Instituto de Matemática e Estatística
Universidade de São Paulo

Trabalho de Formatura Supervisionado: Identificador de Plágio para o Adessowiki

Lucas Ikeda França

Orientador: Prof. Roberto Hirata Jr.

16 de fevereiro de 2012

Sumário

Parte I

Parte Técnica

Capítulo 1

Introdução

O plágio é uma questão polêmica que atinge diversos segmentos da sociedade, do artístico ao científico. Com o avanço nas tecnologias de compartilhamento de informação, a prática foi facilitada e sua ocorrência tornou-se mais comum.

É um problema ético, e há um recente exemplo [14] de sua gravidade: o então ministro de defesa da Alemanha, Karl-Theodor zu Guttenberg, renunciou ao cargo após um escândalo causado pela descoberta de plágio em sua tese de doutorado.

No meio acadêmico, a atitude é condenada pelos regimentos internos das instituições de ensino, e há um esforço para identificar e inibir a prática de plágio. Esta preocupação agrava-se em ambientes de educação a distância, onde o controle sobre a maneira que as tarefas são executadas pelos alunos é muito menor do que em situações presenciais. O grande volume de tarefas a ser analisado faz com que a detecção manual nesses casos seja inviável, portanto soluções automatizadas são necessárias.

Embora já existam tais ferramentas disponíveis gratuitamente, não havia até o início deste trabalho qualquer integração delas ao ambiente de ensino Adessowiki, descrito no capítulo a seguir.

O objetivo deste trabalho é fornecer ao Adessowiki uma ferramenta para identificação de plágio em seus conteúdos, tanto código-fonte quanto texto em linguagem natural, a fim de detectar os casos suspeitos: estes serão posteriormente encaminhados para uma análise humana. Espera-se que a existência desse tipo de aparato também desencoraje tentativas de plágio por parte dos alunos, garantindo por consequência um melhor nível de aprendizado.

Nos próximos capítulos serão cobertos os conceitos necessários para a compreensão do funcionamento da ferramenta, bem como um relato do desenvolvimento do trabalho, os resultados obtidos, e as conclusões.

Capítulo 2

Conceitos

2.1 Educação a Distância

É definido em [13] como o processo de aprendizagem onde professores e alunos não estão presentes no mesmo local, ou, inclusive, ao mesmo tempo. A interação entre as partes ocorre por meio de tecnologias de comunicação.

Apesar de não ser uma ideia nova[15], passando ao longo do século XX por meios como correspondência, rádio e televisão, com a popularização da Internet no final do período tornou-se uma forte tendência no meio educacional, pois apesar do distanciamento físico entre as partes envolvidas há um grande ganho no dinamismo da troca de experiências, esclarecimento de dúvidas e obtenção de resultados. Isto ocorre pois o aprendizado não fica restrito ao horário ou local de aula.

É um conceito com potencial para democratizar o acesso à informação e ao conhecimento, uma vez que o acesso à Internet esteja ao alcance de todos[15].

2.2 Plágio

Uma definição[7] comum para plágio é a de apropriação de ideias de terceiros, isto é, um indivíduo se utilizar do trabalho intelectual realizado por outro, sem dar o devido crédito ao autor original. De acordo com [16], substituições sobre um texto que mantenham maior parte, ou a estrutura do original, também caracterizam plágio.

Ainda de acordo com [16], uma simples citação das fontes pode ser suficiente para evitar a maior parte dos casos de plágio.

A prática é antiga, porém sua ocorrência vem crescendo nos últimos anos: uma pesquisa[2] feita com reitores de faculdades mostrou que 55% deles afirmaram que a incidência de plágio aumentou nos

últimos 10 anos, e 89% destes acreditam que os computadores e a Internet foram os principais fatores que causaram esse crescimento.

Embora sua classificação seja subjetiva, uma vez que algo considerado plágio por alguns possa não ser para outros, é necessário estabelecer um limite de tolerância. Tal limite é um importante parâmetro para as técnicas de identificação de plágio. Porém, a escolha deste limite não é tão bem definida, não havendo uma regra geral que seja aplicável a todo e qualquer cenário.

2.3 Similaridade

Refere-se ao grau de semelhança entre dois documentos. É importante classificar a similaridade em dois níveis: global e local. O primeiro trata de características mais gerais de um documento, como o tema ou tamanho, enquanto o segundo compreende propriedades mais específicas, como sequências de palavras em comum.

Em uma situação de tarefa, onde todos os alunos devem escrever programas a partir da mesma especificação, ou redigir textos sobre um mesmo assunto, é natural que haja similaridade global entre os documentos[7]. Já similaridade local pode ser um indicativo de suspeita de plágio, pois sua ocorrência na situação abordada neste projeto é menos provável, e deve ser analisada cuidadosamente. Este será o tipo de similaridade explorado neste trabalho.

2.4 *Winnowing*

Este é o algoritmo para detecção de plágio utilizado neste projeto. Também é o principal algoritmo utilizado pela ferramenta Moss, descrita no próximo capítulo. A seguir será apresentada uma visão geral do algoritmo, conforme explicado em [18].

O termo *winnow* pode ser traduzido do inglês como peneirar, separar ou ainda selecionar[19]. Isto nos fornece a ideia central do algoritmo: separar valores-chave de um texto (*fingerprints*), para então compará-los com valores selecionados de outros textos, em busca de trechos similares comuns. O resultado é um método eficiente para comparação de textos em busca de similaridade local.

Inicialmente o texto original passa por um pré-processamento.

Em seguida são extraídas subsequências do texto pré-processado, denominadas k-gramas. Cada k-grama é uma subsequência de k caracteres seguidos.

Então, para cada k-grama, é calculado um valor de *hash*, a partir de uma função com baixa probabilidade de colisão (isto é, entradas diferentes devem gerar resultados diferentes). Estes valores calculados são denominados *fingerprints* do texto.

Pode-se então comparar os *fingerprints* de dois textos, e em caso de valores iguais, é muito provável que as entradas compartilhem de trechos em comum. Para recuperar estes trechos, os *fingerprints* devem vir acompanhados da respectiva posição do k-grama gerador no texto original.

O algoritmo apresenta as seguintes propriedades, que motivaram sua escolha para este trabalho:

- Insensibilidade a espaços em branco, pontuação, letras maiúsculas ou minúsculas. Qualquer tipo de inserção de espaços ou pontuação, ou a substituição de maiúsculas por minúsculas (e vice-versa) não afeta a comparação dos documentos, pois o pré-processamento normaliza o texto de forma a eliminar essas diferenças.

- Independência da posição de um trecho de texto. A reordenação ou remoção de trechos (suficientemente grandes) do texto original não afetam a correspondência dos *fingerprints*.

- Supressão de ruído. Correspondência de sequências de caracteres curtas não são interessantes, pois não podem ser usadas como indicativo de suspeita de plágio, e devem ser ignoradas.

Além disso, é o algoritmo utilizado pela ferramenta Moss, um dos mais populares identificadores de plágio para código-fonte, o que atesta sua eficácia.

Capítulo 3

Tecnologias Utilizadas

3.1 Adessowiki

É um ambiente de educação a distância online[10] e, como o termo *wiki*[7] tipicamente sugere, colaborativo. Foi desenvolvido em uma parceria entre a Faculdade de Engenharia Elétrica e de Computação da Universidade de Campinas e o Centro de Tecnologia da Informação Renato Archer. Os detalhes a seguir sobre o ambiente foram retirados de [9].

Um de seus maiores diferenciais está na possibilidade do usuário incluir nas páginas não apenas texto em linguagem natural, mas também código-fonte nas linguagens Python, C e C++, ou seja, não é necessário um interpretador ou compilador instalados em sua máquina: bastam um navegador e acesso à Internet. O código editado em uma página do *wiki* é executado em um *sandbox* no servidor, e os resultados são exibidos no momento da visualização da página. Esta facilidade é especialmente bem-vinda para alunos de cursos de Introdução à Computação, que muitas vezes tem dificuldade em instalar as ferramentas necessárias para programar, e também para alunos iniciando seus estudos em áreas mais específicas, como Visão Computacional e Processamento de Imagens, pois tem, logo de início, muitas das bibliotecas e funções mais utilizadas à sua disposição. Além disso, permite que a programação seja feita a partir de dispositivos móveis ou computadores com baixo poder de processamento, ligados à Internet.

As figuras a seguir mostram a edição e a visualização de um trecho de código em uma página Adessowiki:

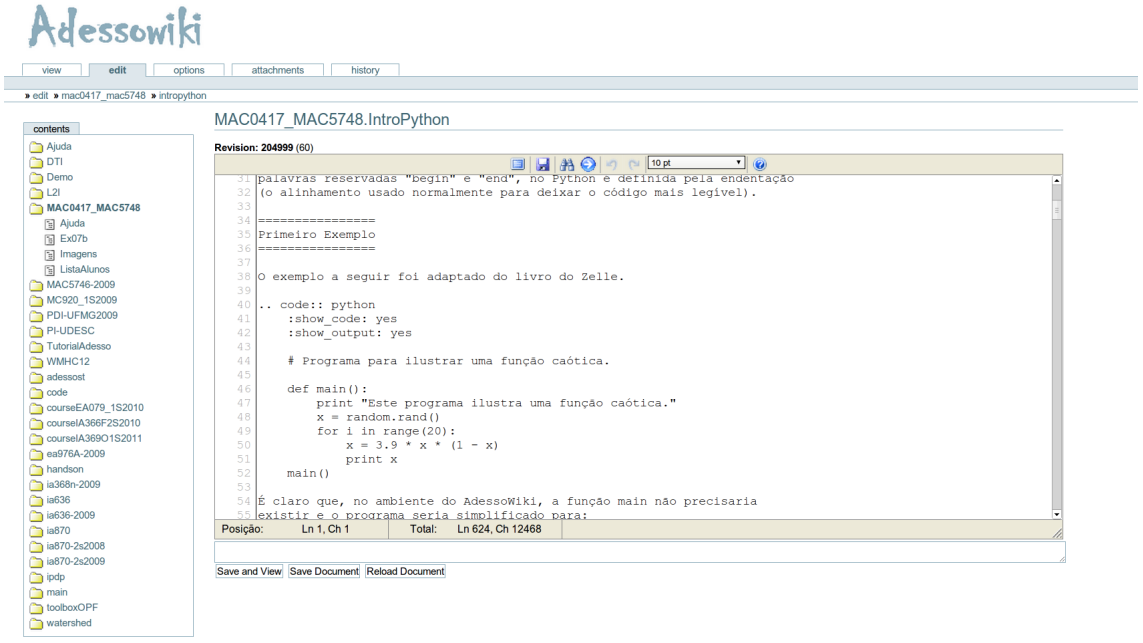

Figura 3.1 – *Edição de código Python por meio do navegador (fonte: [10])*

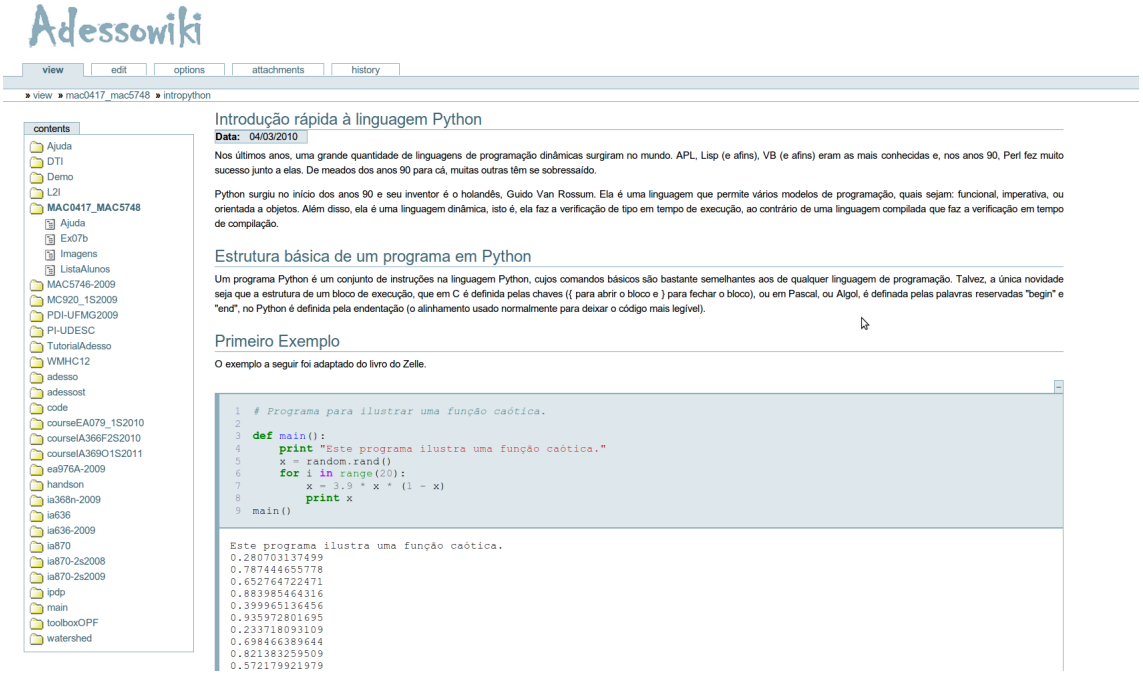

Figura 3.2 – *Visualização do código e sua saída no navegador (fonte: [10])*

Há ainda o aspecto colaborativo, que facilita situações de pesquisa, pois é possível executar diferentes algoritmos sobre determinada entrada e comparar seus resultados diretamente com outras implementações de maneira fácil.

É um sistema escrito na linguagem de programação Python, utilizando o *framework* Django[5]. Sua arquitetura está organizada conforme a imagem a seguir:

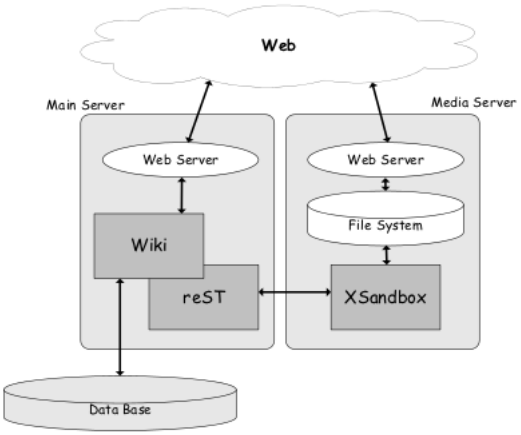

Figura 3.3 *– Organização do sistema nos servidores do Adessowiki (fonte: [9])*

De acordo com [10], o ambiente foi utilizado nos últimos anos principalmente em disciplinas da área de Visão Computacional e Processamento de Imagens, ministradas em renomadas instituições de ensino superior brasileiras, como Universidade de São Paulo, Universidade de Campinas, Universidade Federal de Minas Gerais e Universidade do Estado de Santa Catarina.

3.2 Moss

Moss[1] é uma ferramenta de identificação de plágio gratuita, de código fechado, e muito popular entre professores que ministram cursos de computação ao redor do mundo. Suporta um grande número de linguagens, entre elas Python, C e C++, porém não é compatível com textos em linguagem natural.

Tem como base a técnica de *winnowing*, descrita no capítulo anterior, porém conta com um pré-processamento adaptado para cada tipo de linguagem de programação suportada: partes não interessantes da entrada, como comentários, são ignoradas; nomes funções e variáveis são trocados pois de outra maneira, substituições simples destes valores causariam a não detecção de trechos suspeitos.

É um serviço online disponível desde 1997, hospedado pela Stanford University, e requer que o usuário faça um cadastro para poder enviar código-fonte para análise, que é feita no servidor. Os resultados são exibidos em uma página gerada para cada rodada de execução, a qual fica disponível para consulta apenas por um determinado tempo, sendo removida após este período.

Foi adotada neste projeto como uma estratégia simplificadora para tratar os trechos de código-fonte nos conteúdos do Adessowiki. O pré-processamento deste tipo de conteúdo, para posterior aplicação do algoritmo do *winnowing*, é difícil, pois é preciso normalizar com nomes de variáveis, nomes de funções, além das diversas particularidades de cada linguagem suportada (por exemplo, a indentação em Python não é desprezível, mas em C++ sim).

3.3 Python

Python[6] é uma popular linguagem de programação de alto nível e código aberto, criada por Guido Van Rossum em 1991. É conhecida por sua simplicidade e flexibilidade, sendo utilizada por programadores de todos níveis de experiência, em uma grande variedade de aplicações.

Algumas de suas vantagens que motivaram sua escolha para este trabalho são: sintaxe clara, portabilidade entre sistemas operacionais, e grande quantidade de bibliotecas disponíveis para diversas finalidades. Há ainda um outro ponto positivo: trata-se da mesma linguagem em que o Adessowiki foi desenvolvido, o que possibilita uma futura integração com o sistema – a ferramenta desenvolvida aqui poderia ser adaptada para ser executada diretamente no servidor do Adessowki.

Uma desvantagem está no fato de ser interpretada, o que a torna mais lenta em determinadas tarefas quando comparada a algumas linguagens compiladas; entretanto, para os requisitos de performance deste trabalho, esta diferença não é grande o suficiente a ponto de justificar seu abandono, dadas as vantagens citadas anteriormente.

Foi utilizada a versão 2.7.1.

Capítulo 4

Atividades Realizadas

4.1 Estudo de técnicas de detecção de plágio

Antes de decidir pela técnica de *winnowing*, foram pesquisadas outras propostas para identificação de plágio. As seguintes alternativas foram testadas:

- Métricas[8] – são comparadas características da entrada como quantidade de linhas, laços de repetição e variáveis no caso de código-fonte; e número de parágrafos, e palavras para o caso de linguagem natural. Os dados obtidos para cada entrada são comparados com outros para buscar similaridade.
- Árvore de Sintaxe Abstrata[7] – técnica voltada para detecção em código-fonte; cria-se uma estrutura de dados a partir da entrada, que é comparada com árvores criadas para outras instâncias.
- Modelo de Espaço Vetorial – técnica mais sofisticada, apresentada em [17], consiste na geração de um vetor de características para cada texto; a semelhança entre duas entradas é estimada por meio da comparação dos respectivos vetores, utilizando medidas como seu produto escalar.

Porém, todas as opções acima são particularmente boas para detectar similaridade global, mas não local. Isto faz com que elas não sejam a escolha adequada para o escopo deste trabalho – em um ambiente de educação a distância é esperada similaridade global, conforme explicado anteriormente.

A partir daí, deu-se início a um estudo mais aprofundado do *winnowing*, conforme enunciado em [18], para que a técnica pudesse ser implementada localmente para textos em linguagem natural.

4.2 Desenvolvimento do *parser*

O primeiro passo para o desenvolvimento da ferramenta proposta foi criar um *parser*, que transformasse os conteúdos de páginas do Adessowiki em documentos puros, isto é, contendo apenas um tipo de dado e sem qualquer vestígio de marcadores do Adessowiki. Tais documentos são salvos em um diretório temporário.

A figura a seguir ilustra o funcionamento do parser:

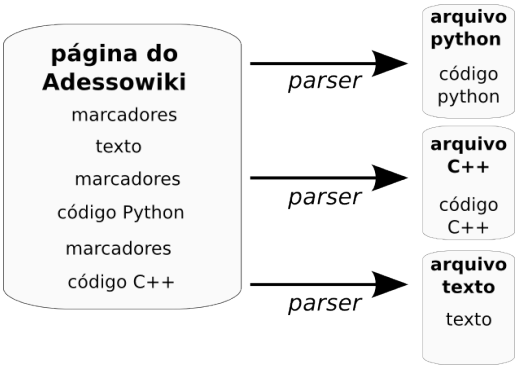

Figura 4.1 *– Aplicação do parser em uma página do Adessowiki*

Uma vez separados os conteúdos, pode-se dar prosseguimento à análise em busca de similaridades entre os documentos, comparando apenas os tipos de conteúdo pertinentes, isto é, texto em linguagem natural não será comparado com código-fonte, e código-fonte em uma determinada linguagem não será comparado com código-fonte em outra linguagem. Desta maneira economiza-se o tempo de comparações que não retornariam qualquer informação útil para o propósito deste trabalho.

4.3 Implementação do *winnowing*

Trata-se da parte mais importante da ferramenta: o conceito da técnica foi apresentado anteriormente, e agora será explicado em detalhes. Vale lembrar que o algoritmo foi implementado localmente para lidar com textos em linguagem natural. O código-fonte das principais funções implementadas encontra-se no Apêndice.

O primeiro passo é o pré-processamento, que visa eliminar características desinteressantes da entrada como espaçamento, pontuação e capitalização. A presença dessas características pode influenciar na detecção de similaridade de baixo nível, pois modificam a correspondência entre dois textos, sem adicionar significado (sob a ótica deste trabalho) aos mesmos.

Simultaneamente, é feito um mapeamento entre as posições do texto original e do pré-processado. Isto é importante para a posterior recuperação dos trechos suspeitos no texto original, pois como a

comparação acontece sobre os textos pré-processados, as correspondências encontradas se referem aos mesmos, e estes são difícil compreensão dadas as operações feitas no pré-processamento.

Finalmente o algoritmo de *winnowing* propriamente dito pode ser aplicado.

Dado um texto pré-processado, são obtidos os seus k-gramas, que são todas as subsequências de k caracteres neste texto. Então, para cada k-grama, calcula-se um valor de *hash*. Para isso foi utilizada a função md5, pois no cenário deste projeto a probabilidade de repetição (colisão) dos valores fornecidos por ela é baixa.

São montadas janelas de comprimento w contendo esses valores de *hash*, que são subsequências de tamanho w, e seleciona-se alguns desses valores da seguinte maneira:

Em uma janela, escolhe-se o menor valor possível. Em caso de empate, escolhe-se o valor mais à direita. Juntamente com o valor selecionado guarda-se a posição em que ele ocorre no texto pré-processado.

Ao passar por todas as janelas, tem-se um conjunto de valores, que são denominados *fingerprints* do documento.

Para comparar dois textos, basta comparar seus *fingerprints*. Em caso de correspondência, levanta-se a suspeita de plágio, pois eles compartilham de uma subsequência suficientemente longa para não ser considerada ruído. O limite deste ruído está relacionado ao parâmetro k. A escolha acertada deste valor é fundamental: se for muito alto, casos que são plágio de fato irão passar desapercebidos (falsos negativos); e se for muito baixo, casos que definitivamente não são plágio serão identificados como suspeitos (falsos positivos). Nenhuma das situações é desejável.

Com os dados posicionais armazenados junto com os *fingerprints*, pode-se recuperar facilmente os trechos correspondentes a cada um deles no texto original: uma simples consulta ao mapeamento das posições feito durante o pré-processamento devolve o intervalo ao qual um *fingerprint* corresponde no texto original.

4.4 Integração com o Moss

4.4.1 Motivos

Embora a técnica do *winnowing* seja aplicável a qualquer tipo de texto, inclusive código-fonte, o pré-processamento exigido por este tipo de conteúdo é complicado, e é específico para cada linguagem de programação, o que impossibilita uma solução genérica.

Ainda que tenham sido estudadas maneiras de realizar este passo de pré-processamento, como o uso de árvores sintáticas, as tentativas foram abandonadas por não apresentar bons resultados, e tomar

tempo de desenvolvimento de outras características importantes da ferramenta.

O serviço do Moss suporta atualmente as linguagens presentes nas páginas do Adessowiki, e utiliza a mesma técnica, a menos do pré-processamento e parâmetros, implementada localmente para detecção de plágio. Estes fatores fazem do Moss o candidato ideal para a integração com a ferramenta desenvolvida neste trabalho.

4.4.2 Detalhes

O envio dos dados para o servidor do Moss é feito através de um *script* Perl, disponível para *download* na página do serviço[1].

Este script é invocado a partir da função principal da ferramenta, e seu resultado, um endereço URL, e finalmente, um *parser* interpreta os resultados disponíveis nesse endereço, a fim de agregá-los ao relatório a ser gerado.

4.5 Geração de relatórios

Os resultados das comparações entre as páginas, com os casos suspeitos de plágio, são exibidos como um relatório em uma página no próprio Adessowiki, cujo endereço é passado como parâmetro para o programa no momento de sua chamada.

As suspeitas são exibidas em uma lista, em que cada elemento corresponde ao seguinte: O par de entradas que levantou suspeita, e o trecho em que foi detectada alta similaridade. No caso de conteúdo em linguagem natural, o trecho é exibido diretamente no relatório; caso o conteúdo seja do tipo código-fonte, é exibido um *link* para a página de resultados do Moss, contendo o trecho suspeito destacado.

Para auxiliar a visualização geral dos resultados é exibido um grafo, onde cada vértice representa um tipo de conteúdo em uma página suspeita do Adessowiki, e cada aresta representa a existência de similaridade de baixo nível entre os vértices que ela conecta. A figura a seguir ilustra um exemplo:

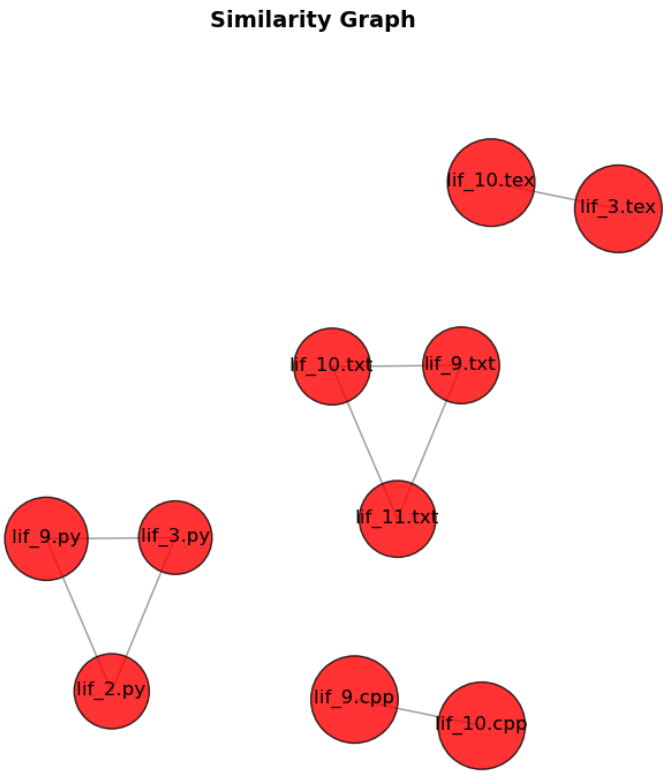

Figura 4.2 – *Exemplo de grafo de similaridade presente em um relatório.*

4.6 Testes

Para avaliar os resultados da ferramenta desenvolvida neste trabalho, foi feita uma comparação com o JPlag[11], um renomado software para identificação de plágio, compatível tanto com textos em liguagem natural quanto código-fonte (Java, C, C++, C e Scheme). Seu uso, apesar de gratuito, é restrito a usuários registrados. Assim como o Moss, é um serviço online.

As figuras a seguir ilustram os resultados das execuções de ambas ferramentas para uma mesma entrada:

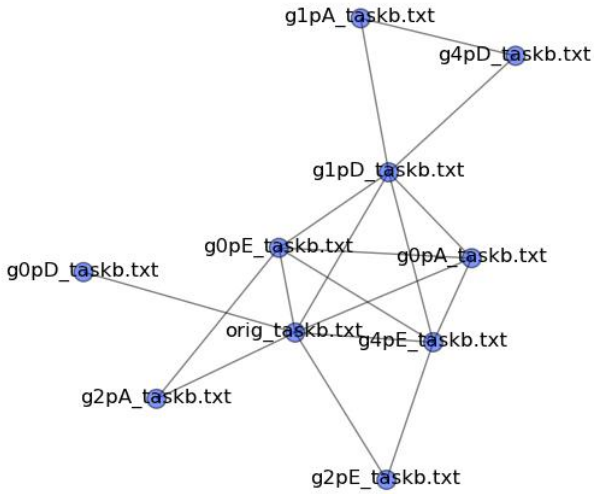

Figura 4.3 – *Grafo criado pelo identificador de plágio*

Matches sorted by average similarity (What is this?):

orig_taskb.txt	->	g0pA_taskb.txt (51.2%)	g4pE_taskb.txt (45.6%)	g1pD_taskb.txt (20.4%)	g0pE_taskb.txt (18.5%)	g2pE_taskb.txt (13.6%)	g0pD_taskb.txt (12.9%)	g2pA_taskb.txt (5.5%)	g3pA_taskb.txt (2.4%)
g1pD_taskb.txt	->	g4pD_taskb.txt (50.5%)	g0pA_taskb.txt (33.8%)	g0pE_taskb.txt (22.9%)	g0pD_taskb.txt (11.5%)	g4pE_taskb.txt (5.6%)	g1pA_taskb.txt (4.4%)	g2pA_taskb.txt (4.0%)	
g0pE_taskb.txt	->	g0pA_taskb.txt (38.4%)	g0pD_taskb.txt (24.5%)	g4pE_taskb.txt (17.4%)	g2pE_taskb.txt (7.0%)	g2pA_taskb.txt (6.2%)			
g0pA_taskb.txt	->	g4pE_taskb.txt (19.3%)	g0pD_taskb.txt (14.3%)	g2pA_taskb.txt (9.2%)					
g4pE_taskb.txt	->	g2pE_taskb.txt (10.8%)	g0pD_taskb.txt (5.5%)	g3pA_taskb.txt (3.9%)					
g2pE_taskb.txt	->	g0pD_taskb.txt (8.1%)							
g4pD_taskb.txt	->	g1pA_taskb.txt (4.8%)							

Figura 4.4 – *Resultado do JPlag, exibido como listas ligadas*

Os testes foram divididos de acordo com o tipo de conteúdo a ser analisado.

4.6.1 Linguagem Natural

Para testar este tipo de conteúdo, foram usadas entradas pertencentes a um *corpus*[3] criado especificamente para estudos de identificação de plágio. A principal vantagem no uso deste tipo de conteúdo consiste em saber *a priori* quais entradas são plágio ou não. O conjunto é formado por 100 textos curtos (comprimento médio de 208 palavras), sobre 5 temas que possuem artigos correspondentes na Wikipedia[7], e escritos por 19 indivíduos instruídos a produzir esses textos respeitando os graus de similaridade apresentados a seguir, em nível decrescente:

(I) cópia direta de trechos do artigo correspondente

(II) uso de trechos do artigo, porém com substituições de alguns termos por sinônimos, e alteração da estrutura gramatical de algumas frases

(III) uso do artigo como base, com forte reestruturação do texto original

(IV) sem consulta ao artigo, mas com a permissão de utilizar outras referências fornecidas além dos conhecimentos próprios

Os graus (I), (II) e (III) foram classificados como plágio, e o grau (IV) como não-plágio.

Os trechos abaixo foram retirados do *corpus* e ilustram um artigo e exemplos de diferentes graus de similaridade:

Vector space model (or term vector model) is an algebraic model for representing text documents (and any objects, in general) as vectors of identifiers, such as, for example, index terms. It is used in information filtering, information retrieval, indexing and relevancy rankings. Its first use was in the SMART Information Retrieval System. A document is represented as a vector. Each dimension corresponds to a separate term. If a term occurs in the document, its value in the vector is non-zero. Several different ways of computing these values, also known as (term) weights, have been developed. One of the best known schemes is tf-idf weighting.

Original

Vector space model, or term vector model as it is also known, is an algebraic model for representing objects (although it is mainly used for text documents) as vectors of identifiers; for example, index terms. It is used in information retrieval and filtering, indexing and relevancy rankings, and was first used in the SMART Information Retrieval System. A document is represented as a vector, with each dimension corresponding to a separate term. If a term occurs in the document, the value will be non-zero in the vector. Many different ways of computing these values (aka (term) weights) have been developed; one of the best known schemes is tf-idf weighting.

Similaridade de grau II (trechos em vermelho são comuns ao original)

Within Information Retrieval each document in a set can be represented as a point in high-dimensional vector space, this representation is called the vector space model. Information Retrieval queries are also represented as vectors in the same vector space; these are then used in conjunction with the document vectors to find relevant documents. The two vectors are compared and the documents with a higher document-query similarity are ranked higher in terms of relevance.

Similaridade de grau IV – não há trechos semelhantes ao original

Nos trechos acima, foram consideradas similares as subsequências de tamanho maior ou igual a 5 palavras, ignorando sinais de pontuação.

As instruções foram passadas ao grupo de participantes de modo a obter 19 textos submetidos para cada tema, além do artigo original da Wikipedia. A tabela a seguir fornece detalhes sobre as submissões:

	Grau de similaridade			
Tema	(I)	(II)	(III)	(IV)
A	4	3	3	9
B	3	4	3	9
C	3	5	4	7
D	4	4	5	6
E	5	3	4	7

Tabela 4.1 *– Ocorrências de cada grau de plágio, em relação ao artigo original*

Os testes foram executados em rodadas, onde cada rodada corresponde às submissões referentes a um tema, acrescentando-se o artigo da original correspondente. Em cada rodada uma foram considerados os casos de plágio do artigo em questão, ignorando possíveis similaridades existentes apenas entre as submissões, devido à natureza dos dados. Os arquivos do *corpus* foram fornecidos ao programa desenvolvido neste trabalho como conteúdo de linguagem natural extraído de páginas do Adessowiki, enquanto para o JPlag bastou especificar o diretório contendo as submissões a serem analisadas.

A tabela a seguir exibe os resultados de todas as rodadas:

			JPlag		TCC	
Tema	Entradas	Plágio	Suspeitas	Verificadas	Suspeitas	Verificadas
A	20	10	9	9	9	9
B	20	10	9	9	9	9
C	20	12	10	10	11	11
D	20	13	13	13	12	12
E	20	12	11	11	12	12
Total	95	57	52	52	53	53

Tabela 4.2 *– Detalhes dos testes para linguagem natural*

Entradas: arquivos a serem analisados; Plágio: arquivos classificados como plágio pelos criadores do corpus; Suspeitas: entradas identificadas pelos programas; Verificadas: suspeitas levantadas pelos programas que foram classificadas como plágio pelos criadores do corpus

Os resultados de ambas soluções podem ser considerados satisfatórios e praticamente idênticos: tanto a ferramenta desenvolvida neste trabalho quanto o JPlag identificaram grande parte dos casos reais de plágio (índices de acerto de 92.9% e 91.2%, respectivamente) e não apresentaram falsos positivos (identificação errônea de uma entrada como suspeita).

4.6.2 Código-fonte

Os testes com este tipo de conteúdo apresentaram algumas complicações. As linguagens suportadas tanto pela ferramenta desenvolvida neste trabalho quanto pelo JPlag são C e C++. Porém, a imensa maioria das tarefas disponíveis no ambiente do Adessowiki, até o momento da realização dos testes, estava escrita em Python. Isso impossibilitou testes com os conteúdos reais do Adessowiki, uma vez que a quantidade de código em C e C++ era extremamente baixa para comparações efetivas.

A solução encontrada foi o uso de exercícios-programa (EPs) na linguagem C, de uma disciplina introdutória oferecida pelo IME-USP no passado, que não utilizou o ambiente Adessowiki no momento em que foi ministrada. Todas as entradas são submissões para uma mesma tarefa, em que foi fornecido um código-base para que os alunos trabalhassem a partir dele. No total, foram analisadas 686 entradas.

Para realizar os testes do identificador de plágio para o Adessowiki, os EPs foram tratados como conteúdos obtidos a partir de páginas deste ambiente. É importante lembrar que, conforme explicado na seção 4.4, a ferramenta desenvolvida neste trabalho utiliza o Moss para análise de código-fonte. Já para executar os mesmos testes no JPlag, bastou indicar o diretório contendo as entradas.

Houve certa dificuldade em acertar os parâmetros corretos das ferramentas, pois nas opções padrão, o número de suspeitas levantado por ambos programas era muito elevado (chegando a 78% do total de entradas), o que indica um provável erro. Ainda há um aparente *bug* no Moss, onde o código-base é ignorado, isto é, incluir a opção *-b* não altera os resultados. Na tentativa de contornar essa limitação, a opção *-n* que limita os resultados do Moss foi utilizada.

Foram realizadas 4 baterias de testes, organizados da seguinte forma:

- Teste 1: foram escolhidos 150 EPs aleatoriamente, dentre os 686 possíveis
- Teste 2: selecionados os 150 EPs menos extensos, com maior probabilidade de serem semelhantes ao arquivo base, e consequentemente, entre si
- Teste 3: escolhidos para análise os 150 EPs mais extensos, por terem maior probabilidade dos alunos terem adicionado conteúdo efetivo ao código-base
- Teste 4: todos os 686 EPs disponíveis foram analisados

Os resultados são descritos na tabela abaixo:

		JPlag		TCC		Em comum	
Teste	Entradas	Suspeitas	%	Suspeitas	%	Suspeitas	%
1	150	29	19.3	42	28.0	7	4.7
2	150	63	42.0	75	50.0	14	9.3
3	150	14	9.4	24	16.0	3	2.0
4	686	72	10.5	98	14.2	15	2.2

Tabela 4.3 *– Detalhes dos testes para código-fonte*

Analisando percentual de suspeitas levantadas por cada ferramenta, podemos considerar os resultados satisfatórios, uma vez que a diferença entre os valores é sempre inferior a 10%. Mas ao verificar o número de suspeitas comuns (levantadas por ambos), os resultados não são bons, especialmente nos testes 3 e 4. É provável que esta falta de consistência esteja relacionada aos parâmetros passados a cada programa, e principalmente ao problema mencionado anteriormente sobre a opção do Moss de trabalhar com um código-base.

Uma dificuldade de trabalhar com dados reais é que não é possível afirmar com total certeza quais casos são de fato plágio, portanto não podemos ter uma ideia precisa da qualidade dos resultados, embora [4] indique valores esperados entre 10% e 20% do total de trabalhos entregues.

Capítulo 5

Conclusão

5.1 Considerações Finais

Apresentamos uma ferramenta que cumpre o objetivo estabelecido inicialmente: identificar casos de possível plágio no ambiente Adessowiki, tanto para código-fonte quanto para texto em liguagem natural.

O programa ainda está em fase de testes, e não foi empregado no decorrer de qualquer curso. O resultado para identificação de plágio para conteúdos em linguagem natural foi bastante satisfatório; mas para conteúdos do tipo código-fonte, a ferramenta mostrou resultados apenas razoáveis.

A principal limitação da ferramenta está na sua dependência de um componente externo (Moss) para a análise dos códigos-fonte extraídos. Ainda que raramente, podem ocorrer falhas no servidor do serviço, fazendo o sistema parar de responder, e impossibilitando a busca por suspeitas de plágio desse tipo de conteúdo.

Apesar da limitação descrita anteriormente, a ferramenta cumpre o objetivo previsto, e espera-se que este projeto possa ser empregado futuramente nos cursos que utilizam o Adessowiki, contribuindo de alguma maneira para uma melhor qualidade de ensino.

5.2 Trabalhos Futuros

Para a continuidade do projeto, seria interessante tratar localmente os conteúdos do tipo código-fonte, conseguindo assim uma independência de ferramentas externas. Também poderiam ser aplicadas outras técnicas de detecção de plágio, em conjunto com o *winnowing*, melhorando os resultados fornecidos pelo programa.

Outra possível melhoria está ligada à apresentação dos relatórios gerados. Conteúdos dinâmicos poderiam melhorar não apenas a interatividade, mas também a quantidade de informação exibida para

o usuário. Porém o suporte a este tipo de conteúdo depende de configurações no servidor do Adessowiki, e no momento estão fora do escopo do projeto.

Referências Bibliográficas

[1] Alex Aiken. Moss – a system for detecting software plagiarism. `http://theory.stanford.edu/~aiken/moss/`.

[2] Pew Reseasrch Center. The digital revolution and higher education. `http://www.pewinternet.org/Reports/2011/College-presidents/Summary.aspx`.

[3] Paul Clough and Mark Stevenson. Developing a corpus of plagiarised short answers. *Language Resources and Evaluation: Special Issue on Plagiarism and Authorship Analysis*, 2011.

[4] Fintan Culwin, Anna MacLeod, and Thomas Lancaster. Source code plagiarism in UK HE computing schools,issues, attitudes and tools. 2001.

[5] Django Software Foundation. Django : The web framework for perfectionists with deadlines. `https://www.djangoproject.com/`.

[6] Python Software Foundation. Python programming language. `http://www.python.org/`.

[7] Wikimedia Foundation. Wikipedia, the free enciclopedia. `http://en.wikipedia.org/`.

[8] Edward L. Jones. Metrics based plagiarism monitoring. *6th Annual CCSC Northeastern Conference*, 2001.

[9] R. A. Lotufo, R. C. Machado, A. Körbes, and R. G. Ramos. Adessowiki on-line collaborative scientific programming platform. *The 5th International Symposium on Wikis and Open Collaboration*, 2009.

[10] Roberto Lotufo and Rubens Machado. Adessowiki. `http://www.adessowiki.org`.

[11] Guido Malpohl, Sreenivas Viswanadha, James Hunt, et al. JPlag – detecting software plagiarism. `https://www.ipd.uni-karlsruhe.de/jplag/`.

[12] Paul McGuire. Pyparsing. `http://pyparsing.wikispaces.com/`.

[13] José Manuel Moran. Educação a Distância. http://www.eca.usp.br/prof/moran/dist.htm.

[14] BBC News. German Defence Minister Guttenberg resigns over thesis, 2011. http://www.bbc.co.uk/news/world-europe-12608083.

[15] Ivônio Barros Nunes. Educação a Distância: Definição, Características e Evolução Histórica. *Revista Educação a Distância*, (4):7–25, 1994.

[16] Plagiarism.org. What is plagiarism? http://www.plagiarism.org/plag_article_what_is_plagiarism.html.

[17] Radim Rehurek. Plagiarism Detection through Vector Space Models Applied to a Digital Library.

[18] Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: Local Algorithms for Document Fingerprinting. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2003.

[19] Walter Weiszflog. *Michaelis – Moderno Dicionário: Inglês-Português/Português-Inglês*. Editora Melhoramentos.

Parte II

Parte Subjetiva

Capítulo 6

Desafios e Frustrações

O maior desafio foi conciliar o tempo e energia gastos entre o projeto, o trabalho e as duas últimas disciplinas restantes. O planejamento feito no início do primeiro semestre teve que ser adaptado às condições adversas do segundo semestre.

Um outro problema foi a falta de referências no assunto (identificação de plágio). Embora haja inúmeras ferramentas disponíveis para esta finalidade, foi bem difícil encontrar artigos explicando detalhadamente as ideias utilizadas. Uma possível explicação para este fato pode estar na quantidade de ferramentas proprietárias para este fim, em contraste com poucas alternativas de código aberto, além da resistência de parte dos docentes em divulgar os métodos que utilizam para detectar desonestidade acadêmica, com receio de que esta informação possa ser usada erroneamente.

A não implementação de outras técnicas para detecção de similaridade também causou frustração. É claro que o escopo do projeto limita as possibilidades, mas algumas estratégias pareceram bastante interessantes, e seriam aplicações práticas de conceitos abstratos aprendidos com muito custo durante o curso. Além disso, o tempo gasto em partes auxiliares da ferramenta, como os *parsers* foi muito maior do que o esperado.

Capítulo 7

Disciplinas e Conceitos Relevantes

A seguir, uma lista das disciplinas mais importantes para a realização do projeto, e os conceitos relevantes para este trabalho abordados em cada uma delas.

- **MAC0122 – Princípios de Desenvolvimento de Algoritmos:** Forneceu a base necessária para compreender o funcionamento de algoritmos, e em especial o estudo de busca de padrões foi útil para a compreensão e implementação do *winnowing*.
- **MAC242 – Laboratório de Programação II:** Aqui foram aprendidos conceitos de processamento de texto e desenvolvimento de *parsers*, peças fundamentais para este projeto.
- **MAC0414 – Linguagens Formais e Autômatos:** Trata-se de uma disciplina mais teórica, porém foi importante para a compreensão do uso de gramáticas e expressões regulares no *parser* que foi implementado.
- **MAC0417 – Visão e Processamento de Imagens:** Apesar deste trabalho não estar diretamente relacionado com a área, foi nesta disciplina que tive o primeiro contato com a linguagem Python, na qual o projeto foi desenvolvido, e que tornou-se minha linguagem favorita.

Capítulo 8

Agradecimentos

Gostaria de agradecer primeiramente ao professor Roberto Hirata, que sempre foi muito solícito, especialmente neste período de orientação, e por acreditar no projeto quando tudo parecia perdido; e ao professor Carlos Eduardo Ferreira pela preocupação e dedicação em garantir o sucesso dos TCCs. Agradeço ainda aos professores Roberto Lotufo e Rubens Machado pela permissão de uso do Adessowiki neste trabalho, e pelos esclarecimentos e sugestões dadas, e à professora Nina Tomita Hirata pelo auxílio com os testes.

Também deixo minha gratidão aos demais professores e funcionários do IME-USP, pelos ensinamentos dados e serviços prestados durante minha graduação.

Finalmente, aos amigos que fiz nestes longos anos de BCC: obrigado pelo apoio!

Apêndice A

Código-fonte do *winnowing*

A seguir, o código em Python das principais funções usadas na técnica de *winnowing*, para detecção de prováveis trechos plagiados entre textos.

A.1 Pré-processamento e mapeamento do texto

```
def cleanAndMapText(self, textFile):
    inputText = open(textFile, 'r').read()
    cleanText = ''
    positionMap = []
    irrelevantChars = [' ', ',', '\n', '.', '\t', '=', '-', ':']
    for i in range(0, len(inputText)):
        if inputText[i] not in irrelevantChars:
            cleanText += inputText[i].lower()
            positionMap.append(i)
    return cleanText, positionMap
```

A.2 *Winnowing*

```
def winnow(self, text, windowSize, k):
    rawHashes = []      #k-grams hashes
    fingerPrints = {}   #saves hash and position
    for i in range(0, len(text) - k + 1):
        rawHashes.append(hashlib.md5(text[i:i+k]).hexdigest())
```

```
    min = 0
    #given a window length w, picks a hash
    for i in range(len(rawHashes) - windowSize + 1):
        window = rawHashes[i:i+windowSize]
        for j in range(0, len(window)):
            if window[j] <= rawHashes[min] or min < i:
                min = i+j
        fingerPrints[rawHashes[min]] = min
    return fingerPrints
```

A.3 Comparação entre dois conjuntos de *fingerprints*

```
def compareFingerPrints(self, fingerPrintsA, fingerPrintsB):
    a = set(fingerPrintsA.keys())
    b = set(fingerPrintsB.keys())
    sharedFingerPrints = list(a.intersection(b))
    return sharedFingerPrints
```

A.4 Busca pela *substring* delimitada por um conjunto de *fingerprints*

```
def findSubstring(self, sharedFingerPrints, fingerPrints, map, text):
    positions = []
    for f in sharedFingerPrints:
        positions.append(fingerPrints[f])
    if positions != []:
        start = map[min(positions)]
        end = map[max(positions)]
        return text[start:end]
    else:
        return ''
```