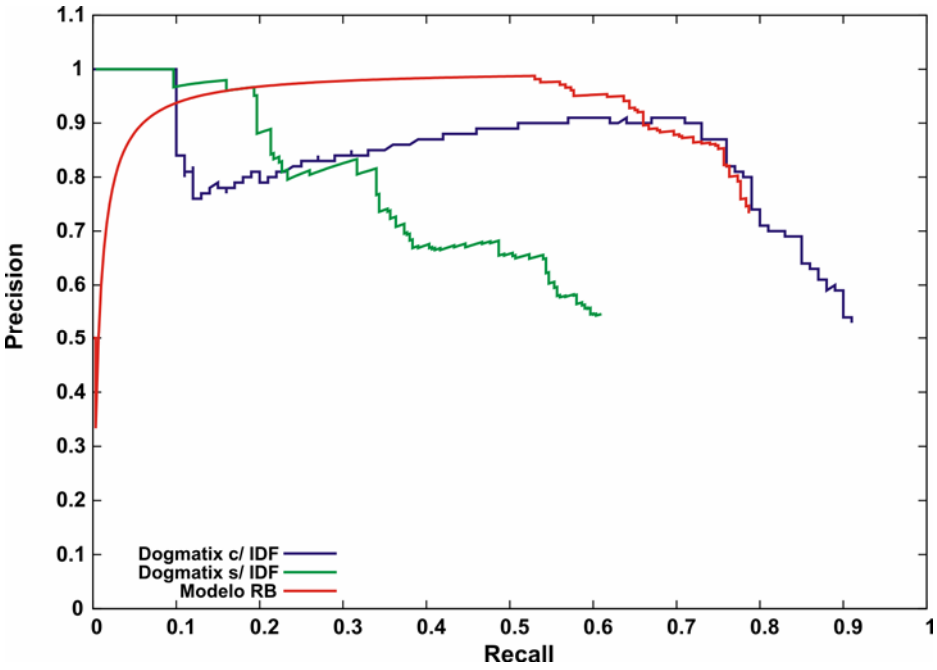


Figura 20: Valores de precisão e *recall* na base de dados FreeDB.

Ao longo destas experiências, constatou-se que a abordagem RB para medir a similaridade é muito eficaz na detecção de duplicados em dados XML. Contudo, esta pode ainda incorporar certas alterações, tais como o uso de IDF com vista a uma melhoria da eficácia.

5.5 Eficiência

Nesta secção é feita uma breve abordagem ao actual estado de eficiência do modelo RB. Apesar de não ser do foro deste trabalho existe, no entanto, algum interesse em fornecer informação sobre onde é dispendida a maior parte do tempo computacional do algoritmo, para tornar mais completa a compreensão do modelo apresentado. Para efeitos de simplicidade, é considerado que a execução do algoritmo tem apenas dois passos essenciais, a criação da rede Bayesiana e a sua avaliação. Considera-se que outras etapas do algoritmo, como o carregamento de parâmetros de configuração, são desprezáveis quando comparadas com os passos mencionados. Os testes foram realizados sobre uma base de dados contendo os primeiros 2000 objectos da base de dados FreeDB, utilizando a aplicação NetBeansProfiler

[58], e o tempo total de execução do algoritmo correspondeu a 80978988 milissegundos (cerca de 22.5 horas).

A criação da rede é concretizada através de dois métodos. O primeiro, que normaliza os níveis das duas árvores que representam os objectos XML a serem comparados, e um segundo que, depois da normalização, constrói a rede Bayesiana segundo a ordem definida para os elementos. É na avaliação da rede que é decidida a similaridade entre dois objectos, para a qual são utilizados dois métodos: a função de similaridade *edit distance* e um método que percorre os nós da rede segundo os parâmetros definidos no ficheiro de configuração. Um esquema da subdivisão aqui mencionada, bem como os tempos de execução dos métodos, estão ilustrados na Tabela 3 e na Figura 21.

CRIAR REDE 4%		**AVALIAR REDE** 96%	
Construir rede	**Normalizar níveis**	**Função similaridade**	**Percorrer rede**
3,9%	0,1%	85,4%	10,6%

Tabela 3: Distribuição dos tempos de execução do modelo RB (em percentagem).

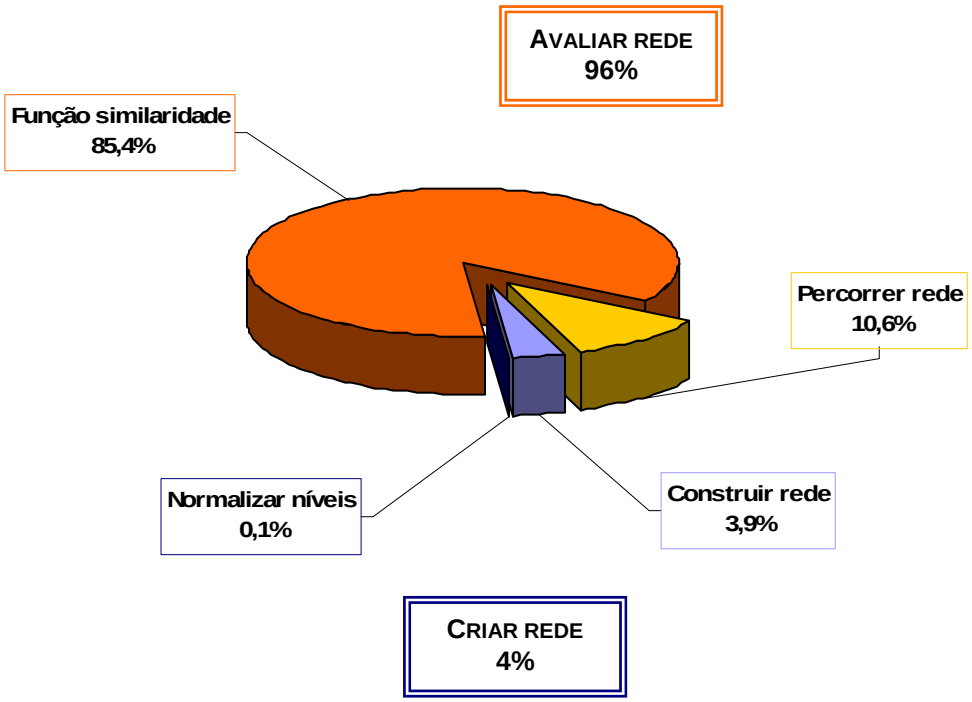


Figura 21: Tempos de execução dos métodos do modelo RB (em percentagem).

De facto, é na avaliação da rede que o algoritmo despende praticamente a totalidade do seu tempo computacional (96%). Como se pode observar pela Figura 21, esta desproporção de tempos despendidos é, essencialmente, da responsabilidade da função de similaridade, que consome 85.4% do tempo de execução do algoritmo. Intuitivamente, a solução para aumentar a eficiência do algoritmo poderia passar por reduzir o número de vezes que a função de similaridade é invocada. No caso da criação da rede, não se verificam grandes consumos de tempo, com 3.9% para a construção da rede e 0.1% para a normalização dos níveis. Por outro lado, percorrer a rede ocupa uma parcela significativa do tempo global de execução do algoritmo, com 10.6%. Este método poderia, eventualmente, ser optimizado recorrendo a heurísticas que permitissem, com base em alguns cálculos, limitar o número de nós percorridos na rede.

6 Conclusões

Este trabalho apresentou um novo método para a detecção de duplicados em XML. Através do uso de uma rede Bayesiana, o método proposto permite determinar de um modo preciso a probabilidade de dois objectos numa dada base de dados serem duplicados. O modelo Bayesiano é derivado a partir da estrutura dos objectos XML que são comparados, e todas as probabilidades são calculadas tendo em conta tanto os valores contidos nos objectos como a sua estrutura interna.

O modelo aqui proposto fornece não só uma base formal para o procedimento de detecção de duplicados, como requer pouca parametrização. Na verdade, todos os resultados aqui apresentados, usando três bases de dados distintas, foram obtidos com a mesma configuração, tal como descrito na Secção 5. Algumas experiências apenas foram necessárias para escolher três parâmetros, a probabilidade por omissão, as funções de combinação para múltiplos nós do mesmo tipo e o *threshold* de similaridade. Os testes tinham apenas o objectivo de avaliar o desempenho do algoritmo sob diferentes condições. Adicionalmente, o modelo também fornece uma grande flexibilidade na sua configuração, permitindo o uso de medidas de similaridade diferentes, para diferentes valores de campos, e diferentes probabilidades condicionais para combinar as probabilidades de similaridade dos elementos XML.

Os testes realizados, tanto nas bases de dados do mundo real como nas artificiais, mostram que o modelo apresentado é capaz de atingir resultados bastante precisos, com valores elevados de precisão e *recall* em todos os casos. Para uma validação posterior destes resultados, a abordagem proposta foi comparada com um sistema de detecção de duplicados em XML que representa o estado da arte neste domínio, o sistema DogmatiX. Nesta comparação, o modelo das redes Bayesianas mostrou de forma consistente melhores resultados do que o DogmatiX quando este não empregava a medida IDF, que aumenta significativamente a sua eficácia.

7 Trabalho Futuro

Os bons resultados alcançados através das experiências realizadas e a flexibilidade fornecida pela solução proposta deixam em aberto espaço para bastante trabalho futuro. Entre outras tarefas, seria de grande interesse estudar o uso de medidas de similaridade dependentes do domínio nas probabilidades *à priori*, estender o algoritmo de construção do modelo de redes Bayesianas para comparar objectos XML com diferentes estruturas, testar o modelo com mais bases de dados e com diferentes configurações da rede, e aplicar métodos de *machine learning*, baseados nos dados existentes, para derivar as probabilidades condicionais. Tal como verificado nos testes comparativos com o DogmatiX, o uso de uma medida IDF pode, potencialmente, melhorar bastante a qualidade dos duplicados encontrados pelo algoritmo e deverá também ser testada no futuro. Outra questão com interesse em considerar é a da escalabilidade, tanto no espaço como no tempo. Para melhorar o tempo de execução do algoritmo poderiam ainda ser usadas heurísticas para evitar o cálculo de algumas probabilidades. No entanto, escalar o algoritmo para tratar grandes quantidades de dados e, também, redes Bayesianas de grandes dimensões, requer o uso de memória externa e deve ser alvo de estudos mais detalhados.

Além das melhorias e extensões descritas com vista a melhorar tanto a funcionalidade como o desempenho do algoritmo, existiria uma grande vantagem em incorporar nesta solução a capacidade de fusão de objectos duplicados num único objecto. Esta capacidade conferiria ao algoritmo uma maior utilidade prática e iria mais ao encontro do que um utilizador final pretenda duma ferramenta deste tipo, pois permitiria, para além da detecção dos duplicados presentes em determinada base de dados XML, a eliminação dos mesmos. Seria também interessante nesta etapa da fusão, à semelhança do que acontece para a detecção de duplicados, que o utilizador tivesse ao seu dispor um meio que lhe permitisse efectuar uma configuração mais completa e que ditasse, entre outras coisas, quais os campos dos dois objectos que devem prevalecer no objecto resultante da fusão. Adicionalmente, a fusão de objectos poderia ser utilizada para fazer convergir objectos com diferentes estruturas numa estrutura singular.

8 Bibliografia

1. H. Galhardas, D. Florescu, D. Shasha, E. Simon and C. Saita. Declarative data cleaning: Language, model, and algorithms. In *Proceedings of the 27th International Conference on Very Large Databases (VLDB-2001)*, pp. 371–380. Rome, Italy, 2001.

2. E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23: 3-13, 2000.

3. V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *Proceedings of 27th International Conference on Very Large Databases (VLDB-2001).*

4. W. E. Winkler. Data cleaning methods. In *Proceedings of the KDD-2003 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*), pp. 1–6. Washington, DC, 2003.

5. W. E. Winkler. The state of record linkage and current research problems. *Technical report, U. S. Bureau of the Census*, 1999.

6. M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *International Conference on Management of Data (SIGMOD)*, pp. 127–138. San Jose, CA, May 1995.

7. I. Bhattacharya and L. Getoor. Relational clustering for multi-type entity resolution. *Workshop on Multi-Relational Data Mining (MRDM)*, 2005.

8. X. Dong, A. Halevy and J. Madhavan. Reference reconciliation in complex information spaces. In *International Conference on the Management of Data (SIGMOD)*. Baltimore, MD, 2005.

9. W. W. Cohen, H. Kautz and D. McAllester. Hardening soft information sources. In *SIGKDD*, 2000.

10. A. K. McCallum, K. Nigam and L. H. Ungar. Eficient Clustering of High-Dimensional Data Sets with Application to Reference Matching. In *SIGKDD*, 2000.

11. W. W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*. Edmonton, Alberta, 2002.

12. S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*. Edmonton, Alberta, 2002.

13. S. Tejada, C. Knoblock and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *SIGKDD*, 2002.

14. A. McCallum and B. Wellner. Toward conditional models of identity uncertainty with application to proper noun coreference. In *IIWEB*, 2003.

15. Z. Chen, D. V. Kalashnikov and S. Mehrotra. Exploiting relationships for object consolidation. In *SIGMOD-2005 Workshop on Information Quality in Information Systems*. Baltimore, MD, 2005.

16. A. E. Monge and C. P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, pp. 23–29. Tuscon, AZ, 1997.

17. M. Weis and F. Naumann. Dogmatix tracks down duplicates in xml. In *Conference on the Management of Data (SIGMOD)*, pp. 431–442. Baltimore, MD, 2005.

18. H. Turtle and W. B. Croft. Inference networks for document retrieval. In *Proceedings of the 13th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1–24. Brussels, Belgium, Sept. 1990.

19. B. Ribeiro-Neto and R. Muntz. A belief network model for IR. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 253–260. Zurich, Switzerland, Aug. 1996.

20. S. Acid, L. M. de Campos, J. M. Fernandez-Luna and J. F. Huete. An information retrieval model based on simple bayesian networks. *International Journal of Intelligent Systems*, 18(2):251–265, Jan. 2003.

21. D. Haines and W. B. Croft. Relevance feedback and inference networks. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2–11. Pittsburgh, PA, USA, June 1993.

22. S. T. Dumais, J. Platt, D. Heckerman and M. Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of the 7th International Conference on Information and Knowledge Management CIKM 98*, pp. 148–155. Bethesda, MD, USA, Nov. 1998.

23. H. Newcombe, J. Kennedy, S. Axford and A. James. Automatic linkage of vital records. *Science*, 130(3381):954–959, 1959.

24. I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 1969.

25. M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 39–48. Washington, DC, 2003.

26. A. Doan, Y. Lu, Y. Lee and J. Han. Object matching for information integration: A profiler-based approach. *IEEE Intelligent Systems*, pp. 54-59, 2003.

27. P. Singla and P. Domingos. Object identification with attribute-mediated dependences. In *Conference on Principals and Practice of Knowledge Discovery in Databases (PKDD)*, pp. 297–308. Porto, Portugal, 2005.

28. I. Bhattacharya and L. Getoor. A latent dirichlet model for unsupervised entity resolution. In *Conference on Data Mining (SDM)*. Bethesda, MD, 2006.

29. D. V. Kalashnikov and S. Mehrotra. Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Transactions on Database Systems (TODS)*, 31(2):716–767, 2006.

30. X. Yin, J. Han and P. S. Yu. LinkClus: Efficient clustering via heterogeneous semantic links. In *Conference on Very Large Databases (VLDB)*, pp. 427–438. Seoul, Korea, 2006.

31. D. Lee, B.-W. On, J. Kang and S. Park. Effective and scalable solutions for mixed and split citation problems in digital libraries. In *SIGMOD-2005 Workshop on Information Quality in Information Systems*. Baltimore, MD, 2005.

32. S. Chaudhuri, V. Ganti and R. Motwani. Robust identification of fuzzy duplicates. In *Proceedings of the International Conference on Data Engineering (ICDE)*. Tokyo, Japan, 2005.

33. L. Jin, C. Li and S. Mehrotra. Efficient record linkage in large data sets. In *Conference on Database Systems for Advanced Applications (DASFAA)*. Kyoto, Japan, 2003.

34. R. Ananthakrishna, S. Chaudhuri and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *Conference on Very Large Databases (VLDB)*, pp. 586–597. Hong Kong, China, 2002.

35. S. Puhlmann, M. Weis and F. Naumann. Xml duplicate detection using sorted neigborhoods. In *Conference on Extending Database Technology (EDBT)*, pp. 773–791. Munich, Germany, 2006.

36. D. Milano, M. Scannapieco and T. Catarci. Structure aware xml object identification. In *VLDB Workshop on Clean Databases (CleanDB)*. Seoul, Korea, 2006.

37. M. Weis and F. Naumann. Duplicate detection in xml. In *SIGMOD Workshop on Information Quality in Information Systems (IQIS)*, pp. 10–19. Paris, France, 2004.

38. M. Weis and F. Naumann. Detecting duplicates in complex xml data. In *International Conference on Data Engineering (ICDE)*. Atlanta, GA, 2006.

39. I. Bhattachary, L. Getoor and L. Licamele. Query-time entity resolution (poster). In *Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 529–534. Philadelphia, PA, 2006.

40. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1st edition, 1999.

41. N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

42. E. S. Ristad and P. N. Yianilos. Learning string edit distance. *IEEE PAMI*, 20(5), 1998.

43. C. Faloutsos and K.-I. Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In M. J. Carey and D. A. Schneider, editors, *SIGMOD*, pp. 163–174, 1995.

44. M. A. Hernández and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1): 9–37, 1998.

45. M. Weis and F. Naumann. Technical Report Nr. HU-IB-206, July 2006.

46. http://www.w3.org/XML

47. http://www.w3.org/html

48. M. Weis. Fuzzy duplicate detection on XML. In *VLDB PhD Workshop*. Trondheim, Norway, 2005.

49. T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. *Introduction to Algorithms,* 2nd edition, MIT Press, 2001.

50. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of plausible inference*. Morgan Kaufmann Publishers, 2nd edition, 1988.

51. http://java.sun.com

52. http://www.w3.org/DOM

53. J. Pearl and S. Russell. Bayesian Networks. In Michael A. Arbib, Ed., *The Handbook of Brain Theory and Neural Networks*, 2nd edition, MIT Press, 2003.

54. http://www.hpi.uni-potsdam.de/naumann/projekte/dirtyxml

55. http://www.imdb.com

56. http://film-dienst.kim-info.de

57. http://www.freedb.org

58. http://profiler.netbeans.org