Um sistema de reconhecimento, busca aproximada, e alinhamento múltiplo de documentos históricos

Gustavo Enrique Salazar Torres

Tese apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Doutor em Ciências

Programa: Ciências da Computação
Orientador: Prof. Dr. Alair Pereira do Lago

São Paulo, Dezembro de 2017

Um sistema de reconhecimento, busca aproximada, e alinhamento múltiplo de documentos históricos

Esta é a versão original da tese elaborada pelo candidato Gustavo Enrique Salazar Torres, tal como submetida à Comissão Julgadora.

Um sistema de reconhecimento, busca aproximada, e alinhamento múltiplo de documentos históricos

Esta versão da tese contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 06/12/2017. Uma cópia da versão original está disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Prof. Dr. Alair Pereira do Lago (orientador) - IME-USP
- Prof. Dr. Carlos Eduardo Ferreira - IME-USP
- Prof. Dr. Roberto Hirata Junior - IME-USP
- Prof. Dr. Tomasz Kowaltowsk - IC/UNICAMP
- Prof. Dr. Adriano César Machado Pereira - DCC/UFMG

Agradecimentos

A Deus por não me abandonar durante este esforço, por me manter firme até o fim.

Ao meu orientador e amigo, Alair, pela sua paciência ao corrigir os meus erros e por me guiar neste caminho emocionante que é fazer ciência.

À minha esposa Hilda, pelo seu amor, carinho e paciência para comigo nesta jornada, pelas inúmeraveis horas em que você me acompanhou, este trabalho é também seu.

Aos meus pais, sogros, irmãos e primos pelo constante apoio e palavras de ânimo e coragem.

Aos meus amigos, Ana Lydia, Daniel, Isabela, Edivando, Thamara, Carlos, Nicola, Alice e Lucila, a companhia de vocês nesses cinco anos foi fundamental para chegar até aqui.

Resumo

Salazar-Torres, Gustavo E. **Um sistema de reconhecimento, busca aproximada, e alinhamento múltiplo de documentos históricos**. 2017. 126 f. Tese (Doutorado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2017.

Este trabalho relata o projeto, implementação e teste de um sistema que integra reconhecimento digital de textos em imagens de documentos antigos com busca e indexação aproximada e alinhamento múltiplo dos documentos.

O reconhecimento de texto em documentos históricos, com tipografia muito diferente da atual e estado de conservação precário, é altamente sujeito a erro de forma que a comparação e o alinhamento múltiplo necessário a uma reconstituição de um estudo da origem filogenética precisa usar técnicas de Busca Aproximada para contornar o problema.

Assim, propomos um sistema de pontuação para documentos históricos que releva o alinhamento de termos similares sujeito aos erros de OCR ou grafias diversas. Para a identificação de palavras semelhantes dentro de uma taxa de erro preestabelecida, usamos um cálculo da distância de edição proposto por Ukkonen que ainda requer muito recurso computacional. Nós propomos e testamos o uso de um filtro que usa sacos de símbolos que garante os mesmos resultados com uma redução drástica da computação das distâncias de edição. Desta forma, para identificação de trechos semelhantes em documentos históricos diversos sujeito a erros, foi proposta uma estratégia de alinhamento múltiplo local que utiliza técnicas de extração de sementes e expansões de alinhamentos locais como a usada pela ferramenta BLAST, muito conhecida na área de Bioinformática.

Diante das deficiências encontradas nos trabalhos de processamento de imagens de documentos históricos no que diz respeito à segmentação de caracteres, propusemos e implementamos um novo algoritmo de segmentação baseado em uma modelagem que permite a elaboração de um algoritmo de otimização através de programação dinâmica, ao contrário das heurísticas existentes baseadas em estratégias gulosas. Foi também proposta uma arquitetura que aproveita dos alinhamentos produzidos contra documentos gabarito de forma a retreinar e aumentar a acurácia do classificador OCR.

Palavras-chave: Processamento de Imagens, Busca Aproximada, Alinhamento Local, Ali-

nhamento Múltiplo, Recuperação de Informação, Programação Dinâmica.

Abstract

Salazar-Torres, Gustavo E. **A recognition, approximate search and multiple align system for historical documents**. 2017. 126 f. Tese (Doutorado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2017.

This thesis describes the design, implementation and experiments on a software system that integrates digital image processing for historical documents with approximate search and multiple alignment for these kind of documents.

Old typography along with a bad preservation state is very common in historical documents. These two features generate noise when images of this kind are processed by modern OCR systems. Therefore, in order to perform comparison and multiple alignment that would lead to a reconstruction of the phylogenetic tree for these documents, one should use Approximate Search techniques to overcome this problem.

We also propose an scoring system for historical documents based on the alignment of similar words bounded by an error rate or with diverse spelling. To identify similar words considering a fixed error rate, we use an an efficient Ukkonen's edit distance algorithm which still demands a lot of computational resources. Thus, we proposed and ran experiments on a filter that uses a distance based on bag of characters that not only guarantees the same results but also drastically reduces the number of calls to Ukkonen's edit distance. In order to identify similar passages among historical documents allowing errors, we proposed a multiple local alignment algorithm that lends techniques like seeding and local alignment expansion from tools like BLAST, very popular in Bioinformatics.

We also identified limitations in many solutions proposed for the problem of touching character segmentation in the image processing literature. We proposed and implemented a novel segmentation algorithm based on an model that allows to introduce an optimization algorithm that uses dynamic programming, unlike existing heuristics based on greedy strategies. We also proposed an architecture that harnesses on the alignments generated against a ground-truth text document in order to retrain and increase accuracy for an OCR system.

Keywords: Image Processing, Approximate Search, Local Alignment, Multiple Align-

ment, Information Retrieval, Dynamic Programming

Sumário

Lista de Figuras

Capítulo 1

Introdução

1.1 Motivação

Documentos históricos são objeto de trabalho e estudo por parte de linguistas e historiadores. Estes documentos possuem importância pois conformam a herança cultural de muitos países tanto no ocidente como no oriente. Os profissionais que de dedicam ao estudos destes documentos realizam tarefas de anotação, transcrição entre outros. Estas tarefas são muito árduas já que geralmente os documentos possuem várias centenas de páginas impressas ou manuscritas em dialetos antigos de idiomas como português, inglês, espanhol, árabe, grego, latim, etc.

Nos últimos anos tem surgido muito material digitalizado a partir destes documentos históricos devido as melhorias das plataformas computacionais em aspectos como poder computacional e de armazenamento. Um exemplo clássico é o Google Books que disponibiliza documentos como "The Works" de Francis Bacon impresso no ano de 1730. No entanto não existe um número suficiente de pessoas treinadas para analisar essa quantidade de documentos históricos mais amplamente acessíveis. É neste ponto que a Ciência da Computação tem contribuído para facilitar o trabalho destes profissionais.

Fruto dessa contribuição foi a aparição das Humanidades Digitais que teve nos seus primórdios o trabalho pioneiro do padre jesuíta Roberto Busa que em 1949 projetou, com o apoio de Thomas J. Watson fundador da IBM, o *Index Thomisticus*, um índice para fazer busca textual nos 59 volumes que compõem a obra de Santo Tomás de Aquino. Atualmente, as Humanidades Digitais contam com congressos anuais onde são expostos trabalhos sobre distintos problemas que surgem no processamento de documentos históricos ([Dh214],[Dh215]).

É nessas conferências que são apresentados os sistemas que dão suporte para tarefas como alinhamento de imagens e transcrições ([The15]), análise de frases paralelas em documentos históricos ([CKP+12]), sistemas de apoio para anotação de documentos ([GSLP14, BAKD13, FBN+14, Gar15], [Hao15]), compartilhamento de textos ([Gle15, Mar15]) e finalmente grandes projetos de digitalização como o IMPACT ([PPCA13]).

Esses documentos são organizados por assunto ou autor gerando de forma natural *bibliotecas* de documentos. Ainda, quando esses documentos são imagens digitais, é necessário obter o texto respectivo através de Sistemas de Reconhecimento Ótico (OCR). Por exemplo, um texto obtido por OCR possui taxas de erro que infelizmente são altas para documentos históricos pois os sistemas OCR não são treinados o suficiente para tipografia antiga ou para imagens que possuem muito ruído ([KFV12, GRRS09, Smi11]). Este fenômeno dificulta ainda mais a tarefa dos linguistas e historiadores pois é mais difícil

encontrar identidade entre textos transcritos e compilados com alta taxa de erro de OCR.

Assim, o problema de buscar e comparar documentos em bibliotecas de documentos históricos possui particular relevância dada a grande quantidade de dados existentes (imagens e texto), o impacto positivo para historiadores e linguistas e o desafio das dificuldades computacionais oriundas dos erros inerentes aos textos obtidos.

1.2 Objetivo

Para resolver o problema definido na seção anterior estabalecemo como objetivo principal desse trabalho:

- Construir um sistema web que facilite a colaboração de vários usuários na análise de documentos históricos. Este sistema deve integrar algoritmos de busca aproximada, alinhamento múltiplo local e um sistema OCR que possa obter textos a partir de imagens de documentos históricos que possam aproveitar das comparações com textos similares conhecidos para diminuir os erros inerentes ao processo.

1.3 Organização do trabalho

O foco do nosso trabalho é em três assuntos: algoritmos de Processamento de Imagens para melhorar a segmentação de páginas de documentos com ruído, algoritmos de Busca Aproximada que permitem submeter consultas que levam em conta o ruído contido em bibliotecas de documentos históricos e, finalmente, algoritmos de alinhamento local e múltiplo para encontrar regiões compartilhadas entre tais documentos.

Os diferentes algoritmos foram organizados em um sistema web apresentado no capítulo 2.1 que permite a diferentes usuários interagir em tarefas de reconhecimento e alinhamento de documentos históricos. A arquitetura do sistema permite executar essas tarefas em paralelo para facilitar a colaboração de diferentes usuários. Também faz parte deste sistema um módulo de visualização de alinhamentos locais e múltiplos escrito em Javascript. Este módulo é mostrado no apêndice A.

Assim, no capítulo 3 - *Processamento de Imagens* - apresentamos um procedimento que melhora a qualidade da imagem escaneada de um documento impresso. Esse procedimento recebe como entrada uma imagem em níveis de cinza e devolve uma imagem binária construída a partir da segmentação da imagem de entrada. Essa segmentação aplica técnicas de limiarização específicas para documentos históricos para em seguida obter as linhas, palavras e caracteres. No que diz respeito à segmentação de caracteres, é proposto um novo algoritmo de segmentação de imagens que possuem caracteres ligados. Esse algoritmo utiliza programação dinâmica e técnicas de Estatística Robusta Multivariada para encontrar tal segmentação. A imagem melhorada é processada por um sistema OCR que produz um documento de texto. O sistema permite agrupar documentos em bibliotecas sobre as quais é possível realizar busca.

No capítulo 4 - *Busca Aproximada e Recuperação de Informação* - mostramos justamente como realizar essa busca em bibliotecas que contém documentos com ruído. Os principais ruídos de interesse referem-se ao fato de alguns termos de um documento possuírem caracteres errados produzidos pelo sistema OCR ou que apresentam escrita diferente devido à evolução da língua em que o documento foi escrito ou por erros de copistas. Assim, nesse capítulo mostramos algoritmos que, dada uma consulta e uma taxa de erro submetidas pelo usuário, obtém termos próximos presentes no dicionário de termos de

uma biblioteca. Para encontrar tais termos próximos, esses algoritmos utilizam técnicas de Busca Aproximada.

No capítulo 5 - *Alinhamento de documentos de texto* - mostramos como é possível adaptar técnicas de Bioinformática para obter alinhamentos múltiplos locais dos documentos de uma biblioteca levando em conta também a presença de erros nos termos. Para obter tais alinhamentos são utilizadas as técnicas de Busca Aproximada propostas no capítulo 4 para produção mais apropriada de sementes de alinhamento como as utilizadas pelo algoritmo BLAST.

No capítulo 6, propomos um método para treinar sistemas OCR a partir do alinhamento de pares de documentos. Mostramos o projeto deste método que implementa um laço onde etapas de alinhamento de textos e reconhecimento de imagens são alternadas para corrigir a saída do sistema OCR.

Finalmente, no capítulo 7, foram rodados experimentos para ter uma noção da efetividade e performance das técnicas apresentadas nos capítulos antes descritos. O conjunto de dados utilizado nos experimentos é composto por imagens e texto de obras originalmente escritas entre os seculos 13-14 e o século 17. As características desse conjunto de dados são descritas no capítulo 7, assim como o resultado dos experimentos rodados sobre esse conjunto.

1.4 Trabalhos relacionados

Trabalhos publicados relacionados ao nosso podem ser classificados nas áreas de Processamento de Imagens, Recuperação de Informação, Detecção de Plágio, Biologia Molecular Computacional e Linguística Computacional. Em seguida descrevemos os trabalhos mais relevantes de cada área que são relacionados a este trabalho.

1.4.1 Processamento de Imagens

Existe uma vasta quantidade de trabalhos nesta área com respeito ao reconhecimento de documentos históricos. A maior parte deles provém do projeto IMPACT [PPCA13] que agrupa vários subprojetos de digitalização e processamento de documentos históricos na Europa. Grande quantidade desses documentos pertencem aos seculos 12 e 13, por tanto, são manuscritos. Contudo, a partir da utilização da imprensa em 1430 por Gutemberg, existe também uma grande quantidade de documentos impressos com tipos móveis. Esse último tipo de documento é do nosso interesse.

Nesse sentido um dos trabalhos que mais chamou a nossa atenção é o de Nikolau et. al.[NMG+10] pois este descreve um fluxo de processamento completo de imagens de documentos impressos desde a limiarização da imagem até a obtenção de caracteres e possui taxas de reconhecimento boas mesmo comparado com motores de OCR industriais. O trabalho propõe uma adaptação de um algoritmo de varredura clássica em imagens binárias para segmentar as linhas para logo continuar com a segmentação de palavras e caracteres. Especificamente, na segmentação de caracteres, a técnica utilizada baseia-se nos esqueletos da imagem original.

Por outro lado, em [ZEP10], é apresentado um método que processa documentos históricos porém chegando somente até a segmentação de palavras. Os autores mostram um método que permite extrair características das regiões que contêm palavras sem extrair os caracteres. Essas características permitem realizar consultas (em forma de imagens) diretamente na imagem original. O método inclui técnicas básicas de limiarização da

imagem (algoritmo de Otsu [Ots79]) para depois produzir a segmentação das palavras. O método não identifica com precisão como são localizadas as linhas da imagem.

Por outro lado, em [RVGF14] foi implementado um sistema que reconhece imagens de documentos que correspondem a dados de meteorologia no inicio do século 20. As imagens contém dados tabulados e não possuem muito ruído. O método utiliza o algoritmo de Niblack [Nib85] para obter uma imagem binária e logo aplicar uma transformação que detecta linhas (transformada de Hough). Assim, obtém as imagens que representam os caracteres para finalmente extrair as suas características.

Em [KFV12] é apresentado um método parecido com [ZEP10] onde a segmentação é realizada até o nível das palavras. A diferença está no método de limiarização que utiliza o filtro de Niblack ([Nib85]) e na técnica de segmentação de linhas. Especificamente, esse trabalho utiliza um algoritmo de varredura clássico para localizar as linhas. Em seguida, são obtidas regiões menores que podem pertencer a caracteres, símbolo de pontuação ou ruído. O ruído é removido usando um critério simples que remove regiões com menos de 10 pixels de área. Finalmente, para cada região que sobra são extraídas características que incluem, por exemplo, o histograma vertical (o número de pixels em cada coluna da região que contém a imagem) ou, para cada coluna da região, a distância desde o topo até o primeiro pixel de foreground. As características têm por intuição refletir a forma externa e interna da cada palavra produzindo assim um índice de características. Esse índice é consultado levando em conta uma palavra de entrada para a qual também são extraídas as mesmas características (o sistema produz uma imagem com os caracteres da palavra de entrada).

Temos observado que a maioria dos trabalhos utiliza técnicas básicas de limiarização ou possui um número pequeno de hipóteses para classificar se uma imagem possui um ou mais caracteres. Por outro lado, existe tendência de evitar a segmentação de caracteres, pois é uma tarefa árdua e propensa a erros dado que a tipografia da maioria dos documentos históricos é desconhecida.

A este respeito, existem na literatura algumas soluções que utilizam programação dinâmica para detectar caracteres individuais em imagens que possuem mais de um caractere [PPLD12]. Esses trabalhos exibem uma forma de geração de regiões da imagem e pontuações para essas regiões obtidas a partir de um modelo supervisionado de Aprendizado de Máquina treinado para reconhecer uma certa tipografia. As regiões são logo combinadas usando uma solução de programação dinâmica e a pontuação da região para encontrar a segmentação ótima.

Por último, recentemente tem surgido trabalhos que utilizam alinhamento múltiplo de palavras para melhorar a saída de um motor OCR [Bos09, Lis14]. Esses trabalhos tem se focado em documentos históricos em grego.

1.4.2 Recuperação de Informação

Não existem muitos trabalhos nesta área que tratem documentos históricos onde a taxa de erro nas palavras pode ser alta. A maior parte dos trabalhos tenta aplicar regras de normalização dos termos de um documento histórico para obter um novo texto que usa termos modernos com respeito a uma língua ([BBP11, GRR+11, HM11, RHM12]). Esses novos textos podem ser indexados de forma que se possam realizar consultas utilizando termos modernos de uma língua.

Por outro lado, podemos citar o trabalho de Hauser [HS07] que utiliza técnicas de Busca Aproximada para encontrar os termos próximos dentro de um documento histórico. Os termos próximos substituem os termos originais criando uma consulta estendida.

1.4.3 Detecção de Plágio

O problema de obter alinhamentos locais para pares de documentos é um problema que tem sido enfrentado no passado. Em particular a área de Detecção de Plágio tem utilizado técnicas de alinhamento local (algoritmo de Smith-Waterman [SW81]) para detectar se houve plágio entre dois textos [Gli14]. A técnica foi utilizada na competição de alinhamento de textos da conferência PAN [PSBCR10]. Vale a pena ressaltar que a maioria das técnicas submetidas na competição utilizam técnicas baseadas métricas como tf-idf [SPGS15] ou em ocorrências de k-gramas em pares de documentos onde se suspeita que existe plágio [TMR13]. Esta última é o fundamento de uma ferramenta comercial de detecção de plágio [PV17].

Em outro tipo de aplicações podemos citar trabalhos como [Gle15, OHR] onde são alinhados pares de documento de textos do século 19 para descobrir se segmentos de texto foram compartilhados. O algoritmo cria índices de k-gramas de termos de documentos que logo são indexados. Ao comparar dois textos a ferramenta utiliza o índice para encontrar repetições de k-gramas. Essas ocorrências são concatenadas encontrando assim alinhamentos locais.

Em [CKP+12] foi utilizado um algoritmo que também se baseia na geração de índices de k-gramas para parágrafos dentro de um documento de texto. Quando uma frase é submetida pelo usuário, o sistema retorna os parágrafos que possuem pelo menos dois k-gramas em comum.

Finalmente, em [CD16], também é utilizada uma lista de k-gramas para os parágrafos de um documento. O sistema permite que o usuário submeta uma frase e uma taxa de erro. Em seguida são retornadas as frases do documento que possuem no máximo o número de erros indicado pelo usuário.

1.4.4 Linguística Computacional

No que diz respeito ao alinhamento múltiplo em documentos histórico, podemos citar alguns trabalhos que se aproximam do nosso. Por exemplo, o alinhamento múltiplo de sentenças tem sido aplicado à geração de paráfrases ([BL03],[SRPE06]). O problema da geração de paráfrases consiste em gerar uma sentença diferente a partir de outra mas mantendo o mesmo significado e utiliza-se alinhamento múltiplo para encontrar um mapeamento entre os sintagmas[1] de várias sentenças.

Não conhecemos nenhum trabalho que se preocupe em obter e exibir alinhamentos locais múltiplos de documentos históricos.

1.4.5 Outros comentários

Há que se observar a utilização de métodos e técnicas computacionais aprimorados na Biologia Molecular Computacional neste trabalho (vide capítulo 5), como o algoritmo estrela para alinhamento múltiplo ou a heurística utilizada pela ferramenta BLAST, desenvolvida dentro do âmbito do NCBI por uma equipe de cientistas que contou com Gene Myers, autor do algoritmo de comparação de textos usado no GNU diff. Tal como Myers, muitos cientistas da Computação especializados em algoritmos de processamento de textos foram atraídos por esta nova área que tinha por ambição desvendar o código

[1] Linguistas agrupam as palavras de uma língua em classes, ou sintagmas, que mostram comportamento sintático similar, e possuem o mesmo tipo semântico. Alguns dos sintagmas mais importantes são os sintagmas nominais (substantivos), sintagmas verbais e sintagmas adjetivos [MS99].

genético da vida. A pesquisa genômica necessitava de novos algoritmos especializados em processar as novas sequências genômicas que estavam sendo obtidas a cada dia e novas ferramentas eram necessárias. Com o desenvolvimento do Projeto Genoma Humano internacional, e com a pesquisa privada estabelecida pela companhia Celera Genomics, do qual Gene Myers foi vice-presidente de algoritmos, estabeleceu-se uma competição que culminou com a publicação de dois rascunhos do genoma humano que foram publicados em Nature e Science no ano 2001. De fato, as grandes quantidades de dados da Genômica requerem algoritmos muitas vezes similares àqueles necessários ao processamento dos grandes volumes de textos, particularmente disponibilizados após o desenvolvimento da Wide World Web, do Altavista e da eclosão das companhias de alta tecnologia como Yahoo e Google, especializadas em prestar serviços de Recuperação de Informação que indexariam a própria Web.

Uma ferramenta como o BLAST, porém, não se baseia apenas nos algoritmos de representação e comparação de textos usados anteriormente, até porque a contribuição do primeiro autor do trabalho científico associado à ferramenta deu-se na área da Estatística. Isto de fato é necessário pois o papel semântico dos aminoácidos presentes nas sequências proteicas são muito mais enigmáticos ao homem que as palavras presentes num documento escrito em linguagem natural. Se a modelagem matemática da Linguística fomentou o desenvolvimento de Linguagens Formais, Teoria de Autômatos e Recuperação de Informação, ciências como a Biologia Molecular Computacional e a Bioinformática foram geradas a partir da Biologia Molecular, nascida desde o princípio dentro de um caráter interdisciplinar a partir do projeto de formação de uma Nova Biologia levada a cabo particularmente pela Divisão de Ciências Naturais da Fundação Rockefeller dirigida pelo matemático Warren Weaver, cujos projetos de pesquisa patrocinados levaram à descoberta do DNA e a tantos prêmios Nobel conquistados por Linus Pauling e outros. Tal como descreve a pesquisadora Lily E. Kay em pesquisa realizada a partir dos documentos da própria Fundação, foi a irmã gêmea da Estatística, a Eugenia — com sua pretensão de desenvolver um novo homem e uma nova sociedade através da aplicação da Teoria do Controle Social tão em voga no início do século XX — a grande motivação que fomentou o desenvolvimento desta Nova Biologia que mais tarde viria a ser chamada de Biologia Molecular. [Kay93]

Depois do fomento da Nanotecnologia, o desenvolvimento de sensores neurológicos cada vez menores e mais precisos tem propiciado que também a Neurociência tenha sido fonte de grandes volumes de dados, de forma que, recentemente, Gene Myers tenha se movido para a área da Ciência Cognocitiva, que também busca o reconhecimento de padrões apoiados na Estatística e na Aprendizagem Computacional. Se os eugenistas do passado hoje preferem chamar-se a si mesmos de engenheiros sociais, foge ao nosso escopo discorrer sobre como é que a Recuperação de Informação, a Bioinformática, a Nanotecnologia e a própria Ciência Cognocitiva contribuem não somente para expandir as fronteiras do conhecimento mas principalmente para forjar engrenagens importantes de um sistema de engenharia social concebido para controlar, adestrar e automatizar o homem e a sociedade. [dL]

Capítulo 2

Arquitetura do sistema

Neste capítulo mostramos a arquitetura de um sistema que se presta a executar tarefas de Busca Aproximada, Alinhamento Local e Reconhecimento de imagens en documentos históricos. O sistema foi projetado para facilitar a colaboração de vários usuários na Internet e, por tanto, possui uma arquitetura que facilita a execução de distintas tarefas em paralelo e de forma assíncrona, ou seja o processo roda no servidor e notifica o usuário (via o navegador) a cada etapa da execução.

O sistema web foi implementado na linguagem Scala pois possui frameworks que facilitam a construção de sistemas web assim como a execução de tarefas em paralelo. Por outro lado, o sistema possui uma interface de usuário escrita em Javascript que ajuda principalmente na visualização dos alinhamentos locais.

A arquitetura do sistema foi concebida em camadas pois, de maneira natural, os algoritmos propostos utilizam serviços básicos que fornecem dados de referência de imagens e documentos de texto. Existem ainda serviços mais básicos que criam estruturas de dados de busca (Lucene) ou chamadas ao sistema OCR.

A figura 2.1 mostra as camadas e os módulos que as compõem assim como as dependências entre eles.

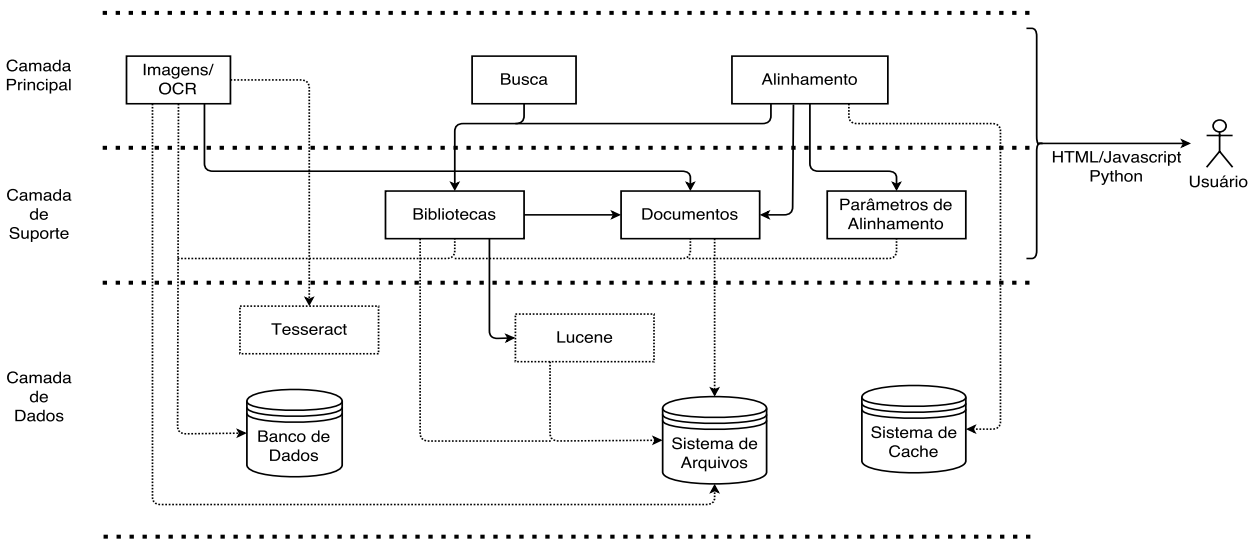


Figura 2.1: *Arquitetura do sistema*

Na continuação descrevemos em mais detalhe as camadas que compõem esse sistema.

2.1 Camadas do sistema

Este trabalho desenvolve um sistema web que integra todas as técnicas propostas nos capítulos 3, 4, 5, sendo que a implementação das técnicas apresentadas no capítulo 6 estão ainda em andamento. Esse sistema possui uma interface HTML/Javascript para permitir aos usuários obterem documentos de texto a partir de imagens, indexar os documentos para realizar consultas e ainda obter alinhamentos entre dois ou mais documentos.

O sistema web foi desenvolvido a partir do framework Play e na linguagem Scala e possui vários módulos organizados em três camadas:

Principal Esta camada oferece um conjunto de serviços fornecidos por três módulos. Estes são os módulos de Imagens/OCR, Busca, e Alinhamento.

O *módulo de Imagens/OCR* implementa os algoritmos propostos no capítulo 3. O usuário pode escolher uma pasta contendo imagens armazenadas dentro do sistema de arquivos do computador que hospeda o servidor web. Além disso, ele deve fornecer os parâmetros de execução dos algoritmos que irão limpar e construir a imagem binária para cada imagem de entrada dentro da pasta selecionada. O sistema executa esses algoritmos de forma assíncrona utilizando a tecnologia Akka disponivel como uma biblioteca para projetos na linguagem Scala. Esta capacidade permite ao usuário executar outras tarefas enquanto espera finalizar a obtenção do texto para as imagens escolhidas. O sistema produz também um arquivo adicional para cada imagem construída contendo as posições de cada caractere nessa imagem. Essa informação é de utilidade para treinar modelos de reconhecimento para o motor de OCR Tesseract que é utilizado pelo sistema. Esses modelos podem ser submetidos ao sistema de arquivos do computador para a sua utilização pelo Tesseract.

Por outro lado, o *módulo de Busca* permite ao usuário submeter consultas para as Bibliotecas criadas no sistema. Os termos das consultas são expandidos levando em conta os termos do dicionário de uma biblioteca e uma taxa de erro, ambos escolhidos pelo usuário. Esse algoritmo roda de forma forma assíncrona e os seus detalhes podem ser encontrados no capítulo 4. A interface de usuário do módulo da Busca permite que este possa selecionar um subconjunto dos documentos recuperados sobre os quais obter alinhamentos locais múltiplos.

Com respeito ao *módulo de Alinhamento*, este permite que o usuário escolha um conjunto de Parâmetros de Alinhamento, uma biblioteca e dois ou mais documentos para obter alinhamentos locais múltiplos. O sistema roda esta tarefa de forma assíncrona e informa o usuário sobre o estado de cada etapa da execução. A execução em si utiliza todos os núcleos disponíveis do computador para obter os alinhamentos. Além disso, ele armazena em um sistema de cache o resultado da etapa de Busca Aproximada para acelerar a computação de futuras requisições de alinhamento. Os detalhes da obtenção dos alinhamentos podem ser encontrados no capítulo 5.

Uma vez que o sistema termina a execução de um alinhamento local, a interface HTML/Javascript mostra os resultados no navegador. A visualização do alinhamento local de duas sequências mostra em duas colunas ambos os textos e os alinhamentos locais grifados em cinza. O usuário pode clicar em um texto grifado para mostrar os detalhes desse alinhamento na linha superior da visualização. O leitor pode consultar o apêndice A para ter uma melhor ideia da funcionalidade deste tipo de alinhamento.

Com respeito ao alinhamento múltiplo, a visualização estende a comparação para três colunas onde um deles é o documento referência e os outros são documentos secundários. Os alinhamentos em cada documento são grifados, da mesma forma que no alinhamento local, e ao serem clicados, o sistema mostra nos outros dois documentos onde se encontra tal alinhamento. O sistema disponibiliza ainda um segundo modo de visualização por colunas de termos. Nessa visualização o documento de referência serve como guia para os outros documentos. O apêndice A oferece uma visualização desta funcionalidade.

Vale a pena ressaltar que na obtenção dos alinhamentos o sistema carrega em memória a sequência de tokens obtida pelo motor de busca Lucene para cada documento selecionado. Ademais, para esses documentos são também carregados distintos tipos de dicionários cujos detalhes podem ser encontrados no capítulo 5.

Caso seja necessário executar alinhamentos locais ou múltiplos em lote, foi desenvolvida uma interface Python que se comunica com o servidor web via uma API JSON. Essa interface Python é utilizada para criar um método que combina etapas de processamento de imagens e alinhamento local de textos para treinar o sistema de OCR Tesseract. Este método é apresentado no capítulo 6.

Suporte Esta camada é composta pelos módulos de *Documentos*, *Bibliotecas* e *Parâmetros de Alinhamento* e permitem que os usuários insiram documentos, criem ou atualizem bibliotecas ou configurem parâmetros de alinhamento. Todas estas informações são armazenadas em um Banco de Dados (MySQL) ou no Sistema de Arquivos (O banco de dados serve para guardar os registros que correspondem a entidades no sistema; por outro lado, no sistema de arquivos são armazenados os índices de documentos). Vale a pena ressaltar que todos os módulos desta camada servem à camada Principal. Os usuários também podem interagir com estes módulos via interfaces HTML e Javascript para editar informações dos documentos, bibliotecas e parâmetros. Pode também interagir via Python para utilizar essas informações em tarefas de processamento por lotes. O apêndice A mostra a funcionalidade dos módulos desta camada.

Dados Esta camada não possui em si nenhum módulo mas fornece serviços básicos às outras duas camadas. Os serviços básicos que são oferecidos são o Banco de Dados (MySQL) que armazena os registros para as entidades que representam as imagens, documentos, bibliotecas e parâmetros de alinhamento. Além dessas entidades, também são representadas as línguas disponíveis no sistema.

Esta camada também contém a lógica de armazenamento e recuperação dos índices de termos das bibliotecas e documentos armazenados no sistema (veja capítulos 4 e 5). Um outro tipo de índice, o índice invertido, é criado para as bibliotecas utilizando o código binário do projeto Lucene e é utilizado pelos módulos de Busca e Alinhamento.

Finalmente, esta camada oferece acesso ao motor de OCR Tesseract que é o encarregado de obter texto a partir de imagens que são fornecidas pelo módulo de Imagens/OCR.

Vale a pena ressaltar que sistema executa as tarefas em paralelo e que, ainda, devido à utilização da biblioteca Akka é possível estende-lo para distribuir a computação entre várias maquinas e assim poder resolver tarefas de alinhamento local com uma quantidade grande de textos.

2.2 Conclusões

Foi mostrado neste capítulo um sistema web que integra tarefas de reconhecimento de imagens com alinhamento local de textos em documentos históricos. Este sistema permite a colaboração entre vários usuários e possui uma arquitetura projetada para executar as tarefas em paralelo.

Capítulo 3

Processamento de imagens

Um dos problemas mais estudados dentro da área de Processamento de Imagens é o Reconhecimento Óptico de Caracteres ou OCR por suas siglas em inglês. As soluções desenvolvidas para resolver esse problema tem tido muito sucesso pois a maioria dos documentos reconhecidos possui características bastante conhecidas: por exemplo, são documentos dos últimos dois séculos ou a tipografia é quase sempre a mesma (Helvetica ou Times Roman).

No entanto para imagens de documentos históricos mais antigos (séculos 14 ou 15) essas hipóteses não são mais válidas ([KFV12],[GRRS09],[Smi11]). Neste capítulo descrevemos o método implementado no nosso sistema para tratar este tipo de imagem.

O método esta baseado fortemente no trabalho de Nikolau [NMG+10] e possui um fluxo bastante comúm na extração de texto a partir de imagens. No entanto, ele possui características específicas para o caso de documentos históricos que o fazem interessante. Estas características são a sua adaptabilidade à linhas que possuem um linhas base diferentes. O método também propõe um algoritmo de segmentação de caracteres grudados baseado em caminhos de segmentação da imagem. Estes caminhos quebram a imagem em regiões a partir do esqueleto da mesma.

Na continuação fazemos uma breve introdução aos conceitos utilizados na implementação do método principal e do algoritmo de segmentação de caracteres grudados.

3.1 Conceitos básicos

3.1.1 Processamento de Imagens

Para introduzir o leitor leigo na área de Processamento de Imagens disponibilizamos a seguir um breve resumo dos termos mais importantes a serem utilizados no restante do capítulo[1]. Definimos como *imagem digital* uma matriz R de tamanho $m \times n$ onde cada posição $R[i,j]$ possui um valor inteiro que representa a luminosidade da imagem original naquela posição. No nosso caso trabalhamos com imagens digitais em níveis de cinza onde cada posição $R[i,j]$ varia entre 0 e 255.

Dada esta definição de imagem digital é intuitivo pensarmos em métricas, ou distâncias, entre duas posições (i,j) e (h,k). Existem especificamente dois tipos de funções de distância que são de muita importância: a distância de Manhattan D_4 definida como:

$$D_4((i,j),(h,k)) = |i-h| + |j-k|$$

[1] Muitas das definições podem ser encontradas em [Son] e [GW08]

e a distância de Chebyshev D_8 definida como:

$$D_8((i,j),(h,k)) = max\{|i-h|,|j-k|\}$$

A figura 3.1 ilustra melhor ambas as distâncias:

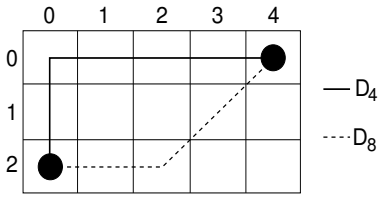
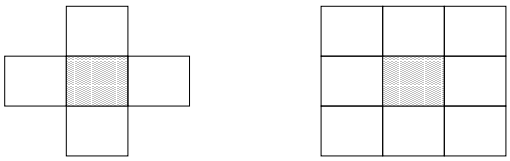


Figura 3.1: *Distâncias D_4 e D_8. A distância D_8 é também conhecida com distância "chessboard" pois ela é igual ao número mínimo de movimentos que o peça do rei realiza para se mover entre duas posições em um tabuleiro de xadrez.*

As métricas apresentadas servem para definir um outro conceito importante em processamento de imagens digitais: a *adjacência de pixels*. Dados dois pixels quaisquer (p,q) estes são chamados de *4-conexos* se a sua distância $D_4(p,q) = 1$. Analogamente, dois pixels são *8-conexos* se a sua distância $D_8(p,q) = 1$. Ambos os tipos de conectividade são mostrados na figura 3.2.



(a) *4-conexo* **(b)** *8-conexo*

Figura 3.2: *Conectividade do pixel representativo (pixel preenchido de cinza no meio) (adaptado de [Son])*

Será necessário considerar conjuntos importantes que consistem de vários pixels adjacentes - ou *componentes conexas*. Mais precisamente, podemos definir um *caminho* do pixel P até o pixel Q como uma sequência de pontos $A_1, A_2, ..., A_n$, onde $A_1 = P$, $A_n = Q$, e A_{i+1} é um vizinho de $A_i, i = 1, ..., n-1$; então uma componente conexa é um conjunto de pixels no qual existe um caminho entre dois pixels quaisquer do conjunto tal que todos os pixels no caminho pertencem ao conjunto.

Se existe um caminho entre dois pixels no conjunto de pixels da imagem, estes pixels são chamados de *conexos*. Alternativamente, podemos dizer que uma *região* é um conjunto de pixels no qual cada par de pixels são conexos ([GW08]). A relação de "conectividade" é reflexiva, simétrica e transitiva e por tanto define uma decomposição do conjunto (a imagem neste caso) em classes de equivalência (componentes conexas).

Suponha regiões disjuntas R_i na imagem que foram criadas pela relação de conectividade e suponha ainda que, para evitar casos especiais, estas regiões não tocam nas bordas da imagem. Seja R a união de todas as regiões R_i, e R^C o complemento de R com respeito a esta imagem. O subconjunto de R^C que é conexo dentro dos limites da imagem é chamado de *background*, e o restante conformam os *buracos*.

Componentes conexas são identificadas na etapa de *segmentação* da imagem. O objetivo da segmentação é dividir a imagem em partes que tem uma alta correlação com áreas ou objetos do mundo real contidos na imagem ([Son]). A segmentação pode ser de dois tipos:

Completa Produz um conjunto de regiões disjuntas que correspondem de maneira única a objetos na imagem. Isto é, dada a imagem R a segmentação completa produz um conjunto finito de regiões R_1,...,R_S tal que:

$$R = \bigcup_{i=1}^{S} R_i \text{ e } R_i \cap R_j = 0, \ \forall i \neq j$$

Parcial Produz regiões que não correspondem diretamente a objetos na imagem

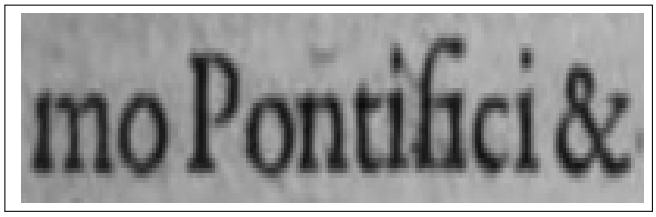
Para obter uma segmentação completa são necessários níveis mais elevados de processamento que usam conhecimento específico do domínio do problema. No nosso caso, este conhecimento está embutido em todas as etapas.

Especificamente, a etapa de *limiarização* é uma segmentação do tipo parcial e obtém uma *imagem binária* a partir da imagem em níveis de cinza. A limiarização produz uma imagem P a partir de R de acordo com um limiar T:

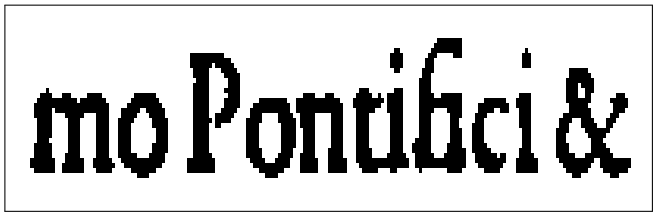
$$P[i,j] = 1, \ \text{se} \ \ R[i,j] \geq T$$
$$P[i,j] = 0, \ \text{se} \ \ R[i,j] < T$$

onde T é o limiar, $P[i,j] = 1$ para elementos do *foreground*, e $P[i,j] = 0$ para elementos do *background*. A figura 3.3 mostra um exemplo de limiarização de uma imagem em níveis de cinza.

Existem duas técnicas de limiarização: global e local. A limiarização global obtém, a partir de informações da imagem, um único limiar T que é aplicado a todos os pixels da imagem, enquanto que a limiarização local particiona a imagem em regiões e determina o limiar a partir de cada uma dessas regiões ([SSW88]).



(a)



(b)

Figura 3.3: *Exemplo de limiarização. A imagem (a) é uma imagem em níveis de cinza enquanto que a imagem (b) apresenta a imagem limiarizada.*

Dada uma componente conexa, às vezes é necessário obter o *esqueleto* dela. A ideia de esqueleto foi introduzida por Blum ([Blu67]) e ilustrada no seguinte cenário: Imagine que tem um campo de grama seca representado por uma região $X \subset \mathbb{R}^2$ e que são iniciados vários "focos de incêndios" em toda a borda do campo e a partir da qual partem

frentes de ondas que se propagam em direção ao interior da região com velocidade constante. O esqueleto $S(X)$ é o conjunto de pontos onde duas ou mais frentes de incêndios se encontram como mostra a figura 3.4.

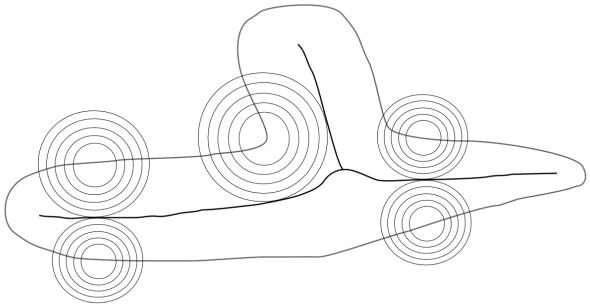


Figura 3.4: *Esqueleto conformado pelos pontos onde duas ou mais frentes de incêndio se encontram (adaptado de [Son])*

Uma definição mais formal de esqueleto se baseia no conceito de disco maximal. Um disco $B(p,r)$ com centro p e raio $r, r \geq 0$, é o conjunto de pontos com distância d a partir do centro que é menor ou igual a r.

O disco B contido em um conjunto X é maximal, se e só se não existe um disco maior incluído em X que contém B, ou seja, para cada disco B', $B \subseteq B' \subseteq X \Longrightarrow B' = B$. Discos não-maximais e discos maximais são mostrados na figura 3.5

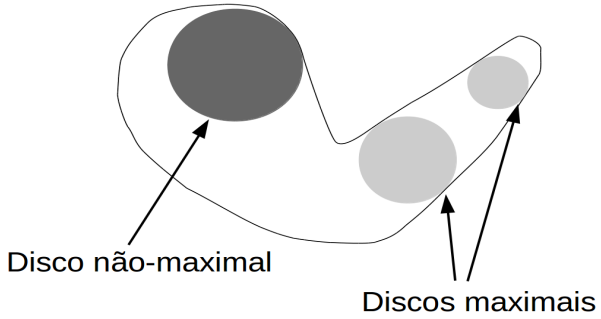


Figura 3.5: *Disco não-maximal e dois discos maximais no plano Euclidiano (adaptado de [Son])*

A métrica de distância d utilizada depende da grade e da definição de conectividade. Discos unitários no plano são mostrados na figura 3.6. O plano $\mathbb{R}^2$ junto com a distância euclidiana definem o disco B_E, enquanto que o plano $\mathbb{Z}^2$ junto com uma grade retangular definem os discos unitários B_4 para regiões 4-conexas e B_8 para regiões 8-conexas. *O*

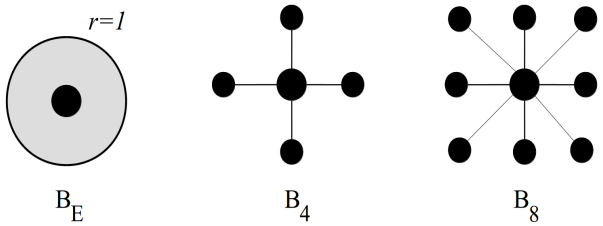


Figura 3.6: *Discos unitários para diferentes distâncias, da esquerda para direita: distância Euclidiana, 4-conexo e 8-conexo (adaptado de [Son])*

esqueleto por discos maximais $S(X)$ de um conjunto $X \subset \mathbb{Z}^2$ é o conjunto de centros p de discos

maximais

$$S(X) = \{p \in X : \exists r \geq 0, B(p,r) \text{ é um disco maximal de } X\}$$

Esta definição de esqueleto tem um significado intuitivo no plano Euclidiano. O esqueleto de um disco se reduz ao seu centro, o esqueleto de uma listra com extremos arredondados é uma linha de grossura unitária no seu centro, etc.

A figura 3.7 mostra vários objetos junto com os seus esqueletos - o retângulo, dois discos adjacentes e um anel. As propriedades do esqueleto (Euclidiano) podem ser vistas aqui - em particular, o esqueleto de dois discos adjacentes consiste de três pontos distintos ao invés de uma linha reta unindo estes dois pontos, como esperado intuitivamente.

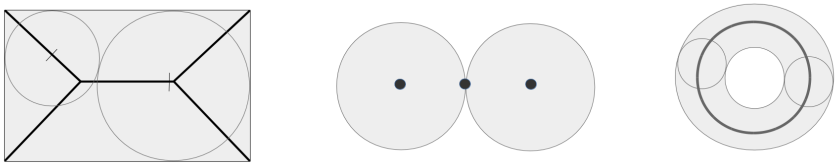


Figura 3.7: *Esqueletos de um retângulo, dois discos adjacentes e um anel (adaptado de [Son])*

O esqueleto por discos maximais tem duas propriedades que deixam ele inapropriado para aplicações práticas. Primeiro, ele não preserva necessariamente a conectividade do conjunto original; e segundo, algumas linhas do esqueleto podem ser mais grossas do que um pixel no plano discreto.

Neste trabalho estas deficiências são corrigidas utilizando o algoritmo de Hilditch ([Hil69]) que adiciona algumas condições para evitar deixar o esqueleto desconexo. A figura 3.8 mostra o resultado da aplicação desse algoritmo em uma região que contém dois caracteres. Na mesma figura mostramos o background da imagem original assim como o seu respectivo esqueleto. Como veremos mais adiante, esta informação é de utilidade para segmentar componentes conexas que contêm mais de um caractere.

Uma *região* R é *convexa* se e somente se para quaisquer dois pontos $x_1, x_2 \in R$, o segmento de reta x_1x_2 definido pelos seus pontos inicial x_1 e final x_2 está contido em R. O *fecho convexo* de uma região é a menor região convexa H que satisfaz a condição $R \subseteq H$.

Os *defeitos do fecho convexo* são as regiões $D \in H - R$, ou seja, regiões que não pertencem à região R mas que estão contidos no fecho convexo. De forma mais específica, o defeito D é a região definida entre o segmento de reta y_1y_2 onde y_1 e y_2 são dois pontos que pertencem ao contorno do fecho convexo H, e a curva entre y_1 e y_2 que pertence à região R. Os pontos y_1 e y_2 são os pontos inicial e final do defeito D (a sua ordem pode mudar se os pontos do fecho são gerados em sentido horário ou anti-horário).

A *profundidade* de D é definida pelo segmento de reta y_fy_p onde y_f é um ponto do fecho convexo e y_p é o ponto mais distante no defeito D tal que o ângulo entre y_fy_p e y_1y_2 é 90 graus. A figura 3.9 mostra um exemplo do fecho convexo.

Finalmente, um *defeito* D é *ascendente* se o ponto mais profundo fica embaixo dos pontos inicial e final (o defeito mostrado na parte direita da figura 3.9 é um defeito ascendente). Um defeito é *descendente* se o ponto mais profundo fica encima dos pontos inicial e final. Os conceitos mostrados até agora são de utilidade na seção 3.2 para entender o fluxo de processamento de imagens proposto neste trabalho. Esse fluxo segmenta a imagem em linhas, palavras e caracteres e utiliza nas últimas duas etapas técnicas de estatística robusta para obter as regiões de interesse. Na continuação apresentamos um resumo dessas técnicas para conjuntos de dados uni-variados e multivariados.

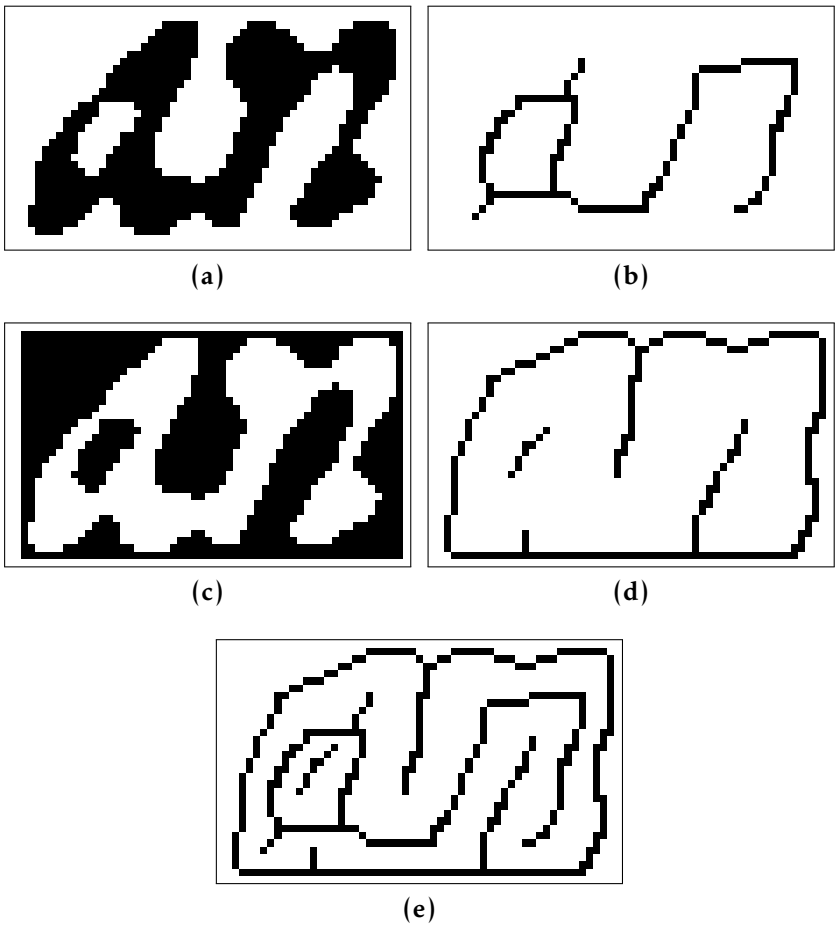


Figura 3.8: *Componentes conexas e os seus esqueletos utilizando o algoritmo de Hilditch. A imagem original, mostrada em (a), corresponde a uma componente conexa com dois caracteres. Em (b) mostramos o esqueleto dessa imagem enquanto que (c) e (d) mostram o esqueleto do background dessa mesma imagem. Finalmente em (e) mostramos a combinação de ambos os esqueletos. Este esqueleto será de utilidade para segmentar componentes conexas que contêm mais de um caractere.*

3.1.2 Estatística Robusta

A área de Estatística Robusta estuda técnicas para modelar conjuntos de dados que apresentam uma taxa considerável de observações atípicas, ou *outliers*. Essas técnicas se aplicam a conjuntos uni-variados e multivariados. Na continuação faremos uma breve revisão de técnicas de Estatística Robusta que foram usadas no nosso trabalho. A revisão esta baseada no trabalho de Maronna [MMY06].

Dados uni variados

Seja $\mathbf{x} = (x_1, x_2, ..., x_n)$ um vetor de valores observados. A media amostral $\bar{x}$ e o desvio padrão amostral (SD) s são definidos como

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i,\ s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

A média amostral é simplesmente a média aritmética do conjunto de dados, e como tal pode-se esperar que forneça uma boa estimativa do *centro* ou da *localização* dos dados.

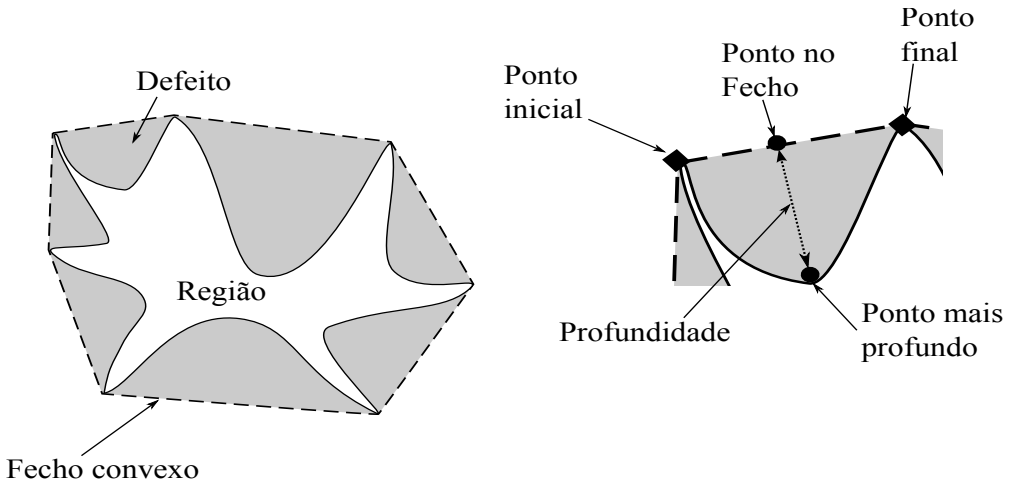


Figura 3.9: *Região mostrando o seu fecho convexo. Também são mostrados os defeitos entre a imagem e o fecho. A imagem da direita mostra em detalhe o defeito indicado na imagem da esquerda. Ali são mostrados os pontos inicial e final de um defeito. A profundidade de um defeito é o comprimento do segmento de reta que une o ponto mais profundo e um ponto no fecho de forma que esse segmento é perpendicular ao segmento de reta definida pelos pontos inicial e final.*

Da mesma forma, pode-se esperar que o desvio padrão s forneça uma boa estimativa da *dispersão* dos dados. No entanto, diante da presença de outliers as estimativas clássicas não são confiáveis.

Considere por exemplo a seguinte amostra com 24 observações sobre a quantidade de cobre em farinha integral (em partes por milhão), ordenadas de forma ascendente ([Com89]):

2.20	2.20	2.40	2.50	2.70	2.80	2.90
3.03	3.03	3.10	3.37	3.40	3.40	3.50
3.60	3.70	3.70	3.70	3.77	5.28	28.95

O valor 28.95 destaca-se do resto dos valores e poderia ser considerado como *outlier*. Os motivos que levaram a coletar esse valor podem ser diversos mas de qualquer forma é um valor que é bastante influente como vamos mostrar.

Os valores da média e SD amostrais para os dados acima são $\bar{x} = 4.28$ e $s = 5.30$, respetivamente. Já que $\bar{x} = 4.28$ é maior que os todos valores com exceção dos dois últimos, então ele não representa bem o centro dos dados. Se o valor 28.95 for removido, então os valores da média e SD amostrais são $\bar{x} = 3.21$ e $s = 0.69$. Agora a média amostral fornece uma boa estimativa do centro dos dados, e o SD é quase sete vezes menor comparado com o SD que contém os dados originais.

O exemplo sugere que uma forma simples de lidar com um outlier é detectá-lo e removê-lo do conjunto de dados. Existem muitos métodos para remover outliers (veja por exemplo [BL94]). Embora seja uma solução ela levanta uma série de perguntas, por exemplo: porque remover o outlier?, ou, quando uma observação é suficientemente "extrema" para ser removida?.

Existem no entanto outras estimativas à disposição que dispensam a remoção de outliers. Um método bastante antigo é o uso da *mediana amostral*. A mediana amostral t de um conjunto de dados é definida como:

$$t = Med(\mathbf{x}), \text{ se } |\{x_i > t\}| = |\{x_i < t\}|.$$

Podemos usar as *estatísticas de ordem* $(x_{(1)}, x_{(2)}, ..., x_{(n)})$ para definir a mediana amostral. As estatísticas de ordem são obtidas a partir da ordenação das observações $\mathbf{x} = (x_1, x_2, ..., x_n)$ de forma crescente

$$x_{(1)} \leq ... \leq x_{(n)}$$

Se n é impar, então $n = 2m - 1$ para algum inteiro m, e no caso $Med(\mathbf{x}) = x_{(m)}$. Se n é par, então $n = 2m$ para algum inteiro m, e qualquer valor entre $x_{(m)}$ e $x_{(m+1)}$ satisfaz a definição de media amostral, logo podemos tomar

$$Med(\mathbf{x}) = \frac{x_{(m)} + x_{(m+1)}}{2}.$$

No exemplo mostrado anteriormente, o valor da mediana amostral é 3.38, enquanto que, a mediana amostral sem o outlier é 3.37, mostrando que a mediana não é afetada pela presença desse valor.

Existe também uma versão robusta para o SD amostral conhecida como *desvio mediano absoluto sobre a mediana (MAD)*, definida como

$$MAD(\mathbf{x}) = MAD(x_1, x_2, ..., x_n) = Med\{|\mathbf{x} - Med(\mathbf{x})|\}$$

Essa estimativa usa a mediana amostral duas vezes, primeiro para estimar o centro dos dados e construir o conjunto de resíduos absolutos ao redor da mediana amostral, $\{|\mathbf{x} - Med(\mathbf{x})|\}$, e depois para estimar a mediana amostral para estes resíduos absolutos. Para poder utilizar o MAD como um estimador consistente de dispersão, definimos o *MAD normalizado* (MADN) como

$$MADN(\mathbf{x}) = \frac{MAD(\mathbf{x})}{0.6745}.$$

O valor 0.6745 é o MAD da variável aleatória de uma distribuição normal padrão.

Para o exemplo mostrado anteriormente obtemos $MADN = 0.53$, comparado com $s = 5.30$. Depois de remover o outlier obtemos $MADN = 0.50$, comparado com o valor de SD ligeiramente maior $s = 0.69$. O MAD é claramente não muito influenciado pela presença de um outlier com valor grande, e assim fornece uma alternativa robusta para o SD amostral.

Um medida tradicional para medir quão atípica é uma observação x_i com respeito a uma amostra é a proporção entre a sua distância da média e do SD amostrais

$$t_i = \frac{x_i - \bar{x}}{s} \tag{3.1}$$

O valor $|t_i|$ pode ser interpretado como o número de desvios padrão da media amostral, logo as observações com $|t_i| > 3$ são consideradas atípicas. A observação de maior valor nos dados do exemplo acima é $t_i = 4.65$, logo é considerado atípico. A regra clássica define que esses valores devem ser descartados ou ajustar eles para um valor dentro da faixa $\bar{x} \pm 3s$. Essa regra possui algumas dificuldades quando existem muitos outliers: elas podem ser "mascaradas", ou seja, elas podem influenciar as estimativas ao ponto que não é possível identificar outliers.

Considere o seguinte conjunto de dados que possui 20 observações do tempo (em microssegundos) necessários para a luz viajar uma distância de 7442 m ([Sti77]). Os tempos reais são os valores da tabela $\times 0.001 + 24.8$

28	26	33	24	34	-44	27	16	40	-2
29	22	24	21	25	30	23	29	31	19

A figura 3.10 mostra o diagrama Q-Q para esse conjunto de dados. Nele podemos observar que as duas observações -44 e -2 são atípicas. Os seus t_i's respetivos são -3.73 e -1.35 de maneira que o valor de $|t_i|$ para a observação -2 não indica que é um outlier. A razão -2 ter uma valor $|t_i|$ tão baixo é que ambas as observações puxam $\bar{x}$ para a esquerda e inflam s; se diz então que o valor -44 "mascara" o valor -2.

Para superar essa dificuldade podemos substituir $\bar{x}$ e s pelas medidas robustas de localização e dispersão. A versão robusta de t_i pode ser definida utilizando a mediana amostral e o MADN

$$t_i' = \frac{x_i - Med(\mathbf{x})}{MADN(\mathbf{x})}$$

Os t_i''s para as observações -44 e -2 são agora -11.73 e -4.64 e logo podemos dizer que elas são atípicas. Isto sugere que os procedimentos de detecção de outliers baseados em estimativas robustas são mais confiáveis.

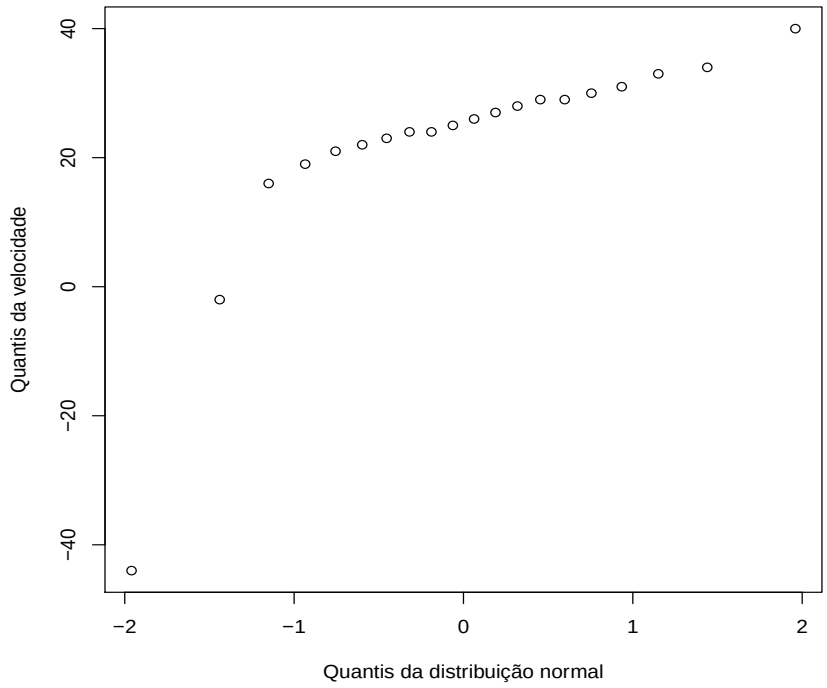


Figura 3.10: *Velocidade da luz: diagrama Q-Q para tempos observados*

Dados multivariados

A análise multivariada lida com situações em que muitas variáveis são medidas de forma independente. Em muitos casos de interesse se sabe, ou se supõe, que algum tipo de relação existe entre as variáveis, e portanto considerar cada uma de forma separada poderia gerar uma perda de informação.

Uma observação p-variada é um vetor $\mathbf{x} = (x_1, ..., x_p)' \in \mathbb{R}^p$ onde $\mathbf{x}'$ é a transposta de $\mathbf{x}$. Na abordagem clássica, a localização de uma variável aleatória p-variada é descrita pela expectância $\mu = E[\mathbf{x}] = (E[x_1], ..., E[x_p])'$ e a dispersão é descrita pela matriz de covariância

$$\mathbf{Var}(\mathbf{x}) = E[(\mathbf{x} - \mu)(\mathbf{x} - \mu)'].$$

Métodos multivariados clássicos de estimação são baseados na suposição de uma amostra de observações independente e identicamente distribuída (i.i.d) $X = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ onde cada $\mathbf{x}_i$ tem uma distribuição normal p-variada $N_p(\mu, \Sigma)$ com densidade

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{p/2}\sqrt{|\Sigma|}} exp(-\frac{1}{2}(\mathbf{x}-\mu)'\Sigma^{-1}(\mathbf{x}-\mu)) \tag{3.2}$$

onde $\Sigma = \mathbf{Var}(\mathbf{x})$ e $|\Sigma|$ é o determinante de Σ.
Sabe-se que sob a distribuição normal 3.2, as estimativas de μ e Σ para uma amostra $\mathbf{x}$ são respetivamente a média amostral $\hat{\mu}(X)$ e a matriz de covariância amostral $\hat{\Sigma}(X)$

$$\hat{\mu}(X) = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i, \quad \hat{\Sigma}(X) = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{x}_i - \hat{\mu}(X))(\mathbf{x}_i - \hat{\mu}(X))'.$$

Em uma ou duas dimensões, é fácil detectar *outliers* mediante o uso de diagramas dos dados observados, mas a detecção é mais difícil em dimensões maiores. Uma forma de identificar possíveis observações p-variadas atípicas é calcular a distância de cada observação $\mathbf{x}$ com respeito à estimativa de localização μ.

A distância quadrática de Mahalanobis (DM) entre a observação $\mathbf{x}$ e μ com respeito à matriz Σ é definida como

$$DM(\mathbf{x}, \mu, \Sigma) = (\mathbf{x}-\mu)'\Sigma^{-1}(\mathbf{x}-\mu))$$

Logo, a medida multivariada para medir quão atípica é a observação $\mathbf{x}_i$ é

$$DM_i = DM(\mathbf{x}_i, \hat{\mu}(X), \hat{\Sigma}(X)).$$

Se sabe que se $\mathbf{x} \sim N_p(\mu, \Sigma)$ então $DM(\mathbf{x}, \mu, \Sigma) \sim \chi^2_p$ ([Seb04]). Se a média $\hat{\mu}(X)$ e covariância $\hat{\Sigma}(X)$ amostrais são próximas dos seus valores verdadeiros, podemos examinar o diagrama Q-Q das distâncias DM_i versus os quantis da distribuição χ^2_p e remover as observações para as quais DM_i é maior que uma nota de corte. Geralmente, é escolhido o quantil 97.5 da distribuição χ^2_p como nota corte.

No entanto, em conjuntos de dados com aglomerações de observações com outliers, a distância DM colapsa ([PS36]). Assim como nos dados uni-variados, o problema de mascaramento ocorre em dados multivariados com múltiplos *outliers* ou aglomerações de *outliers*.

Considere, por exemplo, o conjunto de dados sintéticos de Hawkins, Bradu e Kaas (HBK)([HBK84]). Este conjunto contém 75 observações de dimensão três e foi construído para mostrar o efeito do mascaramento em dados multivariados. A figura 3.11 mostra o diagrama Q-Q para os quantis das distâncias de Mahalanobis DM_i versus os quantis de χ^2_3 para o conjunto de dados HBK. Podemos observar somente um outlier apesar de que 14 outliers tenham sido inseridos. Este problema se deve à utilização da média e covariância clássicas do conjunto inteiro resultando no mascaramento das outras 13 observações.

Problemas como o mascaramento podem ser resolvidos utilizando estimativas robustas das medidas de localização e dispersão dado que estas não são afetadas pelos outliers. Para obter essas estimativas podemos utilizar distintos algoritmos existentes na literatura estatística sendo o Determinante Mínimo de Covariância (DMC) [Rou84, RL87] um dos mais conhecidos. Este algoritmo, descoberto em 1984, encontra um subconjunto $J \subset X$ com as seguintes características:

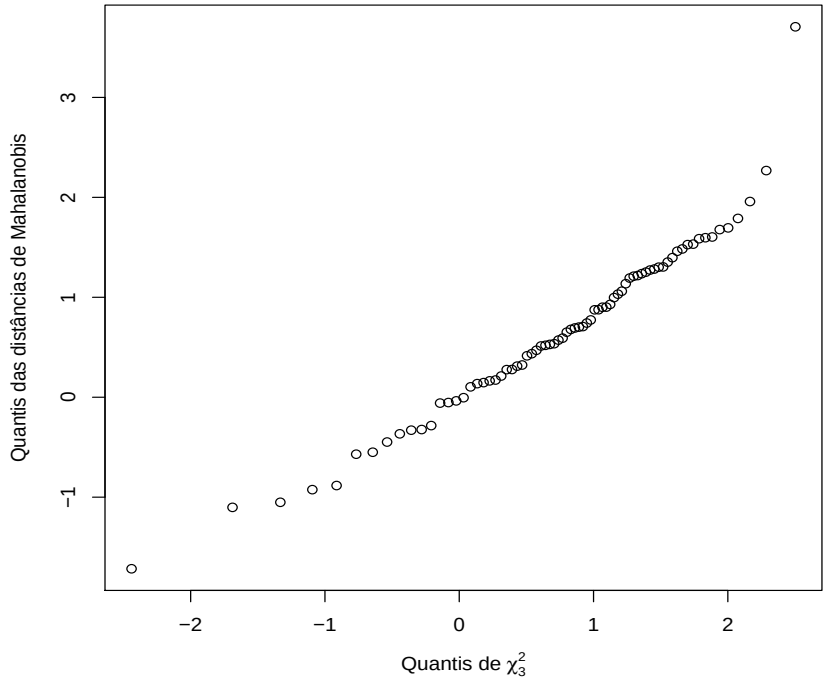


Figura 3.11: *Distâncias quadráticas de Mahalanobis para o conjuntos de dados HBK versus as estatísticas de ordem χ^2_3 utilizando a média e covariância amostrais tradicionais. Os dados incluem 14 outliers que são mascaradas utilizando as medidas tradicionais. Os valores mostrados são os logaritmos para facilitar a visualização.*

- O tamanho $h = |J|$ é igual a uma proporção do tamanho $n = |X|$ do conjunto X, isto é, $h = k * n$ onde k é um parâmetro definido pelo usuário e possui um valor dentro do intervalo $[0.5, 1]$
- A determinante $\Sigma(J)$ é mínima com respeito a todos os subconjuntos de X

O algoritmo de busca não é trivial pois é necessário avaliar um número exponencial de subconjuntos de X até encontrar aquele que cumpre com a condição do DMC.

Essa dificuldade foi superada somente em 1999 no trabalho de Rousseeuw e Van Driessen [RvD99], o que possibilitou a sua aplicação em áreas como controle de qualidade, medicina, finanças, análise de imagens e química (veja [HRA08, HRV12]).

O DMC do conjunto X pode ser representado por uma tupla que contém o subconjunto $J \subset X$ assim como as suas medidas de localização e dispersão:

$$DMC(X) = (J, \hat{\mu}(J), \hat{\Sigma}(J))$$

onde

$$J \subset X \text{ e } |J| = h \text{ e } \forall K \subset X, |K| = h \text{ e } |\hat{\Sigma}(J)| < |\hat{\Sigma}(K)|$$

A figura 3.12 mostra o efeito da utilização do DMC para determinar outliers. Essa figura mostra o diagrama Q-Q para os quantis das distâncias de Mahalanobis versus os quantis da distribuição χ^2_3 para o conjunto de dados HBK utilizando o estimador robusto DMC. Observe que, a diferença da figura 3.11, agora podemos observar claramente os 14

pontos que foram inseridos de forma artificial no conjunto HBK e que correspondem com valores extremos.

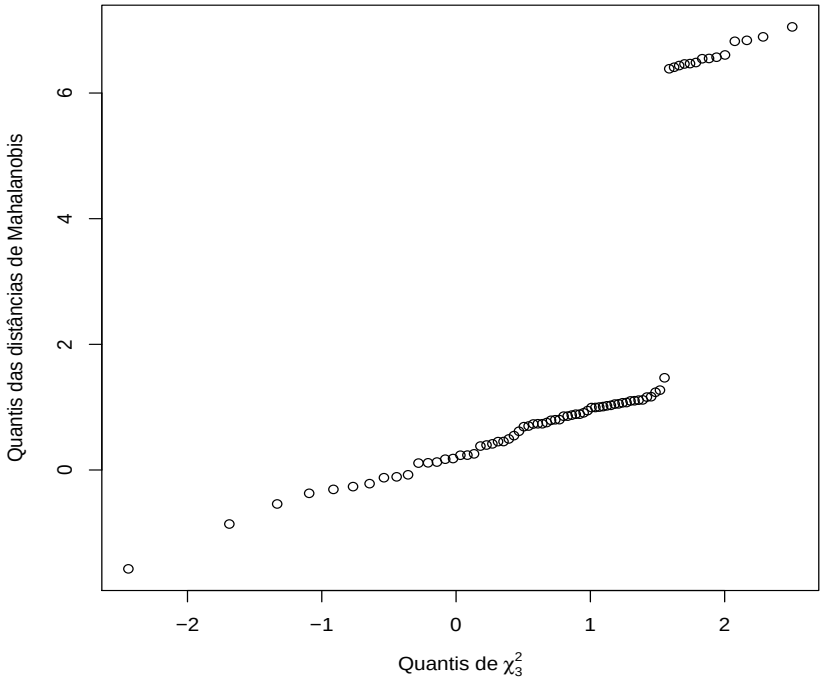


Figura 3.12: *Distâncias quadráticas de Mahalanobis para o conjuntos de dados HBK versus as estatísticas de ordem χ^2_3 utilizando a média e covariância robustas computadas usando o DMC. Os 14 outliers do conjunto de dados são facilmente visíveis. Os valores mostrados são os logaritmos para facilitar a visualização.*

No entanto, mesmo obtendo as estimativas robustas, ainda pode não ficar claro qual é o quantil mais apropriado da distribuição X^2_p para detectar *outliers*. A explicação para este fenômeno é que a distribuição das distâncias de Mahalanobis não converge muito rápido para a distribuição chi-quadrado [RvZ91] o que provoca a remoção excessiva de observações.

Em [Har00] foi mostrado que a distribuição F aproxima a distribuição das distâncias de Mahalanobis de maneira mais exata, logo é possível usá-la para obter uma nota de corte mais precisa. Especificamente, dadas duas constantes conhecidas m e c obtidas por simulação, podemos ajustar a distribuição das distâncias de Mahalanobis para ela se aproximar de uma distribuição F com graus de liberdade p e $m-p+1$

$$\alpha \cdot DM(\mathbf{x}, \hat{\mu}(J), \hat{\Sigma}(J)) \sim F_{(p,m-p+1)},$$

onde $\alpha = \frac{c(m-p+1)}{mp}$ é o fator de ajuste.

Assim o critério de corte se baseia na comparação da distância de Mahalanobis ajustada com um quantil da distribuição $F_{(p,m-p+1)}$ (geralmente esse quantil é 97.5).

3.2 Segmentação de imagens

Nesta seção apresentamos o método de segmentação utilizado no nosso sistema para processar documentos históricos. Este método se baseia no trabalho de Nikolau et.al [NMG+10] e segue o fluxo mostrado na figura 3.13.

A diferença para outros métodos se concentra na utilização do algoritmo ARLSA e na segmentação de caracteres conexos sendo que ambos são específicos para tratar alguns casos de imagens de documentos históricos. Estes casos incluem, por exemplo, linhas levemente inclinadas.

Nas próximas seções explicaremos as etapas do método mostrando em cada uma as imagens produzidas por nosso sistema.

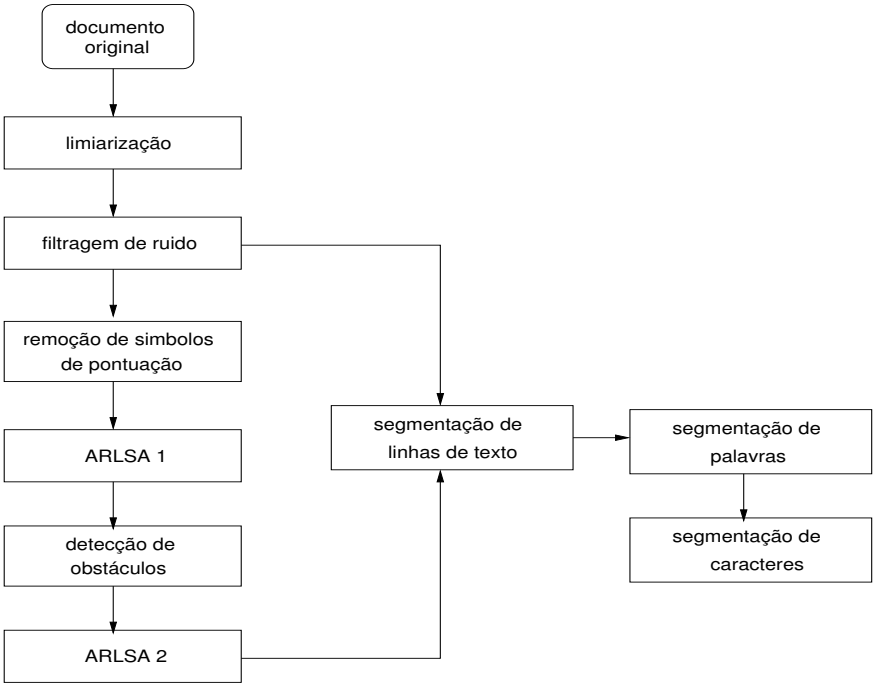


Figura 3.13: *Fluxo do algoritmo original de segmentação de documentos históricos (adaptado de [NMG+10])*

3.2.1 Adaptive Run Length Smoothing Algorithm

Dentro da literatura de processamento de imagens, existem muitos algoritmos para análise do *layout* de imagens digitais sendo que os algoritmos de varredura são os mais comuns. Dentre estes, o RLSA, proposto por Wahl ([WWC82]) é um dos mais utilizados.

Dada uma imagem digital R de tamanho $m \times n$, o algoritmo identifica *sequências horizontais* ou *sequências verticais* de pixels. Uma sequência horizontal S é uma tupla de tamanho (comprimento) $L(S) = p + 1$ definida da seguinte forma:

$$S = (R[j,a], R[j,a+1], ..., R[j,a+p]),$$

onde cada elemento $R[j,k]$ é um pixel localizado na linha j e da coluna $a \leq k \leq a+p$. De forma análoga, uma *sequência vertical* de tamanho $p+1$ é definida da seguinte forma:

$$S = (R[b,i], R[b+1,i], ..., R[b+p,i]),$$

onde $R[l,i]$ é um pixel localizado na linha $b \leq l \leq b+p$ e coluna i.

Definimos ainda uma *sequência de foreground*, vertical ou horizontal, como aquela onde todos os seus pixels $R[i,j]$ têm valor 1. De forma análoga, uma *sequência de background* é aquela onde todos os seus pixels tem valor 0.

Assim, dada uma sequência horizontal de pixels de background, o algoritmo RLSA verifica se o seu comprimento é menor que o limiar T_{max}. Se for, então a sequência de background é convertida para uma de foreground produzindo na imagem regiões homogêneas similares ao efeito de marcar uma linha utilizando uma caneta marca-texto.

Este algoritmo não se comporta bem quando existe muita irregularidade entre as componentes conexas que compõem uma linha (por exemplo uma linha com uma leve inclinação). Em [NMG+10] foi proposta uma versão adaptativa, o ARLSA, que supera as deficiências do algoritmo original.

No ARLSA são introduzidas algumas restrições para contornar casos como inclinação de linhas ou fontes com tamanhos diferentes. A entrada do algoritmo é uma imagem binária assim como as suas componentes conexas.

Para cada componente conexa CC_i são considerados dois tipos de sequências maximais de pixels de background: sequências internas (aquelas que ocorrem dentro da componente) e sequências externas (aquelas que ocorrem entre duas componentes conexas consecutivas CC_i e CC_j). A figura 3.14 mostra estes dois tipos de sequências maximais de background.

Figura 3.14: *Exemplo dos dois tipos de sequências maximais de background consideradas no ARLSA: (a) sequências internas e (b) sequências externas (adaptado de [NMG+10])*

A identificação de sequências maximais de background é importante para poder localizar as linhas dentro da imagem. Esta identificação é simples e imediata no caso de sequências internas. No entanto, para sequências externas, é necessário avaliar 4 medidas que determinam se uma sequência externa (de background) pode ser convertida para uma de foreground. Dada uma sequência externa S_{ij} que ocorre entre duas componentes conexas consecutivas CC_i e CC_j, são calculadas as seguintes medidas:

1. $L(S)$: o comprimento da sequência externa S_{ij};

2. $H_R(S)$: proporção entre as alturas das componentes CC_i e CC_j definida como

$$H_R(S) = \frac{max\{h_i, h_j\}}{min\{h_i, h_j\}}$$

   , onde h_i, h_j são as alturas de CC_i e CC_j, respectivamente;

3. $O_H(S)$: Representa a sobreposição horizontal entre os dois retângulos que contêm a CC_i e CC_j. Sejam (Xl_i, Yl_i) e (Xr_i, Yr_i), respectivamente, as coordenadas das esquinas superior esquerda e inferior direita do retângulo que contêm a CC_i. De forma

análoga, definimos as coordenadas (Xl_j, Yl_j) e (Xr_j, Yr_j) para a componente CC_j. Logo, a sobreposição horizontal $O_H(S)$ está definida pela seguinte fórmula

$$O_H(S) = max\{Yl_i, Yl_j\} - min\{Yr_i, Yr_j\}$$

A sobreposição horizontal está representada na figura 3.15 e podemos observar que, quando $O_H(S) < 0$, existe sobreposição entre as duas componentes conexas CC_i e CC_j.

4. $N(S)$: uma função de saída binária. O valor dela é 0, quando na vizinhança 3×3 de pelo menos um pixel na sequência S_{ij} existe uma terceira componente CC_k onde $k \neq i, j$. Caso contrário, o valor é 1. Como é mostrado na figura 3.16, na vizinhança 3×3 das sequências de pixels cinza não existe outro objeto além de CC_i e CC_j. Neste caso, $N(S)$ tem valor 1. No caso das sequências de pixels brancos, a regra não se cumpre e $N(S)$ tem valor 0.

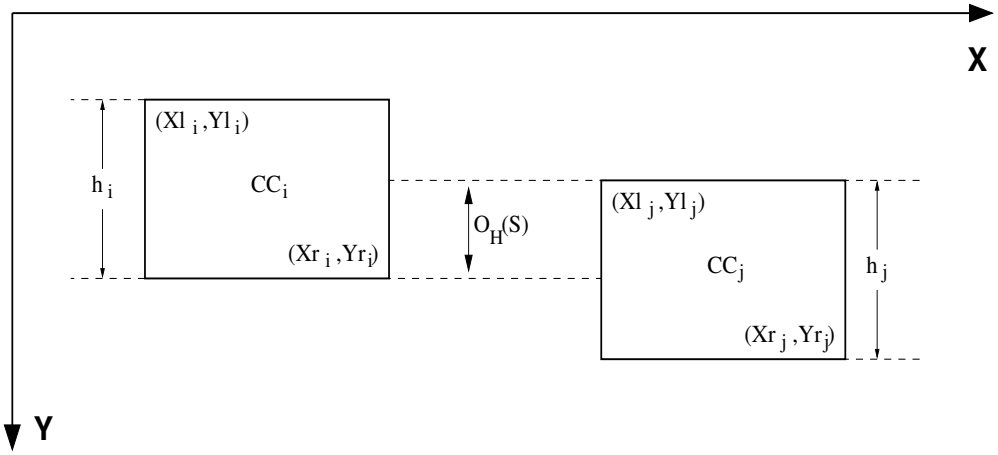


Figura 3.15: *Sobreposição horizontal de duas componentes conexas CC_i e CC_j (adaptado de [NMG+10])*

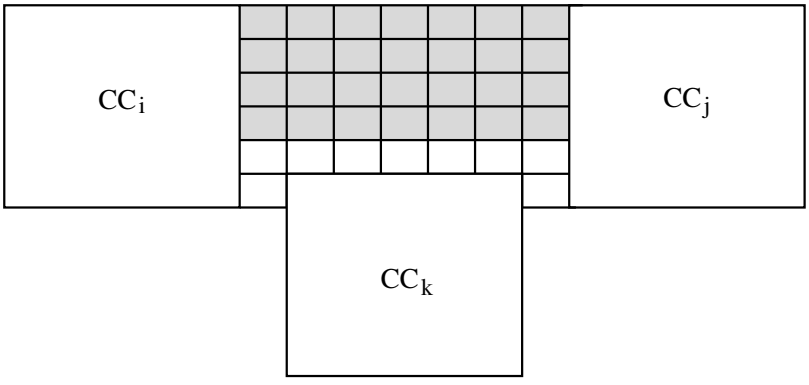


Figura 3.16: *Restrições sobre sequências externas baseadas na função $N(S)$. Os pixels em cinza representam os pixels de background que serão substituídos por pixels de foreground (adaptado de [NMG+10])*

Baseado nas medidas acima mencionadas, o algoritmo substitui a sequência S_{ij} por pixels de foreground se a seguinte condição for verdadeira:

$$(L(S) \leq T_l) \wedge (H_R(S) \leq T_h) \wedge (O_H(S) \geq T_O) \wedge (N(S) = 1),$$

onde T_l, T_O são limiares computados localmente e T_h é um limiar global.
O limiar de comprimento T_l de cada sequência está relacionado com as alturas das componentes conexas. Sejam h_i e h_j as alturas de CC_i e CC_j, respectivamente. Então:

$$T_l = a * min\{h_i, h_j\}$$

, onde a é um parâmetro definido pelo usuário.

O limiar T_h controla a proporção de altura entre dois caracteres. Comumente este valor varia entre 2 e 3 considerando a altura do menor caractere que é um "o" minúsculo e o ascendente (ou descendente) de um caractere do mesmo tamanho. No trabalho original, observou-se que o valor ótimo era 3.5 pois permite contemplar caracteres com tamanho variável. No entanto, neste trabalho foi observado que existem alguns casos onde este valor junta erroneamente duas linhas de texto. Esta limitação do algoritmo original para os dados que utilizamos possui uma solução simples que é descrita na seção 3.5.

O limiar T_o controla a superposição horizontal entre as componentes CC_i e CC_j e é expresso pela porcentagem de superposição com respeito à componente de menor altura:

$$T_O = c * min\{h_i, h_j\},$$

onde c é um parâmetro definido pelo usuário. Por exemplo, quando c é igual a 0.4, o algoritmo considera que pelo menos 40% da altura da componente menor deve ser coberta para que a sequência externa possa ser substituída.

A restrição sobre a função $N(S)$ garante que a sequência externa será substituída por pixels de foreground caso não exista uma componente na vizinhança 3×3 dos pixels que conformam a sequência. O seu propósito é prevenir a criação de sequências externas inadequadas que depois resultam na criação de grupos de componentes de objetos não desejados. Isto é muito útil para evitar juntar caracteres que pertencem a linhas diferentes e no entanto estão superpostos, tal como ocorre com frequência em textos antigos.

A figura 3.17a mostra a imagem de um documento impresso no ano 1623 de autoria do Frade irlandês Luca Wadding enquanto que a figura 3.17b mostra a aplicação do ARLSA na imagem original.

Nas próximas seções continuaremos explicando as outras etapas mostradas no esquema da figura 3.13.

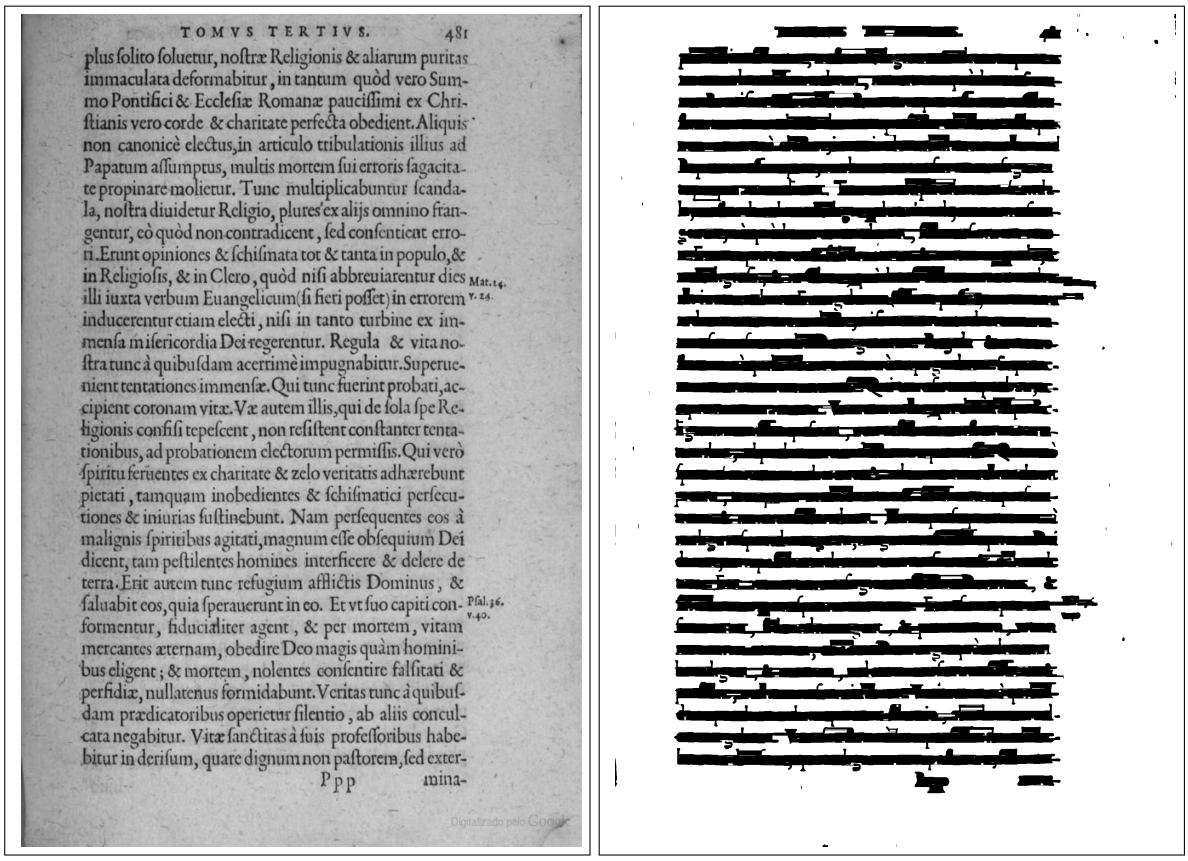
3.3 Limiarização e Remoção de ruído

Nesta seção mostramos o resultado da limiarização da imagem original assim como a remoção dos símbolos de pontuação.

Limiarização

Especificamente, no caso da limiarização, foi implementado o algoritmo proposto por Gatos([GPP06]). Este algoritmo utiliza um procedimento de limiarização local, ou seja, avalia a vizinhança de cada pixel para determinar o limiar de limiarização. A intuição por trás do algoritmo é obter as imagens de background e foreground de interesse, ou seja, fornecer um critério que permita diferenciar as regiões com texto daquelas que pertencem ao background da imagem [2].

[2]Foi também implementado o filtro bilateral ([TM]) e tivemos resultados bons na remoção de ruído mas, infelizmente, este filtro aumenta a grossura dos caracteres dificultando a segmentação de linhas e



(a) *Imagem original*

(b) *Resultado da aplicação do ARLSA com* $a = 1.5$, $T_h = 3.5$, $c = 0.4$

Figura 3.17: *As imagens acima mostram o antes e depois da aplicação do ARLSA em uma imagem do século 17 (a imagem foi previamente limiarizada).*

A figura 3.18 mostra a aplicação do filtro de Gatos na imagem de exemplo. Nela são mostrados as imagens de foreground e background calculadas pelo algoritmo para finalmente produzir a imagem binária.

Remoção de ruído

Depois de produzir a imagem binária o algoritmo continua com a remoção de ruído. Esta etapa começa com o cálculo das alturas das componentes conexas da imagem para logo obter o histograma destas alturas. A mediana das alturas AH é obtida a partir deste histograma (veja [KGN+07]) e serve como ingrediente principal no cálculo de três medidas para todas as componentes conexas. Estas medidas verificam se uma componente conexa pode ser classificada, ou não, como ruído e são descritas a seguir:

- A altura da componente conexa deve ser menor que $cf * AH$. O parâmetro cf é definido pelo usuário. O valor deste parâmetro geralmente permanece fixo em 1/3 pois é bastante comum que o ruído tenha altura menor que 1/3 da altura mediana.

- O alongamento $E(CC)$ da componente conexa deve ser menor que o parâmetro T_E (definido pelo usuário) para ser considerado como ruído. De forma precisa, o alon-

caracteres.

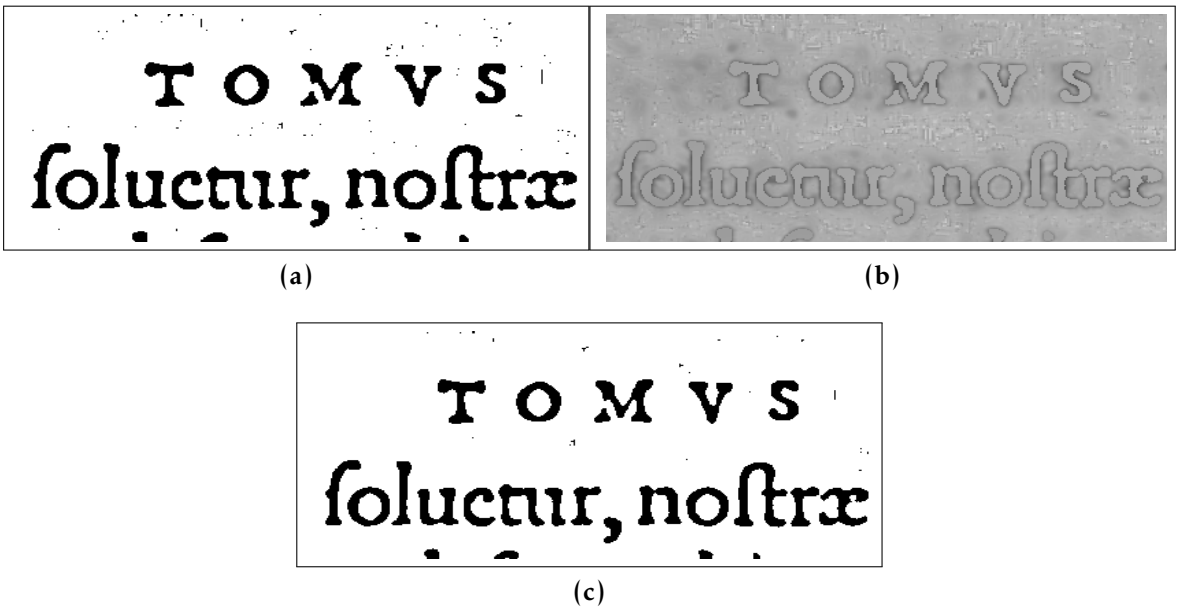


Figura 3.18: *Efeito da aplicação do filtro de Gatos. A figura (a) mostra o cálculo da imagem de foreground. A figura (b) mostra a obtenção da imagem de fundo. Finalmente, a imagem (c), mostra a imagem binária final com base nas duas imagens (a) e (b).*

gamento está definido como

$$E(CC) = \frac{min(H(CC, W(CC))}{max(H(CC, W(CC))},$$

onde $H(CC)$ e $W(CC)$ correspondem à altura e largura da componente CC, respectivamente.

- Finalmente é calculada a densidade $D(CC)$. Ela está definida como

$$D(CC) = \frac{P_{num}(CC)}{BB_{size}(CC)},$$

onde $P_{num}(CC)$ é o número de pixels foreground e $BB_{size}(CC)$ é o número total de pixels da componente CC. Para considerar a componente conexa como ruído, a sua densidade deve ser menor que o parâmetro T_D definido pelo usuário.

Para mostrar o efeito dessas métricas, observe a figura 3.19 onde comparamos um fragmento da imagem limiarizada antes e depois da remoção de ruído.

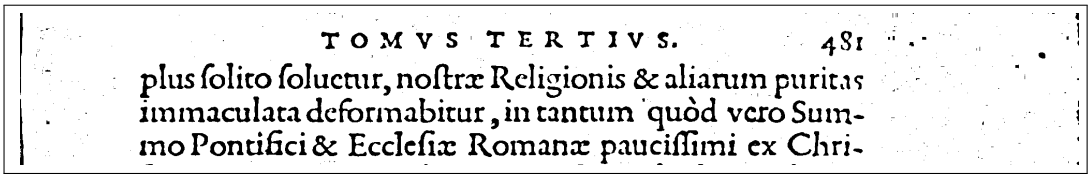
3.3.1 Segmentação de Caracteres de pontuação

Depois da remoção do ruído o algoritmo continua com a segmentação dos caracteres de pontuação. Para cada componente conexa que não foi removida pela filtragem de ruído é calculada a proporção de pixels pretos contidos nela antes e depois da execução do ARLSA. Este critério se baseia na hipótese de que símbolos de pontuação têm pouca variação depois da aplicação do ARLSA e eles tendem a se manter isolados.

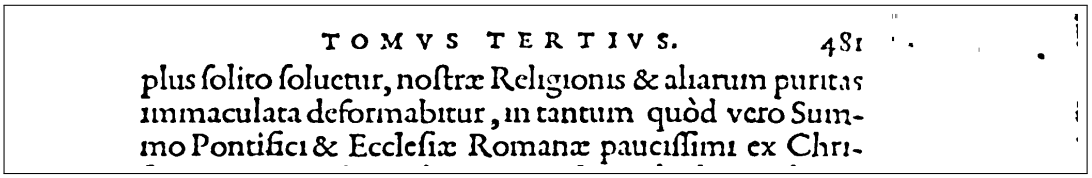
De maneira formal, a proporção P_R que mede esta variação está definida como

$$P_R = \frac{P_O}{P_C} \leq T_R,$$

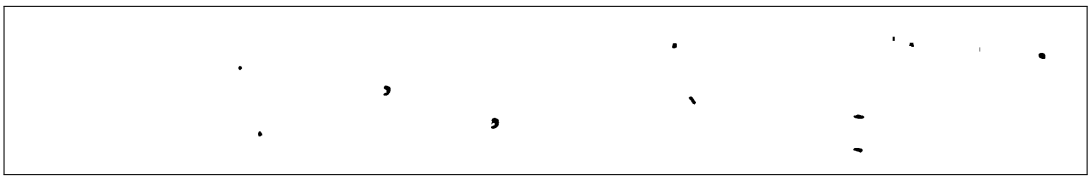
onde P_O corresponde ao número de pixels pretos na componente conexa antes do ARLSA e P_C contabiliza o número de pixels depois da aplicação do ARLSA. O parâmetro T_R é definido pelo usuário. Por exemplo, se $T_R = 1.3$ o sistema espera que exista uma diferença de 30% nas componentes que representam caracteres de pontuação antes e depois do ARLSA. A figura 3.19c mostra um exemplo desta etapa.



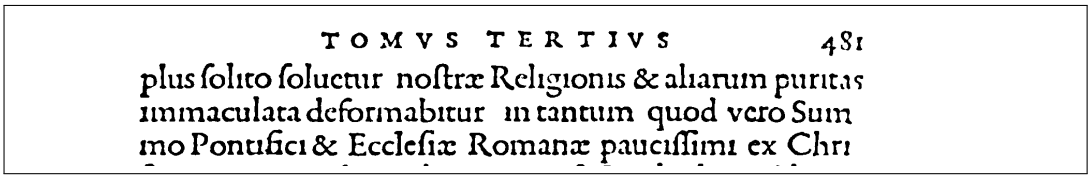
(a) *Imagem limiarizada com ruído*



(b) *Imagem sem ruído*



(c) *Imagem com símbolos de pontuação segmentados*



(d) *Imagem sem símbolos de pontuação*

Figura 3.19: *A figura (a) mostra a imagem original antes de remover o ruído. Observe a componente à esquerda da imagem que é filtrado na imagem (b) onde foi aplicado o ARLSA. Os parâmetros utilizados na remoção foram $T_E = 0.05$ e $T_D = 0.08$. Na figura (c) foram segmentados os símbolos de pontuação com $T_R = 1.3$ e na figura (d) são mostrados somente os objetos que correspondem com caracteres.*

Na continuação serão detectados os obstáculos e depois as linhas. A imagem com os caracteres de pontuação será reutilizada na etapa de segmentação de linhas.

3.4 Segmentação de obstáculos

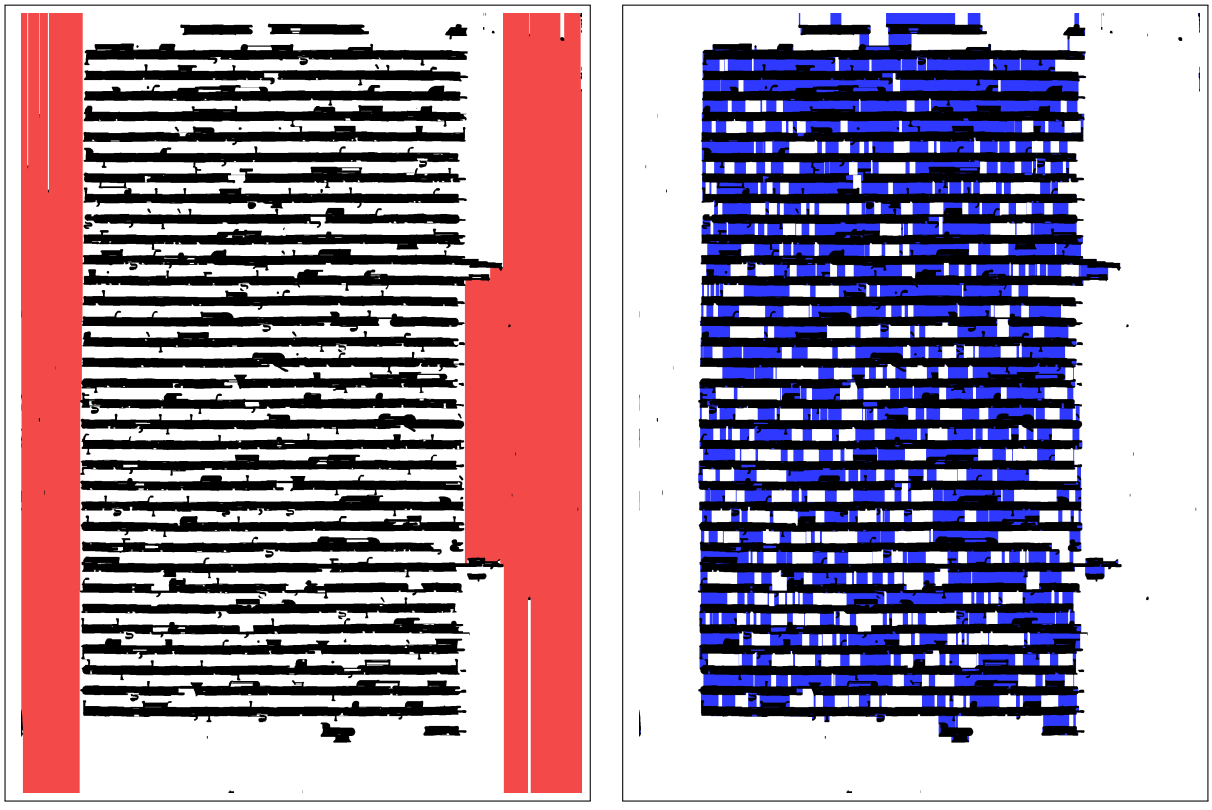
O propósito da segmentação de obstáculos é isolar diferentes colunas e linhas de texto definindo regiões dentro do documento que o algoritmo de varredura não poderá atravessar. O algoritmo considera dois tipos de obstáculo: colunas; linhas de texto. Obstáculos

do tipo coluna são extraídos para separar diferentes regiões de texto como colunas enquanto que obstáculos de linhas de texto ajudam na localização de regiões de linhas de texto que pertencem à mesma coluna.

Depois da execução do ARLSA, da remoção de ruído e dos caracteres de pontuação é calculado o histograma $h_v(w)$ dos comprimentos das sequências verticais de background. Se um documento tem colunas de texto ou bordas então vão aparecer sequências verticais de grande comprimento. Aquelas sequências verticais com comprimento maior que $k \times H$ são considerados como obstáculos do tipo coluna. A altura da imagem é representada pela variável H enquanto que o parâmetro k é definido pelo usuário. Embora k seja definido pelo usuário, o valor dele quase sempre é 1/3.

Com este valor evitamos a criação de obstáculos entre objetos que pertencem à mesma linha e detectamos ainda o espaço entre diferentes colunas. A figura 3.20a mostra a detecção de colunas.

No caso de obstáculos de linhas, aquelas sequências verticais de background com comprimento menor a $M_v = argmax\ h_v(w)$ representam os obstáculos de linhas. A figure 3.20b mostra a detecção desse tipo de obstáculo.



(a) *Detecção de obstáculos de colunas* **(b)** *Detecção de obstáculos de linhas*

Figura 3.20: *Exemplo de detecção de obstáculos*

3.5 Segmentação de linhas

A segmentação de linhas de texto é feita mediante a execução do ARLSA na imagem original depois da remoção do ruído e dos caracteres de pontuação (veja a figura 3.19). No entanto nesta etapa o ARLSA é restringido pelos obstáculos de coluna e texto obtidos na etapa anterior e configurado com um valor maior do parâmetro a para poder conectar

componentes de uma mesma linha[3]. Finalmente, cada componente conexa que representa um caractere de pontuação é conectada com a linha mais próxima. A figura 3.21 mostra o resultado desta etapa.

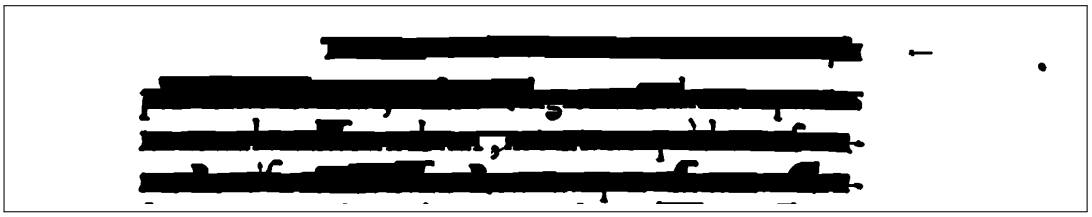


Figura 3.21: *Linhas segmentadas com símbolos de pontuação*

Como consequência das condições de conectividade entre componentes conexas, especificamente do limiar T_l que determinar se duas componentes devem ser conectados, o ALRSA tende a juntar linhas que devem permanecer separadas e isto acontece quando a distância entre as linhas é pequena. Se este cenário não parece frequente em textos em grego usados pelos autores, ele é bastante frequente em documentos históricos que fazem uso de letras latinas minúsculas pois era muito natural aproveitar ao máximo o papel ao confeccionar os livros. A figura 3.22 mostra esse tipo de cenário na nossa imagem de exemplo.

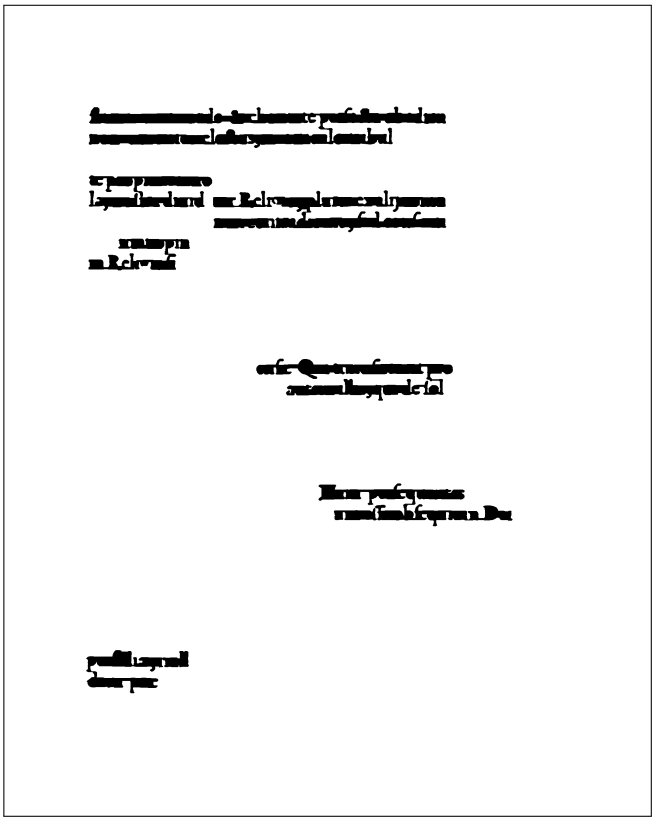


Figura 3.22: *Erros de segmentação de linhas do ARLSA. Estas componentes conexas são segmentados utilizando o histograma que contabiliza a cada posição no eixo vertical o número de pixels pretos no eixo horizontal. Logo são detectados os vales no histograma para quebrar as linhas.*

[3]Este parâmetro afeta o limiar T_l que controla o comprimento da sequência externa. Se o comprimento for maior a T_l a sequência de pixels de background é trocada por uma de foreground

Para superar esta limitação do ARLSA foi adicionada uma pequena etapa que segmenta de forma correta estas componentes. O algoritmo que corrige este comportamento utiliza uma técnica bastante comum na segmentação tradicional de linhas. Esta técnica se baseia na hipótese de que se for feita uma varredura vertical da imagem contabilizando a cada ponto o número de pixels pretos na horizontal teremos um histograma com vários vales e picos onde os vales indicam a separação das linhas. Assim obtemos uma segmentação correta das componentes conexas que contêm várias linhas.

3.6 Segmentação de palavras

A segmentação de palavras é realizada em cada linha de forma independente começando pelo cálculo das suas componentes conexas. Estas são ordenadas pela sua abscissa x e é construído o vetor de valores $\mathbf{h}$ que armazena as distâncias horizontais entre componentes adjacentes. Se o valor for negativo ela é assume o valor zero.

Essas distâncias $\mathbf{h}$ possui dois tipos de observações: valores pequenos pertencem a distâncias entre dois caracteres e valores grandes pertencem à separação entre palavras. Assim, podemos utilizar a técnica de detecção de ruído para dados uni-variados definida na seção 3.1.2.

Seja $h'_i = (\mathbf{h}[i] - Med(\mathbf{h}))/MADN(\mathbf{h})$ onde Med e $MADN$ são a mediana e o MAD normalizado respectivamente. Logo, se $|h'_i|$ for maior ou igual a 2.1 então uma nova palavra é criada.

A figura 3.23 mostra uma linha onde foi aplicada essa técnica. Ela mostra uma caixa para cada palavra que foi identificada. Observe que ela segmenta de forma satisfatória para a maioria dos casos nessa figura. No entanto, também comete erros sobretudo quando os caracteres foram impressos com muito espaço entre eles. Este é o caso das últimas 4 palavras.

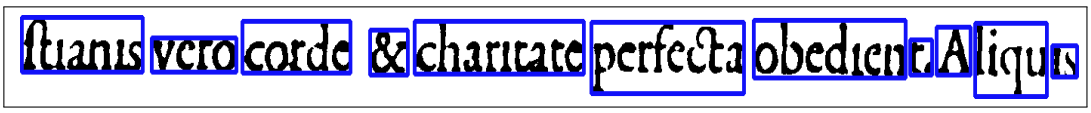


Figura 3.23: *Exemplo de segmentação de palavras. O algoritmo comete alguns erros porque houve muita distância entre dois caracteres.*

3.7 Segmentação de caracteres

Nesta etapa a página possui somente componentes conexas que possuem caracteres. Dependendo da imagem que foi recebida como entrada, as componentes conexas podem apresentar caracteres de quatro tipos: *minúsculos*, *maiúsculos*, *diminutos* e *ligações*. Um *caractere diminuto* é um que possui tamanho consideravelmente comparado a caractere minúsculo. Por exemplo, este tipo de caractere é geralmente atribuído as componentes conexas que possuem notas de rodapé. Finalmente, uma *componente conexa com ligações* possui dois ou mais caracteres minúsculos conexos por alguma região. Este tipo de componente conexa apresenta vários caracteres minúsculos ligados por regiões. Estas ligações são produto da impressão do documento original ou da digitalização.

A obtenção das componentes conexas que não possuem caracteres com ligações, isto é minúsculos, maiúsculos e diminutos, é simples. No entanto, as componentes conexas que possuem caracteres com ligações, apresentam um problema mais difícil de ser resolvido.

A dificuldade reside no fato de obter um critério que encontre de forma satisfatória as regiões da componente conexa que constituem caracteres minúsculos mesmo quando a sua tipografia seja desconhecida.

Dentro das propostas de solução para este problema, existem soluções que utilizam algoritmos de programação dinâmica junto com modelos supervisionados dos caracteres minúsculos para obter a segmentação "ótima" [SPD10, PPLD12]. Em alguns casos, como em [PPLD12], o critério de otimalidade é definido pela soma da verosimilhança de sub-imagens da componente conexa com um caractere minúsculo. A verosimilhança é computada a partir de um classificador treinado previamente com a tipografia da imagem original.

Neste trabalho propomos um algoritmo de programação dinâmica que utiliza um modelo não supervisionado para obter a segmentação ótima de uma componente conexa com caracteres ligados. O modelo não supervisionado descreve as características dos caracteres minúsculos da página e utiliza técnicas de estatística robusta para computá-los. O algoritmo, por sua vez, quebra a componente conexa com caracteres ligados em sub-imagens fazendo uso de *caminhos de segmentação*. As sub-imagens recebem uma pontuação com base no modelo não supervisionado e a solução ótima é obtida mediante a computação da média harmônica dessas pontuações.

Nas seguintes seções descrevemos o método que obtém o modelo não supervisionado. Em seguida mostramos a obtenção dos caminhos de segmentação para finalmente mostrar o algoritmo de programação dinâmica.

3.7.1 Estatísticas robustas de caracteres

Seja CC o conjunto de componentes conexas obtidos até agora e $cc \in CC$ uma componente conexa individual. Para cada componente cc obtemos as seguintes características:

Altura Altura do retângulo que contém a componente conexa;

Largura Largura do retângulo que contém a componente conexa;

Comprimento do perímetro Tamanho do perímetro da componente conexa;

Número pixels de foreground Número de pixels de background dentro do retângulo que contém a componente conexa;

Número de pixels de background Número de pixels de foreground dentro do retângulo que contém a componente conexa;

Número de defeitos ascendentes Os defeitos que possuem profundidade maior ou igual que metade da componente conexa são coletados de forma que, o número de defeitos ascendentes seja contabilizado;

Número de defeitos descendentes De forma análoga aos defeitos ascendentes, contabilizam-se os defeitos descendentes;

Estas características foram obtidas através de um processo de seleção de características. O nosso critério de seleção foi o de obter o maior número de caracteres minúsculos ao criar o modelo não supervisionado.

A figura 3.24 mostra as características para uma componente conexa que possui caracteres ligados.

As componentes conexas e suas características são processadas em duas etapas para obter subconjuntos para os quatro tipos de caractere.

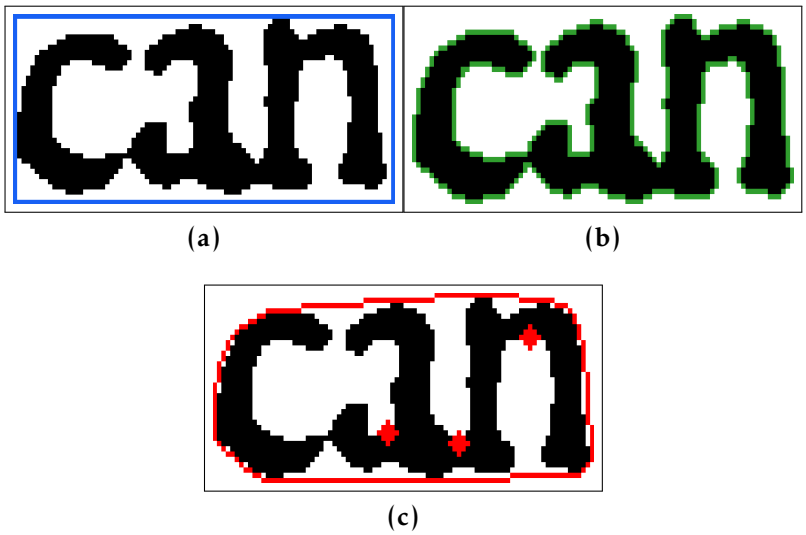


Figura 3.24: *Características de uma componente conexa. Na figura (a) o retângulo em azul indica a largura e altura da componente. Dentro desse retângulo, é contabilizado o número de pixels de background e foreground. Na figura (b) o perímetro é o comprimento do polígono em verde ao redor da componente e. Na figura (c) o fecho convexo está representado pelo polígono em vermelho. Os pontos em vermelho indicam os pontos de profundidade dos defeitos ascendente e descendente. No caso foram identificados dois defeitos ascendentes e um descendente.*

Etapa 1: Seleção por defeitos do fecho convexo É selecionado o subconjunto $CC_x \subset CC$ das componentes conexas que tenham no máximo um defeito ascendente e um descendente. Este critério se baseia na hipótese de que uma componente conexa que possui somente um caractere não deve ter muitos defeitos de grande tamanho.

Etapa 2: Seleção por estatísticas multivariadas O primeiro critério pode resultar em componentes conexas que possuem caracteres maiúsculos ou diminutos. Eles podem criar clusters e são dificilmente identificáveis a partir da sua altura ou largura.

Vamos utilizar as características de cada componente conexa de CC_x para criar um conjunto de observações multivariadas X. Cada componente conexa $cc_i \in CC_x$ é representada pela observação multivariada $\mathbf{x}_i$ que possui as seguintes características: altura, largura, comprimento do perímetro e o número de pixels de foreground e background da componente conexa cc_i (deixamos de fora o número de defeitos).

A partir do conjunto de observações multivariadas X, computamos o estimador robusto Determinante Mínimo de Covariância (DMC) para separar o subconjunto de componentes conexas que possuem um caractere minúsculo do resto. Este estimador encontra o subconjunto de observações que não são outliers. No nosso caso adaptamos esse critério da seguinte forma: o subconjunto de observações que não são outliers são aquelas que representam caracteres minúsculos.

Logo, o DMC é executado no conjunto X com parâmetro $h = 0.5$, isto é, esperamos que o conjunto de caracteres minúsculos seja pelo menos metade do conjunto X. A saída da execução do DMC é o subconjunto $J \in X$ junto com a sua estimativa robusta de localização $\hat{\mu}(J)$ e de dispersão $\hat{\Sigma}(J)$ [4].

Essas estimativas, $\hat{\mu}(J)$ e $\hat{\Sigma}(J)$, são utilizadas para calcular a distância de Mahalanobis ajustada $\alpha \cdot DM(\mathbf{x}, \hat{\mu}(J), \hat{\Sigma}(J))$ para cada vetor $\mathbf{x} \in X$ onde $\alpha = \frac{c-(m-p+1)}{mp}$ é o fator

[4] Para rodar o DMC foi utilizada a implementação do pacote "DetMCD" do R

de ajuste.

O fator de ajuste depende de três parâmetros p, c, m: O valor de p é a dimensionalidade das observações, logo $p = 5$. Os valores de c e m dependem do tamanho da amostra que esta sendo processada, assim, para poder comportar páginas com um número variado de componentes conexas, esses valores foram pré-computados para vários tamanhos, de 25 até 4000 caracteres por página [5].

Finalmente, no conjunto $CC_t \in CC_x$, são coletadas as componentes conexas de CC_x cujas observações $\mathbf{x} \in X$ possuírem distância de Mahalanobis ajustada menor ao quantil 97.5 da distribuição $F_{(p,m-p+1)}$.

Assim, nesta etapa, o conjunto CC_t possui os caracteres minúsculos da página. Esse conjunto será utilizado para classificar o resto das componentes conexas nos outros três tipos: maiúsculos, diminutos e ligações.

Etapa 3: Seleção por estatísticas uni-variadas Nesta etapa são calculadas as estatísticas robustas MADN para as alturas e larguras das componentes do conjunto CC_t obtido na etapa anterior. Essas duas estatísticas servem para classificar todas as componentes conexas de CC nos quatro tipos definidos anteriormente.

Sejam **h** e **w** os conjunto de valores que correspondem com as alturas e larguras das componentes de CC_t respectivamente. Para cada componente conexa $cc_i \in CC_x$ é calculada a medida robusta da altura

$$h'_i = (h_i - Med(\mathbf{h}))/MADN(\mathbf{h}).$$

De forma análoga é obtida a medida robusta da largura w'_i utilizando o conjunto de dados **w**.

Com essas duas medidas, cada componente conexa cc_i é classificada em um dos quatro tipos definidos anteriormente. Por exemplo, o critério para decidir se uma componente conexa possui caracteres ligados é $w'_i \geq 2.1$ e $|h'_i| \leq 2.1$, ou seja a componente conexa se encontra a 2.1 desvios padrão com respeito à estatística MADN da largura e altura do conjunto CC_t (o número de desvios padrão foram obtido mediante experimentação).

Os critérios para cada tipo são mostrados na figura 3.25.

Esta etapa finaliza com a obtenção dos conjuntos para cada tipo de caractere, no entanto, o conjunto que possui caracteres ligados é processado ainda mais para tentar obter caracteres minúsculos. Sejam CC_y e CC_z os conjuntos das componentes conexas que possuem caracteres minúsculos e caracteres ligados respectivamente. As componentes de CC_z são segmentadas utilizando as medidas de localização $\hat{\mu}(Y)$ e dispersão $\hat{\Sigma}(Y)$ do conjunto de dados multivariado Y que é obtido para o conjunto CC_y.

O algoritmo que segmenta as componentes conexas *cc* com caracteres ligados recebe como entrada os caminhos de segmentação da componente conexa *cc*. Na continuação vamos mostrar como obter esses caminhos de segmentação para depois mostrar o algoritmo de segmentação.

[5] O programa que computa os valores se baseia na técnica definida em [Har00] e está baseado no código fonte disponível em http://pages.pomona.edu/~jsh04747/Research/cm.r.

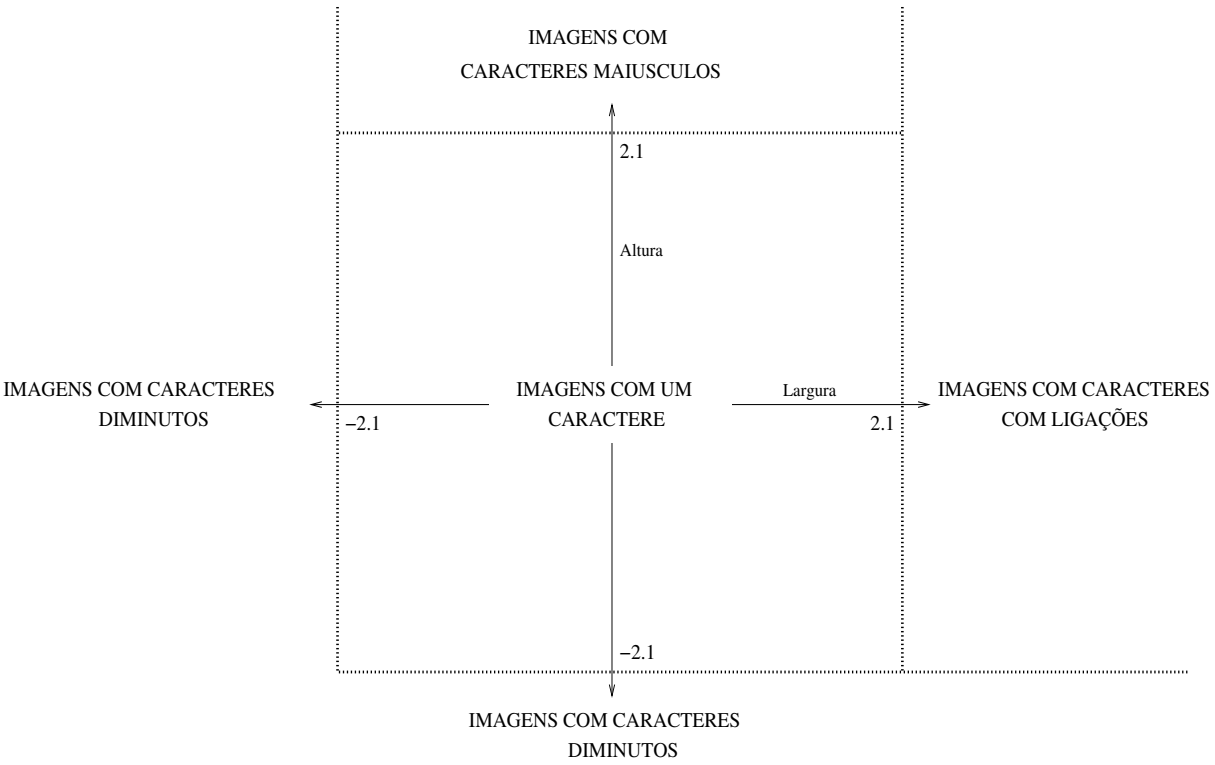


Figura 3.25: *Classificação das componentes conexas de uma página. O critério de classificação utiliza a estatística MAD normalizada as alturas e larguras. Os número na figura indicam a quantidade de unidades MADN que devem ser ultrapassadas para classificar uma componente conexa em uma classe específica.*

3.7.2 Geração de Caminhos de Segmentação

Depois de obtido o conjunto de componentes conexas CC_z com caracteres ligados, essas componentes são processadas para obter mais caracteres minúsculos. Para tal, para todo elemento de CC_z é gerado uma sequência de caminhos de segmentação.

Um caminho de segmentação é uma sequência de retas que une segmentos do esqueleto de background com o esqueleto de foreground. Os caminhos de segmentação fornecem hipóteses que serão utilizadas posteriormente para encontrar a segmentação ótima da componente conexa com caracteres ligados.

Especificamente, com respeito à técnica de obtenção do esqueleto, foi implementado o algoritmo de Hilditch [Hil69] por uma questão de simplicidade[6].

Os passos do procedimento são os seguintes:

1. Dada a componente conexa calculamos o esqueleto do foreground e background como mostram as figuras 3.26c e 3.26d

2. Em seguida, a partir de ambos esqueletos, obtemos três tipos de segmentos:

   Segmento superior É o segmento gerado a partir da parte superior do esqueleto de background. Para obter este segmento são primeiro localizados dois pontos, P_E e P_D, no esqueleto de background. Os pontos P_E e P_D correspondem às regiões superior esquerda e superior direita respectivamente.

[6]Na proposta original de [NMG+10] foi utilizado o algoritmo proposto em [LC92] que é em si uma otimização de performance sobre o algoritmo de Hilditch.

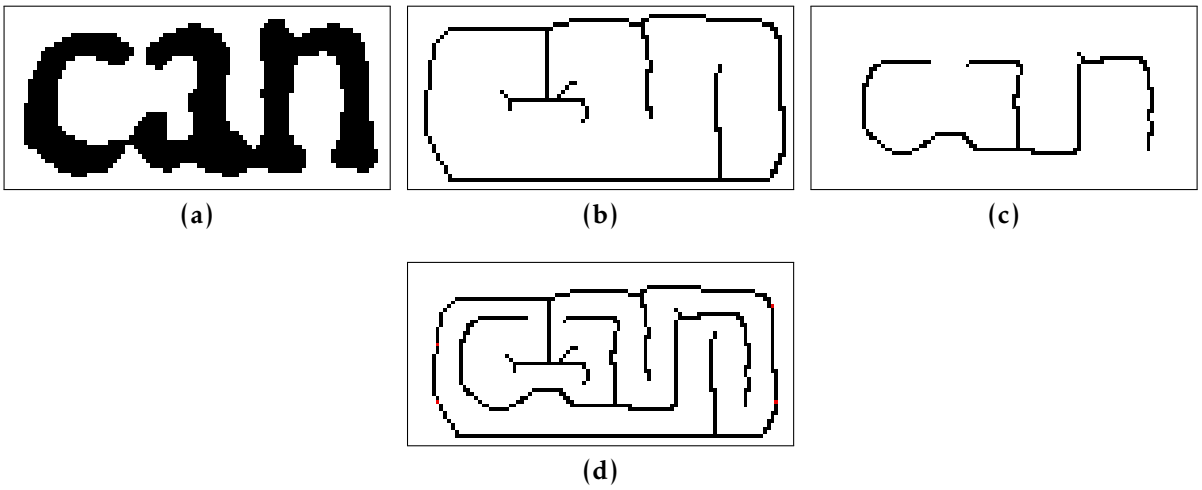


Figura 3.26: *Linha com caracteres grudados que serão segmentados. A figura (a) mostra uma componente conexa com três caracteres ligados. As figuras (b) e (c) mostram os esqueletos de background e foreground desta componente conexa. A figura (d) mostra a imagem de ambos os esqueletos combinados que serve de base para encontrar os caminhos de segmentação.*

Para encontrar o ponto P_E fazemos uma varredura vertical da imagem utilizando uma janela de tamanho 5×10 começando da esquina superior esquerda. A varredura é suspensa se a janela atual contém algum pixel de foreground, logo é selecionado o pixel com a maior ordenada e menor abscissa dentro da janela (o sistema de coordenadas tem origem na esquina superior esquerda). Caso a varredura não encontre nenhuma janela com pelo menos um pixel, então a varredura começa de novo a partir do topo da imagem mas deslocando a janela em 5 pixels para a direita.

De forma análoga, é feita a busca do ponto P_D a partir da esquina superior direita como mostra a figura 3.27.

Uma vez encontrados ambos pontos é feita uma busca em profundidade partindo de P_E até encontrar o ponto P_D. Para identificar somente pixels da parte superior são descartados todos os pixels com ordenada menor que P_E (P_D) dentro da sua vizinhança 3×3.

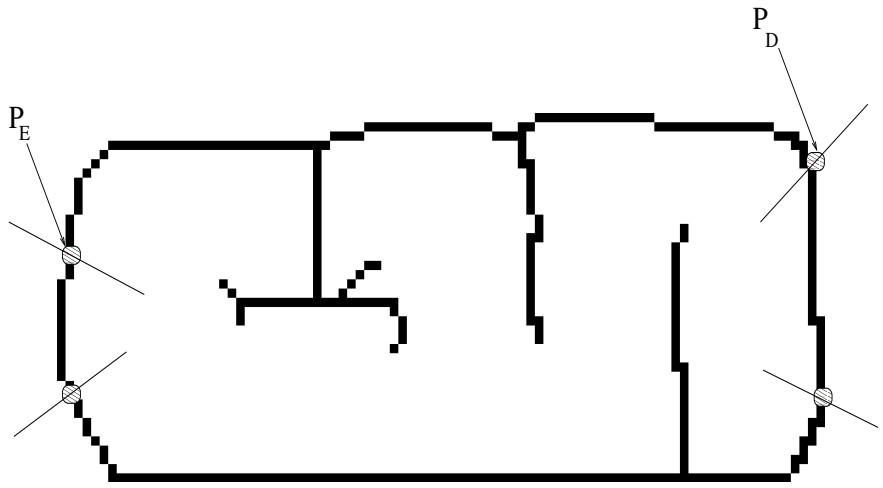


Figura 3.27: *Segmentos superior e inferior a partir do esqueleto de background. O segmento superior são todos os pixels que se encontram entre P_E e P_D. O segmento inferior é análogo ao superior.*

Segmento inferior É o segmento gerado a partir da parte inferior do esqueleto do background. O método para encontrar este segmento é análogo ao segmento superior, basta girar a imagem 180 graus e repetir o mesmo processo realizado no segmento superior.

Segmento interior É o segmento formado pelos pixels do esqueleto de foreground.

A figura 3.28a mostra esses segmentos a partir dos esqueletos anteriormente apresentados.

3. Logo é realizada uma varredura nos pontos de todos os segmentos para identificar os *pontos característicos*. Esses pontos podem ser de três tipos:

   Ponto de bifurcação É o ponto que conecta mais de dois segmentos do esqueleto;

   Ponto final O ponto de segmento que tem somente um pixel na sua vizinhança 3×3;

   Ponto de esquina O ponto de segmento onde a curvatura do segmento muda drasticamente

   Especificamente, para encontrar os pontos de esquina, usamos o método proposto em [CW00], onde eles são definidos como aqueles onde ocorre uma mudança de pelo menos 20 graus na curvatura do segmento. Para calcular o ângulo da curvatura é levada em conta a vizinhança de tamanho 9×9 ao redor do ponto característico. A figura 3.28b mostra os pontos etiquetados em cada um dos segmentos.

4. Os pontos característicos de cada segmento são utilizados para construir os caminhos possíveis de segmentação.

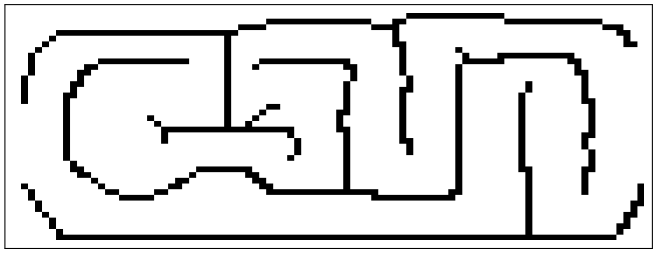
   Um caminho de segmentação está conformado por um ponto característico do segmento superior, um ou dois pontos no segmento interior e um ponto característico do segmento inferior. A distância (euclidiana) entre quaisquer par de pontos deve ser menor que 80% da mediana das alturas.

   Caso não exista um ponto característico disponível então é traçada uma linha vertical até o segmento inferior. Este processo se repete de forma análoga a partir do segmento inferior com direção ao segmento superior. Assim obtemos a coleção de caminhos de segmentação.

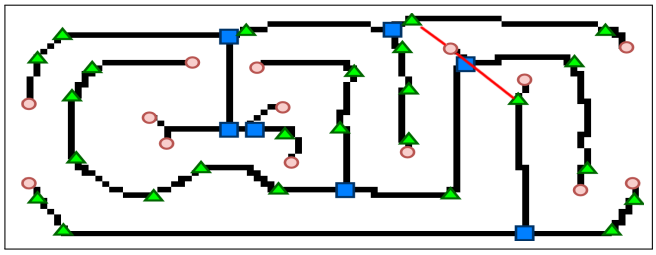
   Cada caminho possui um *comprimento* definido como a distância euclidiana entre os pontos no segmento superior e inferior. Podemos falar também em *largura* do caminho que é definida como a diferença entre as abscissas dos pontos mais à esquerda e mais à direita no caminho de segmentação.

   Os caminhos da coleção são filtrados de acordo com os seguintes critérios:

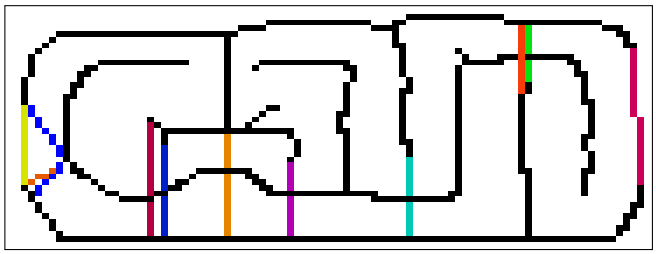
   - A altura do caminho deve ser menor que a mediana das alturas das componentes conexas de CC_y
   - A largura deve ser menor que 30% da mediana das larguras das componentes conexas de CC_y
   - A proporção de pixels de foreground e background que o caminho de segmentação contém deve ser menor que $pathPxRatio$.



(a) *Segmentos superior, interior e inferior obtidos a partir dos esqueletos*



(b) *Pontos característicos. A linha vermelha é um caminho de segmentação inválido pois o segmento interior é cortado entre o segundo e terceiro pontos.*



(c) *Caminhos de segmentação válidos obtidos a partir dos pontos marcados anteriormente*

Figura 3.28: *A sequência de figuras mostra a segmentação dos caracteres conexos. Na figura (b) os pontos marcados com círculo são pontos finais. Já os pontos marcados com triângulos e retângulos correspondem aos pontos de esquina e de bifurcação respectivamente*

- Entre dois pontos de um caminho de segmentação não deve existir algum ponto do segmento interior, superior ou inferior. Um exemplo da aplicação desta regra é o caminho de segmentação representado pelas linhas vermelhas na figura 3.28b. Esse caminho de segmentação é inválido porque entre o segmento interior é cortado entre segundo e terceiro ponto desse caminho de segmentação.
- Se for um caminho vertical, ele deve cortar o segmento interior em exatamente um ponto.

Neste ponto o trabalho original de Nikolau [NMG+10] se tornou irreprodutível pois o método descrito utiliza uma busca multi-criterial que não ficou muito clara [7].

Por outro lado, em [PPLD12], também foi proposto um algoritmo de programação dinâmica que utiliza informação de caminhos de segmentação para quebrar imagens com caracteres ligados. Infelizmente, a modelagem apresentada não demonstra possuir a propriedade de estrutura ótima.

[7] Tentamos inclusive entrar em contato com o autor, diversas vezes, de forma direta ou através de colaboradores, mas sem sucesso.

Assim, decidimos propor uma solução ao problema usando ainda os caminhos de segmentação. A nossa solução utiliza os caminhos de segmentação ordenados pela abscissa do ponto mais à direita. A sequência de caminhos ordenada é logo processada para encontrar a segmentação ótima. Essa segmentação, apresentada na continuação, utiliza um algoritmo de programação dinâmica que combina os caminhos para encontrar essa segmentação.

3.7.3 Segmentação ótima baseada em sequência de caminhos

Baseados na sequência de caminhos obtidos na seção anterior podemos obter regiões associadas a cada par de caminhos. Suponha uma componente conexa a ser segmentada e uma sequência de caminhos de quebra $(p_0,...,p_n)$, com $n \geq 1$. Cada par de caminhos consecutivos p_{i-1} e p_i define a região S_i. Contudo, também definimos uma região entre caminhos não consecutivos. Sejam inteiros i, j e k tais que $1 \leq i \leq k \leq j \leq n$. Definimos a região $S_{i,j} = S_i S_{i+1} \cdots S_j$ como sendo a região formada pelos pixels entre os dois caminhos de segmentação p_{i-1} e p_j, de forma que $S_{i,i} = S_i$ e $S_{i,j} = S_i S_{i+1} \cdots S_j$. Em particular, $S_{1,n} =$

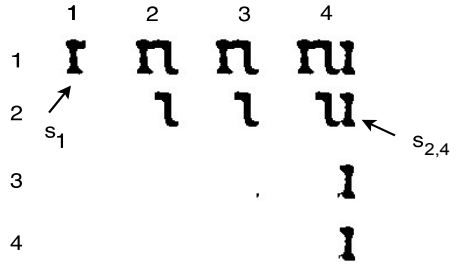


Figura 3.29: *Regiões $S_{i,j}$ definidas por caminhos de segmentação p_{i-1} e p_j. A componente conexa é $S_{0,4}$ (posição 0,4 da matriz), definida por p_0 e p_4. As regiões S_1, S_2, S_3 e S_4 estão na diagonal principal.*

$S_1 S_2 \cdots S_n$, a concatenação das regiões $S_1, S_2, \ldots, S_n$, é a próxima componente conexa a ser segmentada. Ademais, o caminho de quebra p_k divide a região $S_{i,j}$ nas regiões $S_{i,k}$ e $S_{k+1,j}$. Por exemplo, a sequência de caminhos $(p_0, p_1, p_2, p_3, p_4)$ define as regiões S_1, S_2, S_3 e S_4, e o caminho p_1 quebra a região $S_{1,4} = S_1 S_2 S_3 S_4$ nas regiões S_1 e $S_{2,4} = S_2 S_3 S_4$. Na figura 3.29 mostramos uma matriz com as regiões definidas por 5 caminhos de segmentação (p_0 até p_4). A imagem da componente conexa está representada pela posição $1,4$ da matriz.

Uma segmentação que utilize esses caminhos deverá selecionar um subconjunto dos $n-1$ caminhos internos $\{p_1,...,p_{n-1}\}$ de forma que a componente conexa deve ser quebrada nestes caminhos, produzindo regiões que supostamente deverão conter caracteres minúsculos. Surge naturalmente a pergunta: de quantas formas diferentes podemos segmentar essa sequência? Dado que o número de maneiras de se escolher um subconjunto de caminhos internos é 2^{n-1}, um algoritmo de busca exaustiva da segmentação mais apropriada é necessariamente exponencial no número de caminhos de segmentação.

Visando a obtenção de um algoritmo polinomial para resolver o problema, realizamos nesta seção uma modelagem realista e que possibilita a obtenção de um algoritmo de programação dinâmica que encontra uma segmentação de uma componente conexa que seja ótima segundo algum critério razoável.

Primeiramente, precisamos avaliar quão próxima uma região se encontra de um caractere minúsculo. Aqui podemos lançar mão do modelo estatístico mostrado nas etapas anteriores para dar uma pontuação a cada região S_{ij}. Isto nos obriga a extrair de cada região as mesmas características definidas na seção 3.7.1. Essas características são a altura, largura, comprimento do perímetro e número de pixels de foreground e background. Se

a altura (ou largura) de uma região S_{ij} for menor que a mediana da altura (ou largura) do conjunto de componentes conexas com caracteres minúsculos (conjunto CC_y da Etapa 3, seção 3.7.1), então a altura (largura) da região é corrigida com a mediana correspondente (altura ou largura) desse conjunto. Dadas essas características, representamos por $\mathbf{s}_{ij}$ o vetor que corresponde à região $S_{i,j}$. Definimos então a seguinte função de pontuação da região $S_{i,j}$:

$$r_{i,j} = \frac{1}{1 + DM(\mathbf{s}_{i,j}, \hat{\mu}(Y), \hat{\Sigma}(Y))},$$

onde $\hat{\mu}(Y)$ e $\hat{\Sigma}(Y))$ são as medidas de localidade e dispersão robustas do conjunto CC_y, respectivamente, e DM é a distância de Mahalanobis. Observe que a pontuação $r_{i,j}$ constitui um critério de proximidade da região $S_{i,j}$ a um caractere minúsculo pois diminui e tende a zero quando a distância de Mahalanobis cresce e tende a infinito, ao passo que atinge seu valor máximo, 1, quando a distância atinge o valor mínimo, 0. Ela toma como termo de referência a dispersão robusta. Assim, a função $r_{i,j}$ é um *critério de similaridade* entre uma região $S_{i,j}$ e os caracteres minúsculos da imagem. Voltando ao nosso exemplo, na tabela 3.1 mostramos as pontuações individuais $r_{i,j}$ de cada região $S_{i,j}$ presente na figura 3.29.

$r_{i,j}$	1	2	3	4
1	0.073	0.097	0.087	0.024
2		0.231	0.144	0.093
3			0.004	0.052
4				0.035

Tabela 3.1: *Pontuações $r_{i,j}$ das regiões S_{ij}*

Além da pontuação de uma região $S_{i,j}$ que não sofre segmentação mas é avaliada segundo a pontuação $r_{i,j}$ de similaridade a um caractere minúsculo definida acima, faz-se necessário pontuar a eventual segmentação associada a um caminho de quebra p_k que segmenta a região $S_{i,j}$ nas regiões $S_{i,k}$ e $S_{k+1,j}$. Naturalmente, o processo pode ser iterado até $n-1$ vezes. Usando os parênteses como indicação de onde quebrar uma região $S_{i,j}$, listamos a seguir algumas possíveis segmentações da componente conexa $S_{1,4}$ do exemplo, assim como a sequência dos caminhos de quebra que foram ordenadamente utilizados:

$$\begin{gathered}
S_1\ (\ S_2\ (\ S_3\ S_4\ )\ ) \rightarrow (p_1, p_2) \\
S_1\ (\ (\ S_2\ S_3\ )\ S_4\ ) \rightarrow (p_1, p_3) \\
(\ (\ S_1\ S_2\ )\ S_3\ )\ S_4 \rightarrow (p_3, p_2) \\
S_1\ (\ S_2\ S_3\ S_4\ ) \rightarrow (p_1) \\
(\ S_1\ S_2\ )\ (\ S_3\ S_4\ ) \rightarrow (p_2) \\
(\ S_1\ S_2\ S_3\ )\ S_4 \rightarrow (p_3) \\
S_1\ S_2\ S_3\ S_4 \rightarrow ()
\end{gathered}$$

Observe que no último caso temos uma segmentação vazia, que mantém intacta a componente conexa $S_{1,4} = S_1S_2S_3S_4$. Se as parentizações feitas lembram o problema de parentização da multiplicação associativa de matrizes onde se busca minimizar o número de multiplicações escalares, faz-se necessário desenvolver um critério de otimização adequado para o problema da segmentação de imagens.

A adoção de um tal critério deve permitir que se combine as pontuações de duas regiões consecutivas $S_{i,k}$ e $S_{k+1,j}$, delimitadas respectivamente pelos pares de caminhos (p_{i-1}, p_k) e (p_k, p_j), de forma a pontuar a própria segmentação da região original $S_{i,j}$. Tomar o mínimo das pontuações? O máximo? Qualquer um destes critérios leva à perda de informação a respeito da qualidade da pontuação da outra imagem. Algum tipo de média? Que tipo de média? Como fazê-lo se estamos lidando com imagens? Problemas de natureza semelhante aparecem em Recuperação de Informação, onde a avaliação da relevância de um documento à necessidade de um usuário que realiza uma consulta tem uma componente fortemente subjetiva, mas deve ser pontuada numericamente para que se possa devolver uma lista de documentos ordenada por relevância ao usuário.

Para pontuar a segmentação de $S_{i,j}$ pela sequência de caminhos de quebra (p_k), deve-se fazê-lo em função das pontuações $r_{i,k}$ e $r_{k+1,j}$ associadas às regiões $S_{i,k}$ e $S_{k+1,j}$, respectivamente. Num caso mais geral, em que a sequência de caminhos de quebra P usada na segmentação inicia-se com p_k mas pode ter mais que um caminho de quebra, após se fazer a segmentação por p_k, outras segmentações segundo os caminhos de quebra restantes devem ser aplicadas às regiões $S_{i,k}$ e $S_{k+1,j}$. Sejam pois x e y as pontuações associadas às segmentações induzidas por P nas regiões $S_{i,k}$ e $S_{k+1,j}$, respectivamente. Em particular, temos que $x = r_{i,k}$ e que $j = r_{k+1,j}$ para $P = (p_k)$. Avaliamos a segmentação de $S_{i,j}$ por P através da seguinte *pontuação de combinação* φ, proposta neste trabalho, e definida por:

$$\varphi(x, y) = \beta \frac{2xy}{x + y}.$$

Observe que $\varphi(x, y)$ corresponde à média harmônica entre x e y penalizada por um fator multiplicativo β, de forma que chamamos esta pontuação de combinação φ de *média harmônica penalizada*. O parâmetro $\beta < 1$ (ou $\beta > 1$) funciona como uma penalização (ou incentivo) associada(o) a uma segmentação pois a média harmônica das pontuações x e y deve superar $r_{i,j}/\beta$ para que compense segmentar $S_{i,j}$ pela sequência de caminhos de quebra P. Quanto menor for β, mais penalizada é uma eventual segmentação. Em particular, para $k = i + 1, \ldots, j$ e $P = (p_k)$, não vale a pena segmentar a região $S_{i,j}$ por P se

$$\beta \leq r_{i,j} \cdot \frac{r_{i,k} + r_{k+1,j}}{2 \cdot r_{i,k} \cdot r_{k+1,j}}.$$

Caso exista um k para o qual a desigualdade não valha, $\varphi(r_{i,k}, r_{k+1,j}) > r_{i,j}$ e vale a pena segmentar $S_{i,j}$ segundo a sequência de caminhos de quebra P.

Em nossos experimentos, a média harmônica penalizada φ mostrou-se superior ao ser comparada a outras funções de pontuação de combinação alternativas (foram testadas também a média geométrica penalizada, média aritmética, mínimo e multiplicação simples das pontuações individuais). Nossos experimentos mostraram maior assertividade na segmentação e maior robustez para $\beta = 0.5$.

Fazendo uso das pontuações mostradas acima, podemos computar a pontuação de

cada uma e das segmentações já consideradas em nosso exemplo:

$$
\begin{aligned}
S_1\ (\ S_2\ (\ S_3\ S_4\ )\ ) &\rightarrow 0,027\\
S_1\ (\ (\ S_2\ S_3\ )\ S_4\ ) &\rightarrow 0,020\\
(\ (\ S_1\ S_2\ )\ S_3\ )\ S_4 &\rightarrow 0,006\\
S_1\ (\ S_2\ S_3\ S_4\ ) &\rightarrow \mathbf{0,041}\\
(\ S_1\ S_2\ )\ (\ S_3\ S_4\ ) &\rightarrow 0,034\\
(\ S_1\ S_2\ S_3\ )\ S_4 &\rightarrow 0,025\\
S_1\ S_2\ S_3\ S_4 &\rightarrow 0,024
\end{aligned}
$$

A segmentação ótima está destacada em negrito e corresponde à segmentação da imagem $S_{1,4}$ pelo caminho p_1 nas regiões S_1 e $S_{2,4}$, que também estão destacadas na figura 3.29.

Seja pois $s_{i,j}$, *a máxima pontuação obtida por uma segmentação, possivelmente vazia, da região* $S_{i,j}$.

Suponha que a segmentação ótima seja a vazia, de pontuação $r_{i,j}$, portanto. Qualquer outra segmentação quebra a componente conexa segundo algum caminho p_k dividindo-a em regiões $S_{i,k}$ e $S_{k+1,j}$, com pontuações para as segmentações induzidas respectivamente $s'_{i,k} \leq s_{i,k}$ e $s'_{k+1,j} \leq s_{k+1,j}$. Note que $\varphi(x,y) \geq \varphi(x',y')$ sempre que $x \geq x' > 0$ e $y \geq y' > 0$, pois a média harmônica é estritamente crescente em cada componente. Assim, a pontuação da segmentação alternativa é $\varphi(s'_{i,k}, s'_{k+1,j}) \leq \varphi(s_{i,k}, s_{k+1,j}) \leq r_{i,j}$.

Suponha uma segmentação não vazia de $S_{i,j}$ que seja ótima, com pontuação $s_{i,j}$. Seja p_k o primeiro caminho de quebra aplicado a esta segmentação. Temos então que as segmentações induzidas nas regiões $S_{i,k}$ e $S_{k+1,j}$ também devem ser ótimas. De fato, se a segmentação induzida em $S_{i,k}$ fosse subótima e pontuasse $s'_{i,k} < s_{i,k}$, poderíamos trocar a segmentação de $S_{i,k}$ por aquela de pontuação $s_{i,k}$ e sua combinação com a segmentação induzida sobre $S_{k+1,j}$, de pontuação z, teria pontuação $\varphi(s_{i,k}, z) > \varphi(s'_{i,k}, z) = s_{i,j}$, uma contradição. De forma análoga, a pontuação da segmentação induzida sobre $S_{k+1,j}$ é ótima.

Como o cálculo desta função recorrente $s_{i,j}$ tem esta propriedade de admitir uma solução composta de subproblemas cujas soluções devem ser também ótimas, é possível calcular $s_{i,j}$ através de uma recorrência que admite uma solução eficiente através de programação dinâmica:

$$
s_{i,j} = \max \begin{cases} r_{i,j} \\ \varphi(s_{i,k}, s_{k+1,j}), \text{ para todo } k \text{ tal que } i \leq k < j. \end{cases}
$$

Ao calcular $s_{1,n}$ obtemos a pontuação da segmentação ótima da região $S_{1,n}$ e, implicitamente, a própria sequência de caminhos de quebra utilizada. Se $i = j$, temos que $s_{i,i} = r_{i,i}$. Se $i < j$, então consideramos as possíveis segmentações por p_k para calcular $s_{i,j}$ a partir da média harmônica penalizada entre as pontuações $s_{i,k}$ e $s_{k+1,j}$ das segmentações ótimas de $S_{i,k}$ e de $S_{k+1,j}$, respectivamente. A implementação da solução com matriz de programação dinâmica é bastante direta se computamos a matriz que guarda os valores de $s_{i,j}$ a partir de diagonais crescentes, como mostra o algoritmo 1. De fato, ao calcular o valor de $s_{i,j}$ para guardá-lo na posição i, j, na diagonal $j - i = l$ da matriz, os valores $s_{i,k}$ e $s_{k+1,j}$ já estarão armazenados nas diagonais $k - i < j - i$ e $j - k - 1 < j - i$, respectivamente.

A complexidade do algoritmo é $O(n^3)$ pois para cada um dos $\frac{n(n+1)}{2}$ elementos $s_{i,j}$ em que $j \geq i$ (laços das linhas 5 e 7) são avaliadas k posições onde $i \leq k < j$ (laço da linha 12). O algoritmo gasta espaço $O(n^2)$ pois utiliza as matrizes: s, para armazenar a pontuação da solução atual; e a matriz SEP, para armazenar o índice k onde a região $S_{i,j}$ vai ser

quebrada para produzir as regiões S_{ik} e $S_{k+1,j}$.

Finalmente, o algoritmo de construção de soluções 2, começa pela célula $SEP[1,n]$ e volta atrás construindo as regiões segmentadas. Caso o número de soluções finais seja menor ou igual a 1 então o algoritmo não encontrou nenhuma solução satisfatória e o procedimento devolve a imagem original.

Algoritmo 1 SEGMENTAÇÃO ÓTIMA

Entrada:
p, sequência de caminhos $(p_0, p_1, ..., p_n)$
r, pontuações $r_{i,j}$ das regiões $S_{i,j}$
Saída:
SEP matriz de tamanho $n \times n$ inicializada com zeros

```
1  Inicializa matriz s de tamanho n × n com zeros
2  para i ← 1 até n faça
3     s[i,i] ← r_{i,i}
4  fim
5  para l ← 1 até n − 1 faça
6     limit ← n − l
7     para p ← 1 até limit faça
8        i ← p
9        j ← p + l
10       maxScore ← r_{i,j}
11       maxSep ← −1
12       para k ← i até j − 1 faça
13          score ← β * 2 * s[i,k] * s[k+1,j] / (s[i,k] + s[k+1,j])
14          se score > maxScore então
15             maxScore ← score
16             maxSep ← k
17          fim
18       fim
19       s[i,j] ← maxScore
20       SEP[i,j] ← maxSep
21    fim
22 fim
```

A figura 3.30 mostra a execução do algoritmo 1 para a componente conexa da figura 3.26a. Os caminhos de segmentação foram computados anteriormente e, para essa imagem são mostrados na figura 3.28c. Os elementos da diagonal são as regiões definidas por esses caminhos de segmentação, enquanto que as células em vermelho são as soluções ótimas encontradas pelo algoritmo 2.

Finalmente, os caracteres segmentados são retornados e, se houver símbolos de pontuação, estes são combinados nas componentes conexas mais próximas. Utilizando a informação de linhas e palavras e os caracteres, é extraído texto fazendo chamadas a um motor de OCR. O texto de saída é armazenado para ser utilizado em tarefas de alinhamento ou para melhorar o treinamento de um motor de OCR.

Algoritmo 2 CONSTRUÇÃO DE SOLUÇÕES

Entrada:
SEP matriz de tamanho $n \times n$ inicializada com zeros
Saída:
$solutions$ regiões geradas a partir de SEP

```
1  stack é uma pilha vazia
2  Empilhe (1, n) em stack
3  enquanto stack não é vazia faça
4     Desempilhar (i, j) de stack
5     k ← SEP[i, j]
6     se k = −1 então
7        Adicione S_{i,j} em solutions
8     senão
9        Adicione (i, k) e (k + 1, j) em stack
10    fim
11 fim
12 se |solutions| > 1 então
13    devolva solutions
14 senão
15    devolva componente conexa original
16 fim
```

3.8 Resumo dos parâmetros

O nosso método de segmentação possui muitos parâmetros que são definidos pelo usuário. A seguir fazemos um breve resumo deles:

Parâmetro	Descrição
a	é utilizado no cálculo do limiar T_l que controla, no ARLSA, o comprimento das sequências de background que podem conectar duas componentes conexas.
c	também é usado no ARLSA e controla o limiar T_O que define a porcentagem de sobreposição entre duas componentes conexas.
$\mathbf{T_h}$	também é usado no ARLSA. Dada uma sequência de pixels de background entre duas componentes conexas e a proporção de alturas entre essas componentes, a sequência de background é convertida para uma de foreground caso a proporção da alturas seja menor que T_h
cf	fator que controla a altura máxima que uma componente conexa deve possuir para ser classificada como ruído.
$\mathbf{T_E}$	limiar que controla o elongamento $E(CC)$ de uma componente conexa CC. Se $E(CC) < T_E$ então a componente é classificada como ruído.
$\mathbf{T_D}$	limiar que controla a densidade $D(CC)$ da componente CC. Se $D(CC) < T_D$ então a componente é classificada como ruído.
$\mathbf{T_R}$	limiar que controla a proporção P_R que mede a variação de pixels de uma componente CC antes e depois do ARLSA.

	Se $P_R \leq T_R$ então a componente é classificada como símbolo de pontuação.
k	fator que, multiplicado pela altura H da imagem, controla a detecção de obstáculos de coluna. Isto é, se uma sequência vertical de background S tiver comprimento $L(S) \geq k \times H$ então é considerada um obstáculo de tipo coluna.
pathWf	utilizado na segmentação de caracteres. Controla a largura de um caminho de segmentação para ser considerado como um caminho possível.
pathPxRatio	utilizado na segmentação de caracteres. Controla a proporção entre o número pixels de foreground e background que compõem um caminho de segmentação

3.9 Conclusões

Foi apresentado aqui uma adaptação do método de segmentação para documentos históricos proposto em [NMG+10].

A principal contribuição do método é o algoritmo de programação dinâmica que segmenta imagens com caracteres ligados. A solução pode ser facilmente adaptada para outro tipo de imagens (por exemplo manuscritos) e inclusive pode-se incrementar a sua eficacia se usarmos informações adicionais, como por exemplo, a quantidade de pixels que serão removidos após a segmentação dos caracteres ligados.

Mostramos também a utilização de técnicas de estatística robusta para a obtenção de um modelo não-supervisionado que serve como guia para o algoritmo de segmentação de caracteres com ligações.

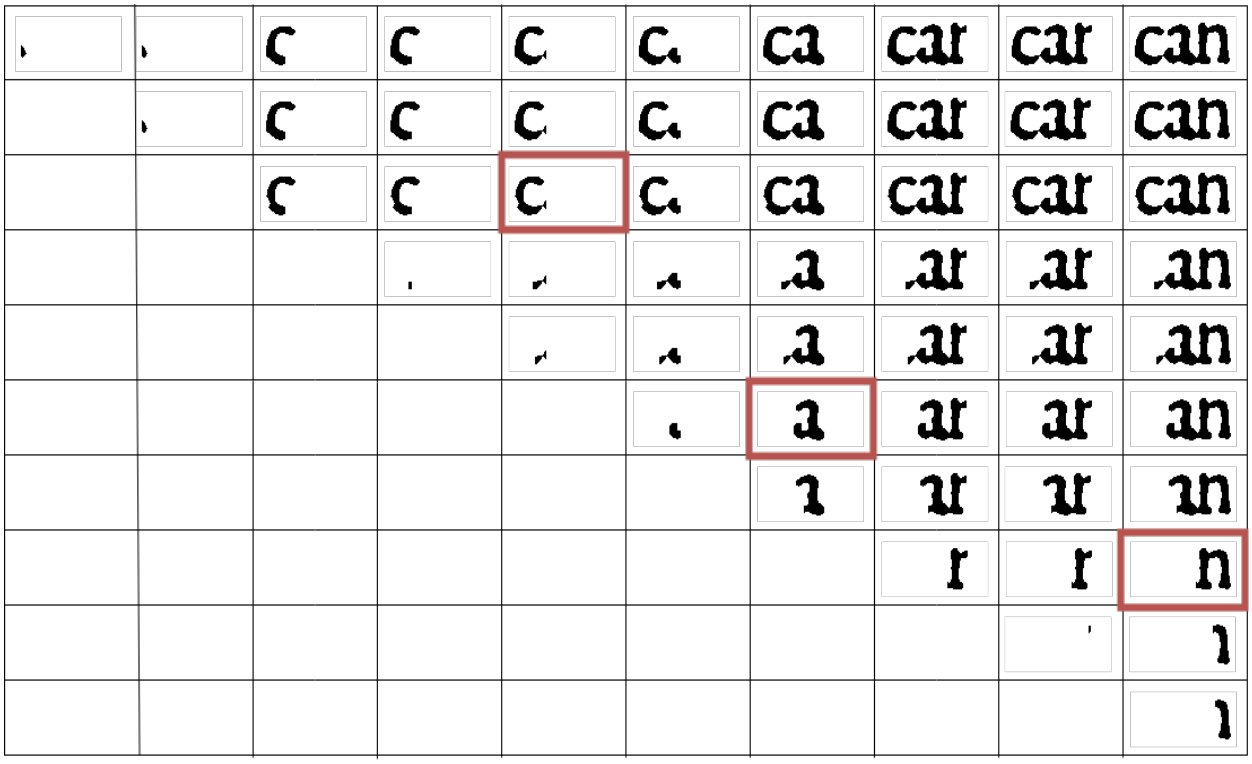


Figura 3.30: *Segmentação ótima de uma componente conexa utilizando programação dinâmica e estatísticas robustas. A imagem é mostrada na figura 3.26a e os seus caminhos de segmentação são mostrados na figura 3.28c. As células em vermelho são as soluções encontradas pelo algoritmo 2.*

Capítulo 4

Busca Aproximada e Recuperação de Informação

Realizar busca em documentos que possuem uma taxa de erro considerável como aqueles gerados por motores OCR é uma tarefa difícil. A dificuldade nasce do fato de que as palavras sofrem transformações que impedem de serem buscadas.

Assim torna-se importante resolver o problema da busca em sequências de símbolos que apresentam uma taxa de erro considerável. O problema da busca com erros, ou busca aproximada, também possui aplicações importantes em Bioinformática onde as sequências são sequenciadas com erros. Outra aplicação importante em Recuperação de Informação (RI) é correção ou sugestão de novos termos para uma consulta feita pelo usuário.

Neste capítulo vamos mostrar as técnicas utilizadas para resolver este problema no nosso sistema. Na seção 4.1 definimos conceitos básicos como sequência de símbolos e k-grama. Nessa mesma seção faremos uma breve revisão das principais técnicas de busca aproximada de símbolos tendo como referência o trabalho de Navarro [Nav01]. Na seção 4.2 mostramos alguns conceitos sobre Recuperação de Informação que são importantes para entender alguns dos algoritmos propostos neste capítulo assim como no capítulo 5. Em seguida, na seção 4.3, apresentamos o nosso algoritmo para realizar busca aproximada em uma coleção de documentos misturando técnicas de filtros e programação dinâmica.

4.1 Busca aproximada em sequências de símbolos

Um *sequência de símbolos* ou *palavra* é a concatenação de zero ou mais símbolos de um alfabeto Σ. Consideramos uma sequência s de tamanho n e adotamos o termo *subpalavra* para denotar um segmento contíguo de s. Também consideramos um inteiro $k \geq 2$ e um conjunto de sequências $s_1, s_2, ..., s_k$. A sequência s de tamanho l no alfabeto Σ está representada por $s[1], ..., s[l]$, onde $s[i] \in \Sigma$ para $1 \leq i \leq l$. Denotamos por $s[i, j]$ a subpalavra $s[i]s[i+1]...s[j]$ de s. Neste caso dizemos que a subpalavra $w = s[i, j]$ *ocorre na posição i em s ou que w começa na posição i em s*. Em particular, um *prefixo* de s é uma subpalavra $s[1, i]$ para algum i e um *sufixo* de s é uma subpalavra $s[j, n]$ para algum j. Um *k-grama de símbolos* de s é uma subpalavra de tamanho k.

Seja $P, T \in \Sigma^*$ duas palavras onde P é um *padrão* de entrada e T um *texto*. Seja também $\lambda \in \mathbb{Z}$ a *taxa de erro máxima permitida* e $d : \Sigma^* \times \Sigma^* \rightarrow \mathbb{Z}$ uma *função de distância*.

O problema da busca aproximada consiste em encontrar as posições j tal que existe um i de forma que $d(P, T_{i..j}) \leq \lambda$.

Considere, para o resto da apresentação, as sequências arbitrárias x, y, z e os símbolos $a, b, c, ...$ A distância $d(x, y)$ entre duas sequências x e y é o custo mínimo da sequência de operações que transformam x em y (e ∞ se essa sequência não existe). O custo da sequência de operações é a soma dos custos das operações individuais. As operações são um conjunto finito de regras da forma $\delta(z \rightarrow w) = t$, onde z e w são sequências diferentes e t é um número real não negativo.

Por outro lado, se para cada operação da forma $\delta(x, y)$ existe a operação respectiva $\delta(y, x)$ ao mesmo custo, então a distância é simétrica (i.e. $d(x, y) = d(y, x)$). Observe também que $d(x, y) \geq 0$ para todas as sequências x e y, que $d(x, x) = 0$, e que a desigualdade triangular $d(x, z) \leq d(x, y) + d(y, z)$ é sempre válida.

Na maioria das aplicações, o conjunto de operações admitidas está restrito a:

- *Inserção:* $\delta(\epsilon \rightarrow a)$, ou a inserção do símbolo a
- *Remoção:* $\delta(a \rightarrow \epsilon)$, ou a remoção do símbolo a
- *Substituição:* $\delta(a \rightarrow b)$ para $a \neq b$, ou a substituição do símbolo a por b

Com estes conceitos podemos definir a função de distância que é de interesse no nosso trabalho [1]. A distância de Levenhstein ou *distância de edição* [LI66] permite os três tipos de operações descritos anteriormente. Na versão simplificada o custo de todas as operações é 1. A distância é simétrica e cumpre a propriedade $0 \leq d(x, y) \leq max(|x|, |y|)$.

Várias técnicas tem sido propostas para o cálculo da distância de edição, e podemos agrupá-las elas em quatro áreas:

Programação Dinâmica É uma das áreas mais antigas. As técnicas propostas nesta área se baseiam em alguma forma de matriz de programação dinâmica.

Autômatos Também é uma área antiga. A técnica básica consiste em construir autômatos finitos não-determinísticos (AFN) para determinar as operações de edição através das transições do autômato.

Paralelismo de Bits Nesta área a técnica básica consiste em aproveitar a capacidade do computador para processar sequências de bits em paralelo. Este paralelismo é aplicado, por exemplo, nas técnicas de AFN.

Filtros Esta área é a mais recente e tem tido bastante desenvolvimento especialmente na área de Bioinformática. A técnica de filtragem consiste em destacar rapidamente subpalavras do texto T que não possuam muita similaridade com o padrão P. Depois dessa etapa é aplicado um algoritmo de programação dinâmica para verificar os resultados. Estes algoritmos são bastante rápidos na prática para taxas de erro baixas e lentos quando a taxa é alta.

O nosso foco neste trabalho é nas técnicas de Programação Dinâmica e Filtros. Com respeito à primeira técnica vamos apresentar brevemente o algoritmo original e em seguida uma otimização proposta por Ukkonen. Sobre as técnicas de filtros vamos propor uma técnica padrão utilizada em Recuperação de Informação e deixarmos a sua definição para a seção 4.2.

[1] O leitor pode encontrar descrições de outras funções de distância em [Nav01]

4.1.1 Técnicas de Programação Dinâmica

A primeira técnica que apareceu em Busca Aproximada foi a de Programação Dinâmica. O algoritmo clássico de programação dinâmica foi descoberto várias vezes em diferentes áreas ([Vin68, NW70, San72]).

Algoritmo clássico

Sejam duas palavras x e y duas palavras quaisquer de comprimentos m e n respectivamente. Suponha que precisamos computar $d(x,y)$. Podemos utilizar uma matriz C de tamanho $(m+1)\times(n+1)$ de forma que C_{ij} armazene o número mínimo de operações necessárias para converter $x[1,i]$ em $y[1,j]$. Essa computação é definida pela seguinte recorrência:

$$C_{i0} = i \tag{4.1}$$

$$C_{0j} = j \tag{4.2}$$

$$C_{ij} = min\begin{cases} C_{i,j-1} + \delta(\epsilon \rightarrow y[j]) \\ C_{i-1,j} + \delta(x[i] \rightarrow \epsilon) \\ C_{i-1,j-1} + \delta(x[i] \rightarrow y[j]) \end{cases} \tag{4.3}$$

Onde

$$\delta(x[i] \rightarrow y[j]) = \begin{cases} 0, & \text{se} \quad x[i] = y[j] \\ 1, & \text{caso contrário} \end{cases}$$

e

$$\delta(\epsilon \rightarrow y[j]) = \delta(x[i] \rightarrow \epsilon) = 1$$

A computação da matriz termina ao calcular $C_{mn} = d(x,y)$. A recorrência define a forma em que a matriz é preenchida: primeiro, C_{i0} e C_{0j} representam a distância de edição entre o prefixo de comprimento i ou j e a sequência vazia. Claramente são necessárias i (e j respectivamente) remoções no prefixo $x[i]$ (e $y[j]$ respectivamente). Para duas subpalavras não vazias de comprimento i e j, supomos por indução que todas as distâncias de edição entre subpalavras mais curtas já foram computadas, e tentamos converter $x[1,i]$ em $y[1,j]$.

Considere os últimos caracteres $x[i]$ e $y[j]$. Se eles são iguais, então não precisamos considerá-los e continuamos com a forma ótima para converter $x[1,i-1]$ em $y[1,j-1]$. Por outro lado, se eles não são iguais devemos proceder em três maneiras diferentes: apagar $x[i]$ e converter de forma ótima $x[1,i-1]$ em $y[1,j]$, apagar $y[j]$ e converter de forma ótima $x[1,i]$ em $y[1,j-1]$ ou substituir $x[i]$ por $y[j]$ e converter de forma ótima $x[1,i-1]$ em $y[1,j-1]$.

Em todos estes casos o custo está definido pela função $\delta()$ mais o custo do resto da computação que já está disponível.

O algoritmo de programação dinâmica preenche a matriz de forma que, para cada célula, os vizinhos da esquerda, acima e superior-esquerda já estejam computados. Isto é feito iterando nas linhas da matriz da esquerda para a direita ou nas colunas de cima para baixo.

Uma vez computada a matriz C, a solução pode ser obtida indo no caminho inverso, a partir de C_{mn} até C_{00} e guardando em cada etapa, qual é a alternativa mínima. Por exemplo, se foi encontrado que C_{ij} faz parte da solução e que $C_{ij} = C_{i-1,j} + \delta(x[i] \rightarrow \epsilon)$, então $C_{i-1,j}$ é a próxima célula a ser considerada na solução e a operação de edição é

"apagar $x[i]$".

O algoritmo tem complexidade $O(mn)$ e consome espaço $O(min(m,n))$ se somente for mantida uma linha ou coluna na iteração da matriz.

A figura 4.1 mostra um exemplo do algoritmo de programação dinâmica. As arestas representam a dependência entre as células $C_{i'j'}$ e C_{ij}, isto é C_{ij} foi obtida a partir de $C_{i'j'}$ na etapa da minimização. As arestas indicam o custo da função δ que foi utilizada. No exemplo foram utilizados custos unitários. O grafo resultante é conhecido como *grafo de dependências*.

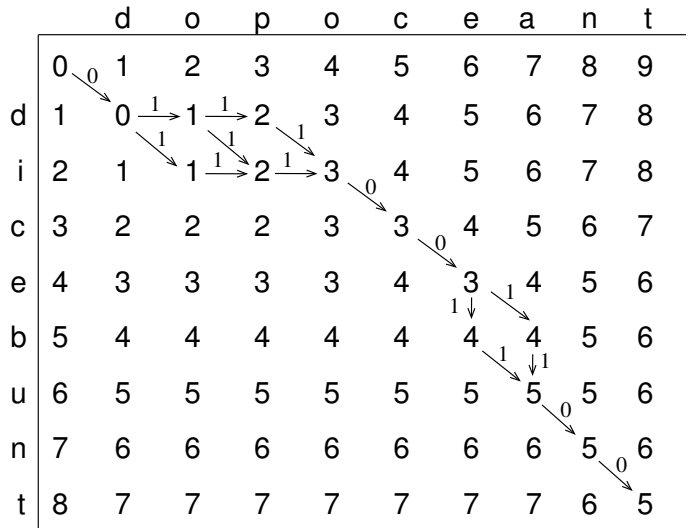


Figura 4.1: *Exemplo da matriz de programação dinâmica com custos unitários. A figura mostra também o grafo de dependências. As palavras usadas no exemplo são o latim para "diziam" (dicebant) e "ensinem" (doceant). Foram inseridos erros: foi trocado o "a" por "u" em dicebant e o caractere "o" foi substituído pela sequência "opo" em doceant)*

O algoritmo clássico não é tão eficiente porque com frequência avalia células C_{ij} de forma desnecessária. Na continuação vamos mostrar uma otimização baseada em observações sobre o padrão de preenchimento da matriz C.

Teste de Ukkonen

Ukkonen propôs um algoritmo que aproveita a estrutura do grafo de dependências ([Ukk85]) definido acima. Observe que no grafo as arestas horizontais correspondem com a operação de inserção $\delta(\epsilon \rightarrow a)$, as arestas verticais correspondem com operações de remoção $\delta(a \rightarrow \epsilon)$ e as arestas diagonais com a operação de substituição $\delta(a \rightarrow b)$.

Observe que o valor de C_{ij} é a soma dos custos nas arestas de qualquer caminho desde C_{00} até C_{ij}. A seguir o Lema 1 formaliza essa observação:

Lema 1 ([Ukk85]). *Se o grafo de dependências contém um caminho dirigido de C_{ij} para $C_{i'j'}$ então $C_{i'j'} = C_{ij} + d$, onde d denota a soma dos custos das arestas no caminho.*

Fica claro no grafo de dependências que somente as células C_{ij} que aparecem em algum caminho de C_{00} para C_{mn} são relevantes para o valor de C_{mn}. De fato, caso pudéssemos conhecer algum caminho a priori, poderíamos calcular C_{mn} computando as entradas no caminho começando de C_{00} e estabelecendo que todas as células que não estão no caminho tem valor ∞. Observe também que $C_{mn} = O(max(m,n))$ já que qualquer caminho de C_{00} até C_{mn} contém no máximo $m+n$ arestas.

Considere agora o problema de testar se a distância $d(x,y)$ é no máximo λ. Este problema pode ser resolvido avaliando (C_{ij}) com o algoritmo tradicional e depois verificar se $C_{mn} \leq \lambda$. Por outro lado, a partir do Lema 1 sabemos que os valores C_{ij} são monotonicamente crescentes ao longo de qualquer caminho no grafo de dependências. Logo, se C_{mn} for de fato $\leq \lambda$ e se algum C_{ij} tem um valor maior que λ, então C_{ij} não pode pertencer a qualquer caminho que leva até C_{mn}. Ainda mais, todas as células que não terão valor maior que λ, devem estar em uma banda diagonal de (C_{ij}) que é tanto mais estreita quanto λ for menor.

Para deixar mais claro esse conceito, vamos denotar Δ como o mínimo custo das operações de inserção e remoção:

$$\Delta = min\{\delta(a \rightarrow b) \mid a \neq b\}$$

Já que função δ é positiva, então $\Delta > 0$. Vamos enumerar as diagonais de (C_{ij}) com inteiros $-m, -m+1, ..., 0, 1, ..., n$ tal que a diagonal denotada por k consiste nos C_{ij} tais que $j - i = k$.

Lema 2 ([Ukk85]). *Se o grafo de dependências contém um caminho dirigido de C_{ij} para $C_{i'j'}$ então $C_{i'j'} \geq C_{ij} + |j' - i' - (j - i)|$.*

O Lema 2 implica $C_{ij} \geq |j - i|\Delta$ para cada C_{ij} em um caminho de C_{00} para C_{mn}, logo, pelo Lema 1, $|j - i| \leq C_{ij}/\Delta \leq C_{mn}/\Delta$. Portanto, para computar C_{mn} basta considerar os elementos C_{ij} na banda diagonal definida por $-C_{mn}/\Delta \leq j - i \leq C_{mn}/\Delta$. No entanto, uma banda diagonal ainda mais estreita pode ser considerada:

Corolário 1 ([Ukk85]). *Se C_{ij} está em algum caminho que leva de C_{00} para C_{mn} no grafo de dependências então $-p \leq j - i \leq n - m + p$ se $m \leq n$, e $n - m - p \leq j - i \leq p$ se $m > n$, onde $p = \lfloor \frac{1}{2}(C_{mn}/\Delta - |n - m|) \rfloor$*

A partir do Corolário 1 segue que para comprovar se $d(x,y) \leq \lambda$, a computação de (C_{ij}) pode estar limitada à banda diagonal que corresponde, caso p denote $\lfloor \frac{1}{2}(C_{mn}/\Delta - |n-m|) \rfloor$, com as diagonais $-p$ e $n - m + p$ quando $m \leq n$, e com as diagonais $n - m - p$ e p quando $m > n$. A figura 4.2 esclarece a enumeração das diagonais assim como mostra a banda diagonal para $m \leq n$.

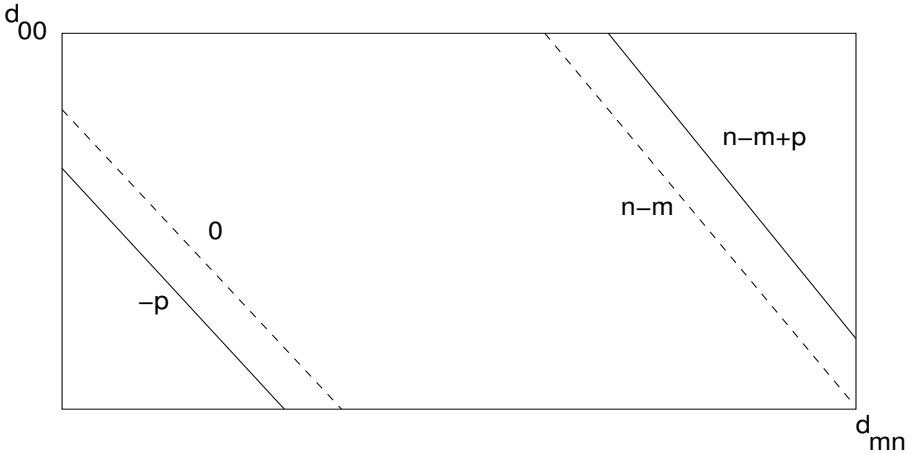


Figura 4.2: *Diagonais $-p, 0, n - m$ e $n - m + p$ quando $m \leq n$*

O algoritmo 3 reúne as ideias expostas até agora. Ele supõe que as células C_{ij} estão inicializadas com ∞:

Algoritmo 3 Teste de Ukkonen ($d_{ukk1}(x, y, \lambda)$)

Entrada:
x, y sequências de símbolos
λ limiar de erro
Saída:
valor lógico indicando se $d(x, y) \leq \lambda$

```
1  se λ/Δ < |n − m| então
2    devolva FALSE
3  senão
4    p ← ⌊½((λ/Δ) − |n − m|)⌋
5    para i ← 0 até m faça
6      se n ≥ m então
7        para j ← max(0, i − p) até min(n, i + (n − m) + p) faça
8          Computar C_ij a partir de 4.3
9        fim
10     senão
11       para j ← max(0, i + (n − m) − p) até min(n, i + p) faça
12         Computar C_ij a partir de 4.3
13       fim
14     fim
15   fim
16   se C_mn ≤ λ então
17     devolva FALSE
18   senão
19     devolva FALSE
20   fim
21 fim
```

O algoritmo 3 avalia (no caso não trivial $\lambda/\Delta \geq |n-m|$) a banda de (C_{ij}) que consiste das diagonais $1+|n-m|+2p$. Já que cada diagonal contém no máximo $min(m,n)$ células e $1+|n-m|+2p \leq 1+\lambda/\Delta = O(\lambda)$, o algoritmo 3 avalia $O(\lambda \cdot min(m,n))$ células. A sua complexidade computacional é portanto $O(\lambda \cdot min(m,n))$. Com respeito ao espaço o algoritmo 3 gasta $O(\lambda \cdot min(m,n))$ armazenando somente as células na banda.

Podemos, no entanto, observar que para computar a seguinte linha da banda precisamos somente da linha anterior. Cada linha contém $1+|n-m|+2p = O(\lambda)$ elementos, logo a complexidade de espaço reduz para $O(\lambda)$. No algoritmo 4 os elementos do arranjo $r_0, r1, ..., r_{|n-m|+2p}$ são utilizados para armazenar sucessivamente as linhas da banda diagonal e r_{-1} e $r_{|n-m|+2p+1}$ são valores sentinela. Inicialmente, $r_i = \infty$ para todo i. Suponha também que $\delta(X \rightarrow Y) = \infty$ se $Y = y[h]$ onde $h < 0$ ou $h > n$:

Algoritmo 4 Teste de Ukkonen ($d_{ukk2}(x, y, \lambda)$)

Entrada:
x, y sequências de símbolos
λ limiar de erro
Saída:
valor lógico indicando se $d(x,y) \leq \lambda$

1 **se** $\lambda/\Delta < |n-m|$ **então**
2 **devolva** $false$
3 **senão**
4 $p \leftarrow \lfloor \frac{1}{2}((\lambda/\Delta) - |n-m|) \rfloor$
5 **se** $n \geq m$ **então**
6 $k' \leftarrow k \leftarrow -p$
7 **senão**
8 $k' \leftarrow k \leftarrow -p + (n-m)$
9 **fim**
10 **para** $i \leftarrow 0$ até m **faça**
11 **para** $j \leftarrow 0$ até $|n-m|+2p$ **faça**
12 **se** $i = j+k$ e $j+k = 0$ **então**
13 $r_j \leftarrow 0$
14 **senão**
15 $$r_j \leftarrow min \begin{cases} r_j + \delta(x[i] \rightarrow y[j+k]) \\ r_{j+1} + \delta(x[i] \rightarrow \epsilon) \\ r_{j-1} + \delta(\epsilon \rightarrow y[j+k])) \end{cases}$$
16 **fim**
17 **fim**
18 $k \leftarrow k+1$
19 **fim**
20 **fim**
21 **se** $r_{|n-m|+2p+k'} \leq k$ **então**
22 **devolva** $true$
23 **senão**
24 **devolva** $false$
25 **fim**

Até aqui vimos algumas técnicas de Busca Aproximada e foram revisados alguns algoritmos de Programação Dinâmica. O algoritmo 4 é utilizado muitas vezes em nosso

sistema, nas componentes de busca e alinhamento de textos.

4.2 Recuperação de Informação

Em Recuperação de Informação (RI), o problema central consiste em recuperar *documentos de texto* que possuem uma ou mais ocorrências de uma sequência de palavras, ou *consulta*, submetida pelo usuário.

Em um sistema de RI um documento é obtido a partir de uma sequência de símbolos. Existem muitas definições de documento, mas neste trabalho ficaremos com a mais genérica onde um documento é um arquivo de texto.

Dada a definição de documento e uma sequência de símbolos, a *fragmentação* quebra o documento em *fragmentos* de símbolos. Esses fragmentos são construídos de acordo com a definição de fragmento para o conjunto de caracteres em que o documento está escrito. Por exemplo, na figura 4.3 foram extraídos fragmentos para um documento escrito em Unicode ([DL17]).

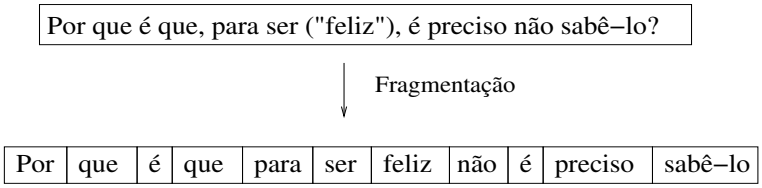


Figura 4.3: *Fragmentação de uma sequência de símbolos em palavras utilizando as regras do Unicode.*

Fragmentos são geralmente conhecidos por palavras ou termos. Existe no entanto uma diferença importante entre esses dois conceitos. Uma *palavra* é a ocorrência de uma sequência de símbolos (fragmento) em um documento. Por outro lado, um *tipo* é a classe de todas as palavras que contém o mesmo fragmento. Um *termo* é um tipo que é utilizado no dicionário de um sistema de RI.

Depois de fragmentar os documentos (e também a consulta do usuário) podemos supor que a consulta possui fragmentos que ocorrem em um documento. No entanto, existem muitos casos onde não é assim e superar a desigualdade de duas sequências de símbolos é vantajosa. Por exemplo, se fizermos uma busca por "USA", gosta riamos de recuperar documentos que contenham o fragmento "U.S.A.".

A *normalização* é o processo de canonização dos fragmentos de maneira que igualdades ocorrem apesar de diferenças superficiais nas sequências de símbolos dos fragmentos. A forma padrão para normalizar é criar implicitamente *classes de equivalência*, que são normalmente identificadas com respeito a um membro do conjunto. Por exemplo, se os fragmentos "pós-graduação" e "pósgraduação" são ambos mapeados para o termo "pós-graduação", nos documentos e nas consultas, então as buscas por um termo vão recuperar documentos que contêm ambos.

Podemos definir de maneira formal um documento $d = d[1],...,d[m]$ de tamanho m como uma sequência de palavras normalizadas. As palavras pertencem a um alfabeto Σ. Seja $d[i,j]$ uma *segmento de palavras* $d[i],d[i+1],...,d[j]$ de d que começa na posição i e termina em j. Um *k-grama de palavras* de d é um segmento de tamanho k.

De forma gráfica podemos mostrar os conceitos acima definidos na figura 4.4. Nessa figura temos dois documentos identificados com "1" e "2" onde o primeiro texto tem 10 palavras enquanto que o segundo possui 11 palavras. Cabe destacar que a forma em que são obtidas as palavras a partir de um texto é um processo que dependem de vários critérios.

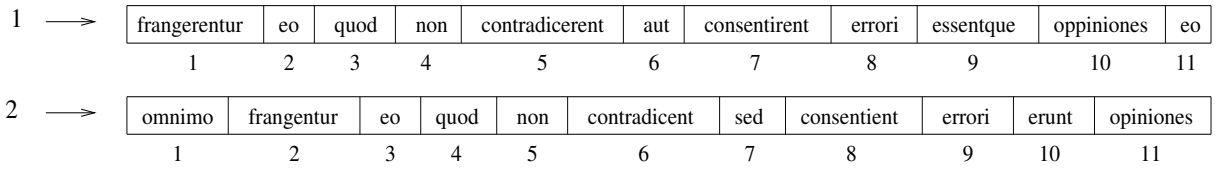


Figura 4.4: *Dois documentos de texto.*

Nas seguintes seções vamos mostrar algoritmos e estruturas de dados que ajudam na resolução do problema de RI.

4.2.1 Busca, dicionários e índices

É bastante razoável, baseados na noção de documento, que existam coleções de documentos. Esta forma de agrupar segue geralmente algum critério: obras de algum autor ou de algum assunto em particular. De maneira formal, podemos definir uma coleção C de tamanho n como o conjunto de documentos $\{d_1, ..., d_n\}$. A figura 4.4 mostra uma coleção de tamanho dois.

A principal tarefa de um sistema de busca é obter os documentos da coleção C que satisfazem uma consulta. A consulta é expressa por palavras, mas pode conter também operadores lógicos. É necessário ter alguma forma de indexação para poder encontrar esses documentos em um tempo razoável pois, caso contrário, será necessário iterar em todos os documentos procurando pela consulta fornecida pelo usuário a cada busca.

Assim, surge de forma natural a estrutura de dicionário. Esta estrutura contém todas as palavras s que ocorrem nalgum documento d da coleção C e pode ser definido como:

$$\Gamma_C = \{s \mid \exists\, d \in C, \exists i, 0 \leq i < |d|, s = d[i]\}.$$

O dicionário é a base para uma estrutura ainda mais complexa que é o *índice invertido*. Para construir este índice é necessário obter os pares de ocorrências $(w, docId)$ onde w representa um termo do dicionário contido no documento $docId$. Este par é chamado de `posto` (no inglês, *posting*) e para cada termo w do dicionário Γ_C designa-se a *lista de postos de w* como sendo a lista ordenada de todos os postos que contêm w como primeira componente. As listas de postos devem estar ordenadas segundo um critério uniforme, normalmente pelo próprio identificador do documento, para que seu processamento seja mais eficiente.

Assim:

$$\Lambda_C(s) = ord(\{i \mid (d_i \in C) \text{ e } (\exists j : s = d_i[j])\}),$$

onde Λ_C é o índice invertido (a função ord mantém a lista de postos ordenada). Na figura 4.5 podemos observar o índice invertido para a coleção $C = \{d_1, d_2\}$ formada pelos dois documentos da figura 4.4. O dicionário Γ_C está conformado por todas as palavras presentes em ambos os documentos e, de forma específica, a lista de postos para a palavra `contradicerent` esta dada por $\Lambda_C(contradicerent) = \{1\}$.

Com esta estrutura de dados podemos realizar *buscas booleanas*. Neste tipo de busca, a consulta utiliza operadores lógicos AND, OR, NOT para expressar a necessidade do usuário. Por exemplo, dados os documentos da figura 4.4 e o índice invertido da figura 4.5, podemos formular a seguinte consulta: `essentque` AND `contradicerent`. Para encontrar os documentos que satisfazem essa consulta obtemos a lista de postos de cada termo para em seguida realizar uma intercalação das listas. Esta intercalação é feita de forma semelhante à fase de intercalação (merge) do algoritmo Merge Sort, que une duas listas

ordenadas numa única lista ordenada, mas mantém somente os documentos que ocorrem em ambas as listas. Isto exige que a lista de postos esteja ordenada para que a operação de intercalação seja eficiente. Assim, para a consulta `essentque` *AND* `contradicerent`, o resultado é uma lista de documentos que contém apenas o documento "1".

Muitos conceitos técnicos ou nomes de produtos e organizações são expressados como frases. Gostaríamos poder submeter a consulta *Universidade de São Paulo* como uma frase e não recuperar um documento como *A cidade de São Paulo tem pelo menos uma universidade*. Muitos motores de busca recentes suportam a sintaxe de aspas duplas ("universidade de são paulo") para *consultas por frase*, que tem-se provado ser facilmente compreendidas e usadas com sucesso pelos usuários. Aproximadamente 10% das consultas na Internet são consultas por frase, e muitas mais são consultas por frase implícitas (tais como nomes de pessoas), submetidas sem aspas duplas. Para dar suporte a esse tipo de consulta, não é suficiente manter listas de postos que sejam listas simples de documentos que contêm termos individuais.

frangerentur	→	1	eo	→	1 2	quod	→	1 2
non	→	1 2	contradicerent	→	1	aut	→	1
consentirent	→	1	errori	→	1 2	essentque	→	1
oppiniones	→	1	omnimo	→	2	frangentur	→	2
contradicent	→	2	sed	→	2	consentient	→	2
erunt	→	2	opiniones	→	2			

Figura 4.5: *Índice invertido para a coleção formada pelos dois documentos da figura 4.4. O dicionário* Γ_C *esta conformado por todas as palavras presentes em ambos os documentos.*

Uma solução é associar a cada posto (w, d) a lista ordenada P de posições j tais que $w = d[j]$. Tais listas de postos são ditas *posicionais* e são úteis para a busca de frases. Assim, definimos o índice invertido com listas posicionais:

$$\Pi_C(s) = \{(i, P) | d_i \in C \wedge \forall p \in P, 1 \leq p \leq |d_i| \wedge s = d_i[p]\}, s \in \Gamma_C$$

Por exemplo, na figura 4.6, temos que a lista de postos para a palavra `contradicerent` esta dada por $\Lambda_C(contradicerent) = \{(1, < 5 >)\}$.

Assim, com este novo índice, podemos formular a consulta "`eo quod non`". O algoritmo realiza basicamente uma intercalação das listas de postos correspondentes às palavras da consulta. Essa intercalação aproveita-se do fato que as listas de postos estão ordenadas para que o processamento e intercalação das listas seja feita em tempo linear na soma dos comprimentos das listas.

Um índice invertido posicional permite processar buscas por frases mas ele não oferece a capacidade de resolver buscas onde os termos não existem no dicionário mas assemelham-se a alguns deles.

Este problema pode ser resolvido utilizado técnicas de Busca Aproximada para os termos de um dicionário. Na continuação vamos mostrar um filtro que encontra os termos de um dicionário que se aproximam de um termo de entrada. Esses novos termos contribuem com listas de postos que são processadas para satisfazer uma consulta.

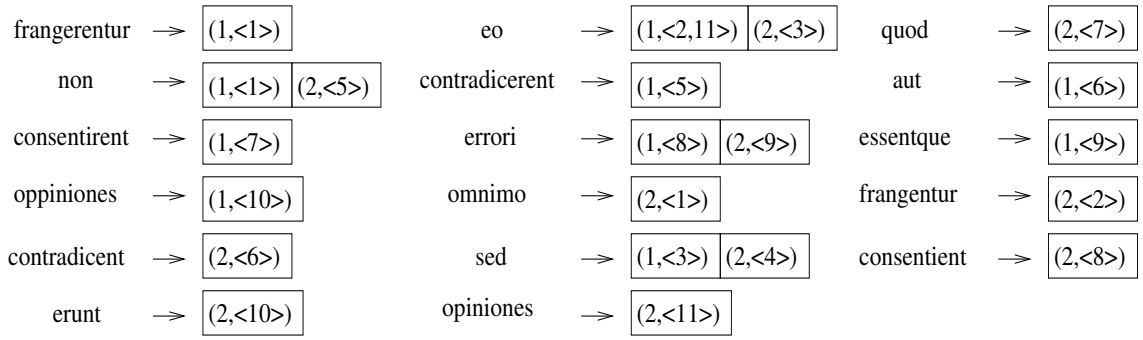


Figura 4.6: *Índice invertido com listas de postos que consideram as posições das palavras nos documentos.*

4.2.2 Busca aproximada em dicionários

Uma das técnicas utilizadas para resolver o problema da busca aproximada é gerar um índice de k-gramas para os termos de um dicionário. O índice de k-gramas é análogo ao índice invertido mostrado na seção 4.2.1. A analogia consiste em tratar os k-gramas como termos e os termos do dicionário como documentos. Assim, o índice de k-gramas mapeia os termos onde um k-grama ocorre. Podemos definir um índice de k-gramas da seguinte forma:

$$\Psi_C(v) = \{w \mid w \in \Gamma_C \text{ e } \exists i, v = w[i, i+k-1]\}$$

onde v é um k-grama.

Por exemplo, na figura 4.7, podemos observar que o índice de k-gramas para $v =$ "ntr" é Ψ("ntr") = {contradicerent, contradicent}.

Para cada termo t de uma consulta é construída a lista de k-gramas. Logo, utilizando o dicionário Ψ_C, são coletadas os termos que tem k-gramas em comum com a lista de k-gramas de t. O número mínimo depende de uma quantidade mínima de erros admitidos.

Alguns critérios podem ser aplicados para filtrar esses termos: Por exemplo, podemos querer ficar com os termos que tiverem pelo menos dois k-gramas em comum com t ou podemos utilizar o coeficiente de Jaccard para determinar quanto ambas as palavras se assemelham. De forma mais específica, seja A o conjunto de k-gramas de t e B o conjunto de k-gramas de um termo do dicionário, logo se o coeficiente de Jaccard $|A \cap B|/|A \cup B|$ for maior que um limiar (definido pelo usuário) então consideramos o termo do dicionário como palavra aproximada.

Considere por exemplo a consulta "`contra`" e o índice da figura 4.7. Para a palavra "`contra`" a lista de termos {`contradicerent`,`contradicent`} possuem pelo menos um k-grama em comum. O coeficiente de Jaccard para "`contra`" e "`contradicerent`" é $4/(12+4-4) = 1/3 = 0.33$, enquanto que para "`contradicent`" o coeficiente é $4/(10+4-4) = 2/5 = 0.4$. Se decidirmos ficar somente com os termos com coeficiente de Jaccard maior ou igual a 0.4 então ficaremos com "`contradicent`". A lista de postos desse termo será considerada quando for realizada a intercalação das listas de postos dos termos da consulta.

A técnica mostrada pode ser também de utilidade para expandir consultas e assim ampliar os conceitos que podem ser buscados no sistema.

4.2.3 Expansão de Consulta

Na expansão de consulta os usuários contribuem com informação adicional sobre os termos da consulta, possivelmente sugerindo termos adicionais. Alguns motores de busca

con → consentirent contradicent contradicerent consentient
ont → contradicerent contradicent
ntr → contradicerent contradicent
tra → contradicerent contradicent
one → oppiniones opiniones

Figura 4.7: *Parte do índice de k-gramas de palavras (k = 3) para o dicionário da figura 4.5*

(especialmente na Internet) sugerem consultas relacionadas como resposta à consulta; os usuários podem decidir utilizar uma das sugestões de consulta. A questão central nesta forma de expansão de consulta é como gerar consultas alternativas ou expandidas para o usuário. A forma mais comum de expansão de consulta é a análise global, utilizando alguma forma de dicionário de sinônimos. Para cada termo t de uma consulta, a consulta pode ser automaticamente expandida com sinônimos e termos relacionados de t a partir do dicionário de sinônimos. O dicionário de sinônimos pode ser substituído também por um dicionário de k-gramas e um algoritmo para encontrar termos relacionados como o mostrado na seção 4.2.2. Os novos termos podem ser combinados com pesos, podemos querer dar menos peso aos novos termos do que aos termos originais.

A vantagem da expansão de consulta com dicionários de sinônimos tem a vantagem de não precisar nenhuma interação com o usuário.

Existem também dicionários manuais mantidos por curadores. Aqui, existe um termo canônico para cada conceito. A utilização de um dicionário controlado é bastante comum em alguns domínios. Um exemplo bastante conhecido é o Unified Medical Language System (UMLS) utilizado pelo MedLine para consultar literatura em pesquisa biomédica. Por exemplo, na figura 4.8, o termo "neoplasms" foi adicionado na busca por "cancer".

consulta do usuário:	cancer
consulta expandida:	"neoplasms"[MeSH Terms] OR "neoplasms"[All Fields] OR "cancer"[All Fields]
consulta do usuário:	skin itch
consulta expandida:	"pruritus"[MeSH Terms] OR "pruritus"[All Fields] OR ("skin"[All Fields] AND "itch"[All Fields]) OR "skin itch"[All Fields]

Figura 4.8: *Alguns exemplos de expansão de consulta utilizando o dicionário de sinônimos do PubMed. Quando o usuário submete uma consulta na interface do PubMed para MedLine em http://www.ncbi.nlm.nih.gov/entrez/, a consulta é mapeada no vocabulário do MedLine.*

Nesta seção foram mostrados conceitos fundamentais em Recuperação de Informação que pretendemos utilizar em nosso sistema. Foram apresentados os conceitos de coleção, índices e dicionários. Além disso foi mostrado também um filtro de busca aproximada sobre esses dicionários. Esse algoritmo se baseia no coeficiente de Jaccard para calcular as lista de palavras aproximadas para o termo de uma consulta. A lista de palavras aproximadas servem para expandir as listas de postos usadas para recuperar documentos.

Esses conceitos são úteis para situar a contribuição da próxima seção que mostra um filtro mais eficiente que utiliza ordenação em tempo linear e o algoritmo 4 para obter a lista de palavras próximas ao termo de uma consulta.

4.3 Busca aproximada por k-gramas e sacos de símbolos

Existe na literatura de Busca Aproximada um tipo de filtro conhecido como *filtro de contagem* ([JTU96, Nav97, BCP02]).

Para um termo t o filtro de contagem utiliza um índice de k-gramas para coletar um subconjunto de palavras de Γ_C que possuem algum k-grama em comum com t. Em seguida, para cada termo desse conjunto e o termo t é computada a *distância por sacos de símbolos*.

Seja x uma palavra sobre o alfabeto Σ, o *saco de símbolos* $\eta(x)$ é computado ordenando os símbolos de x seguindo uma ordem total sobre os símbolos do alfabeto Σ. Por exemplo, se $x =$ "pata" então o seu saco de símbolos é $\eta(x) = (a, a, p, t)$.

Seja $\eta_C(x)$ o saco de símbolos pré-computado para um dicionário Γ_C. Essa versão do saco de símbolos utiliza uma tabela de espalhamento que armazena o saco de símbolos $\eta(x)$ para uma palavra x do dicionário Γ_C.

Dadas duas palavras x e y, a distância por sacos de símbolos $d_{bag}(x, y)$ está definida da seguinte maneira:

$$d_{bag}(x, y) = max\{|\eta(x) - \eta(y)|, |\eta(y) - \eta(x)|\}$$

Na prática, a função $d_{bag}(x, y)$ descarta os símbolos em comum entre x e y, e logo obtém o máximo considerando os elementos "residuais". Por exemplo:

$$\begin{aligned} d_{bag}(\text{"batata"}, \text{"pata"}) &= max\{|\,(aaabtt) - (aapt)\,|, |\,(aapt) - (aaabtt)\,|\} \\ &= max\{|\,(abt)\,|, |\,(p)\,|\} \\ &= 3 \end{aligned}$$

O algoritmo 5 mostra um pseudocódigo do cálculo da distância d_{bag}. Ela recebe como parâmetros as sequências x e y de tamanho $m = |x|$ e $n = |y|$ respectivamente e a taxa de erro λ.

Em seguida, entre as linhas 1 e 2, são criados os sacos de símbolos x' e y' utilizando Counting Sort. O número de erros na distância d_{bag} está definido pelos tamanhos dos conjuntos diferença $x' - y'$ e $y' - x'$. O tamanho para cada conjunto é armazenado nas variáveis d_x e d_y respectivamente (linha 3). As variáveis i e j definidas na linha 4 servem como controle para iterar em x' e y' respectivamente. Finalmente a variável *check*, definida na linha 5, serve como variável de controle para o laço entre as linhas 6- 32.

O laço atualiza o tamanhos do conjunto $x' - y'$ na variável d_x (linhas 9 e 28). Na linha 9 ele é atualizado quando o símbolo $x'[i]$ é menor que $y'[j]$, enquanto que na linha 28 ele é atualizado quando $|x| > |y|$. De forma análoga o tamanho do conjunto $y' - x'$ é atualizado na variável d_y na linha 15 e na linha 26 quando $|y| > |x|$.

O algoritmo exibe duas formas de terminação prematura: nas linhas 11 e 17 é verificado se $max\{d_x, d_y\} > \sigma'$. Caso a comparação seja verdadeira a variável *check* é atualizada e na seguinte iteração o laço finaliza. A segunda forma de terminação ocorre entre as linhas 25 e 30 onde são verificados dois casos: $m > n$ ou $m < n$. Em ambos os casos não existe razão para continuar a iteração e a variável *check* é atualizada.

A complexidade computacional do algoritmo é dominada pelo Counting Sort das linhas 1- 2. Ambas as linhas contribuem com uma complexidade $O(m + n)$ e pelo laço das linhas 6- 32 que, no pior caso quando $\lambda = max\{m, n\}$, possui complexidade $O(min(m, n))$.

Quanto à utilização de espaço este é dominado pelo Counting Sort das duas sequências x e y que é $O(m + n)$.

Suponha que os sacos de símbolos x' e y' estivessem disponíveis. Vamos adaptar a

chamada de $d_{bag}(x, y, \lambda)$ para poder receber esses sacos de símbolos como parâmetros. Assim, a nova chamada é $d_{bag}(x, x', y, y', \lambda)$.

A complexidade total dessa nova versão é $O(min(m, n))$ pois evita completamente a computação dos sacos de símbolos e a execução é dominada pelo laço das linhas 6- 32.

Na seguinte seção vamos mostrar como utilizar essa otimização da distância por sacos de símbolos na construção de uma estrutura de dados simples para filtrar um conjunto de palavras de um dicionário de termos.

4.3.1 Filtro simples no dicionário de sacos de símbolos

Com as estruturas e algoritmos que foram mostrados até agora podemos implementar um filtro que descarta palavras de um dicionário Γ_C que não possuem muito erro com respeito a um termo t.

A taxa de erro no filtro está especificada por um parâmetro $\beta \in \mathbb{R}, 0 < \beta \leq 100$ que ajusta o número máximo de erros para duas palavras t e w. O número máximo de erros esta definido pela função $\sigma(x, y, \beta) = \frac{\beta}{100} \cdot max\{|x|, |y|\}$.

O Algoritmo 6 mostra o pseudocódigo desse filtro. Ele procede em três etapas:

1 Busca por k-gramas O algoritmo começa gerando os k-gramas de t são gerados na linha 1. Eles são armazenados no conjunto ψ_t. Em seguida os k-gramas de ψ_t são procurados em Ψ_C e as palavras que tiverem pelo menos um "hit" são armazenadas no conjunto W_1.

2 Filtro por sacos de símbolos As palavras coletadas na etapa anterior são avaliadas utilizando a distância por sacos de símbolos d_{bag} na linha 11. Para tanto, é obtido o saco de símbolos t' de t na linha 7. Em seguida, entre as linhas 8- 14, é computada a distância d_{bag} para cada palavra w de W_1. Essa computação é realizada encima dos sacos de símbolos t' e w'. Especificamente w' é obtido mediante a chamada $\eta_C(w)$ que possui uma tabela de espalhamento que armazena todos os sacos de símbolos do dicionário Γ_C. Além dos sacos de símbolos, a taxa de erro σ' para t e w é calculada na linha 9 e utilizada na chamada da distância d_{bag}.

3 Filtro por Teste de Ukkonen (d_{ukk}) Na etapa final as palavras coletadas no filtro por sacos de símbolos são verificadas utilizando o Teste de Ukkonen (linhas 16- 21). As palavras que forem aceitas neste filtro são armazenadas no conjunto Γ_t que é a saída do algoritmo.

Para um termo t, o algoritmo $FastFilter$ computa corretamente o subconjunto $\Gamma_t \subset \Gamma_C$ que contém as palavras que apresentam uma taxa de erro β.

Com respeito à complexidade computacional considere o pior caso quando $\beta = 100$. Na etapa 1 o algoritmo obtém em tempo $O(|t| - k + 1)$ o conjunto de palavras $W_1 \subset \Gamma_C$ que possuem pelo menos 1 k-grama em comum com t.

Na etapa 2, para cada palavra $w \in W_1$ é executada a distância de saco de símbolos d_{bag}. Como já temos disponível os sacos de símbolos t' e w', então a distância d_{bag} vai demorar $O(min(|t|, |w|))$. Se supormos que o comprimento médio das palavras em W_1 é $\bar{w}_1$, então a complexidade da etapa 2 é $O(|W_1| \cdot min(|t|, \bar{w}_1))$.

Na terceira etapa, o Teste de Ukkonen d_{ukk} é executado para t e cada palavra $w \in W_2$. A execução precisa do número de erros máximo entre t e w calculado na linha 19. Assim como na análise anterior, suponha que o comprimento médio das palavras em W_2 é $\bar{w}_2$. A complexidade da etapa 3 é portanto $O(|W_2| \cdot \bar{\sigma} \cdot min(|t|, \bar{w}_2))$ onde $\bar{\sigma} = max(|t|, \bar{w}_2)$. Logo, a complexidade é $O(|W_2| \cdot max(|t|, \bar{w}_2) \cdot min(|t|, \bar{w}_2))$.

Algoritmo 5 Distância por Saco de Símbolos ($d_{bag}(x, y, \lambda)$)

Entrada:
x, y uma sequências de símbolos
λ taxa de erro
Saída:
valor lógico que indica se $d_{bag}(x, y,) \leq \lambda$

```
1  x' ← η(x)
2  y' ← η(y)
3  d_x ← d_y ← 0
4  i ← j ← 0
5  check ← true
6  enquanto check = true faça
7     se i < |x| e j < |y| então
8        se x'[i] < y'[j] então
9           d_x = d_x + 1
10          se max{d_x, d_y} > λ então
11             check ← false
12          fim
13          i = i + 1
14       senão se x'[i] > y'[j] então
15          d_y = d_y + 1
16          se max{d_x, d_y} > λ então
17             check ← false
18          fim
19          j = j + 1
20       senão
21          i ← i + 1
22          j ← j + 1
23       fim
24    senão
25       se i ≥ |x| e j < |y| então
26          d_y = d_y + (|y| − j)
27       senão se i < |x| e j ≥ |y| então
28          d_x = d_x + (|x| − i)
29       fim
30       check = false
31    fim
32 fim
33 devolva  max{d_x, d_y} ≤ λ
```

Algoritmo 6 Filtro de busca aproximada(*FastFilter*)

Entrada:

t uma palavra,
Γ_C dicionário de termos de uma coleção C
Ψ_C índice de k-gramas do dicionario Γ_C
η_C sacos de símbolos pré-computados para a coleção C
λ taxa de erro

Saída:

Γ_t lista de palavras próximas

1 Inicializar ψ_t com os k-gramas de t
2 $W_1 \leftarrow \emptyset$
3 **para** $w \in \psi_t$ **faça**
4 $\quad W_1 \leftarrow W_1 \cup \Psi_C(w)$
5 **fim**
6 $W_2 \leftarrow \emptyset$
7 $t' \leftarrow \eta(t)$
8 **para** $w \in W_1$ **faça**
9 $\quad \sigma' \leftarrow \sigma(t, w, \beta)$
10 $\quad w' \leftarrow \eta_C(w)$
11 $\quad$ **se** $d_{bag}(t, t', w, w', \sigma')$ é verdadeiro **então**
12 $\quad\quad W_2 \leftarrow W_2 \cup \{w\}$
13 $\quad$ **fim**
14 **fim**
15 $\Gamma_t \leftarrow \emptyset$
16 **para** $w \in W_2$ **faça**
17 $\quad \sigma' \leftarrow \sigma(t, w, \beta)$
18 $\quad$ **se** $d_{ukk}(t, w, \sigma')$ é verdadeiro **então**
19 $\quad\quad \Gamma_t \leftarrow \Gamma_t \cup \{w\}$
20 $\quad$ **fim**
21 **fim**
22 **devolva** Γ_t

A complexidade total do filtro no pior caso é $O((|t|-k+1)+|W_1|\cdot min(|t|,\bar{w}_1)+|W_2|\cdot max(|t|,\bar{w}_2)\cdot min(|t|,\bar{w}_2))$.

O filtro apresentado é utilizado em dois módulos do nosso sistema: Na Expansão de Consultas e no Alinhamento de Textos (capítulo 5). A complexidade do filtro, no pior caso quando $\beta = 100$, é dominada pela etapa 3. No entanto, na prática, o valor de β geralmente oscila entre 30 e 40 aumentando a utilização da Etapa 2 onde é computada a distância por sacos de símbolos e, consequentemente, diminuindo as chamadas para o Teste de Ukkonen d_{ukk2} na Etapa 3. Podemos formular também uma versão mais ótima do filtro se o saco de símbolos do parâmetro t estiver disponível. Nesse caso podemos evitar a computação da linha 7 e o filtro passa a aceitar mais um parâmetro com o saco de símbolos t' ficando a chamada $FastFilter(t,t',\Gamma_C,\Psi_C,\eta_C,\lambda)$. Na última seção vamos mostrar como este filtro é utilizado para expandir consultas formuladas pelo usuário.

4.3.2 Expansão de Consulta por filtros de k-gramas

A interface de busca projetada para o sistema utiliza o filtro mostrado anteriormente para expandir os termos de uma consulta. Uma consulta q é uma sequência de palavras com comprimento bastante menor que um documento e possui termos que podem não existir em uma coleção C.

A expansão de consulta encontra termos próximos para cada termo de q e gera novas consultas ampliadas pelos termos próximos. O procedimento completo é mostrado nos algoritmos 7 e 8. O algoritmo *QueryGeneration* da listagem 7 permite a criação de novas consultas a partir de q. A consulta é tokenizada para obter os termos w para logo encontrar os seus termos aproximados na coleção C (linha 3). Em seguida para cada novo termo aproximado w' é gerada uma nova consulta $\tilde{q}$ substituindo w. Esta estratégia limita o número de novas consultas que são geradas.

O algoritmo *BooleanExpansion* da listagem 8 utiliza a saída do algoritmo de geração de consultas *QueryGeneration* da listagem 7 para criar uma consulta lógica disjuntiva (OR). Cada consulta individual é uma modificação da consulta original q e possui um peso w que indica ao motor de busca como calcular a pontuação de cada documento retornado.

Vamos denotar por q^w uma consulta q com peso w onde $w > 0$. No algoritmo são utilizadas três constantes que denotam os distintos tipos de pesos adotados: w_1 corresponde com o peso da consulta original, w_2 corresponde com o peso da consulta normalizada, isto é, a consulta original mas sem símbolos de acentuação e w_3 corresponde a todas as consultas que possuem termos próximos. Os pesos possuem uma hierarquia, em outras palavras, $w_1 > w_2 > w_3$, ou seja damos mais prioridade aos termos originais, depois aos termos normalizados e por último aos termos próximos.

Na linha 1 o algoritmo *BooleanExpansion* adiciona a consulta original q com peso w_1 na consulta lógica $bool\_q$. Em seguida são geradas todas as consultas com termos próximos na linha 2. Cada consulta recebe o peso w_3 e é adicionada na consulta lógica $bool\_q$ (linhas 3-5).

Na linha 6 é gerada a consulta $\tilde{(q)}$ que é a normalização de q. A normalização tira os símbolos de acentuação. Se as duas consultas $\tilde{q}$ e q forem diferentes, então são geradas as consultas com termos próximos para $\tilde{q}$ e estas são adicionadas com peso w_3.

Algoritmo 7 Geração de consultas(*QueryGeneration*)

Entrada:
q uma consulta
Γ_C dicionário de termos de uma coleção C
Ψ_C índice de k-gramas do dicionario Γ_C
η_C sacos de símbolos pré-computados para a coleção C
β taxa de erro
Saída:
Q conjunto de consultas finais

```
1 Q ← ∅
2 para w ∈ q faça
3    W ← FastFilter(w, Γ_C, Ψ_C, η_C, β)
4    para w' ∈ W faça
5       Criar nova consulta q̃ substituindo w' por w
6       Q ← Q ∪ q̃
7    fim
8 fim
9 devolva Q
```

Algoritmo 8 Expansão de consulta(*BooleanExpansion*)

Entrada:
q uma consulta
Γ_C dicionário de termos de uma coleção C
Ψ_C índice de k-gramas do dicionario Γ_C
β taxa de erro
w_1 peso da consulta principal (constante)
w_2 peso da consulta normalizada (constante)
w_3 peso das consultas próximas (constante)

```
1  bool_q ← q^{w_1}
2  Q_1 ← QueryGeneration(q, Γ_C, Ψ_C, β)
3  para q̂ ∈ Q_1 faça
4     bool_q ← bool_q ∨ q̂^{w_3}
5  fim
6  Normalizar q e armazenar em q̃
7  se q̃ ≠ q então
8     bool_q ← bool_q ∨ q̃^{w_2}
9     Q_2 ← QueryGeneration(q̃, Γ_C, Ψ_C, β)
10    para q̂ ∈ Q_2 faça
11       bool_q ← bool_q ∨ q̂^{w_3}
12    fim
13 fim
14 Executar bool_q no motor de busca
```

4.4 Conclusões

Neste capítulo apresentamos algoritmos em Busca Aproximada e Recuperação de Informação que resolvem o problema da busca em documentos com erro no nosso sistema. Os algoritmos de Busca Aproximada são bastante simples de serem implementados e possuem complexidades razoáveis. Em Recuperação de Informação foi apresentada uma forma de gerar consultas aproximadas com base nos algoritmos de Busca Aproximada. Essa parte do sistema foi implementada encima de um motor de busca Lucene.

Capítulo 5

Alinhamento de documentos de texto

Neste capítulo introduzimos o problema do alinhamento de documentos de texto. Vamos mostrar primeiro, na subseção 5.1, como é realizado o alinhamento de sequências biológicas. O alinhamento local para este tipo de sequências é um problema bastante estudado contando inclusive com heurísticas como o BLAST que tem se mostrado bastante eficientes na prática para encontrar tais alinhamentos. Com respeito ao alinhamento múltiplo de sequências descrevemos as técnicas básicas: alinhamento múltiplo por árvore e por estrela. Ambas as técnicas são aproximações da solução exata que precisa de tempo exponencial para obter um alinhamento múltiplo.

Depois, na subseção 5.2, vamos mostrar a heurística BLASTD para obter alinhamentos locais de documentos de texto. Definimos de maneira formal a similaridade de documentos assim como o custo das operações para converter um documento em outro. Essa heurística faz uso de técnicas de Busca Aproximada definidas no capítulo 4 para encontrar alinhamentos locais.

Finalmente, na subseção 5.3 mostramos como obter alinhamentos múltiplos para documentos de textos a partir de alinhamentos locais obtidos utilizando o BLASTD.

Ao todo, o BLASTD e sua aplicação na obtenção de alinhamentos multiplos locais são a principal contribuição deste capítulo.

5.1 Alinhamento em sequências biológicas

Um dos problemas centrais na área de Biologia Computacional é o de comparar duas sequências de aminoácidos. A comparação consiste basicamente em encontrar as regiões onde as sequências são similares e onde são diferentes. Embora este conceito pareça simples, existe uma grande variedade de problemas com diversas formalizações e que requerem estruturas de dados e algoritmos completamente diferentes para encontrar uma solução eficiente.

Um desses problemas é o de *alinhamento de duas sequências*, que consiste em sobrepor uma sequência sobre outra para poder mostrar a correspondência entre caracteres similares ou palavras a partir das sequências[1]. As sequências podem ter tamanhos diferentes. Considere, por exemplo, as duas sequências de DNA: $GACGGATTAG$ e $GATCGGAATAG$ e um alinhamento entre elas mostrado na figura 5.1:

Um alinhamento, como podemos observar no exemplo apresentado, insere *espaços em lugares arbitrários ao longo das sequências de maneira que ambas fiquem do mesmo tamanho.*

[1] Os conceitos apresentados nesta seção são apresentados originalmente em [SM97]

$$GA{-}CGGATTAG$$
$$GATCGGAATAG$$

Figura 5.1: *Duas sequências de DNA e um alinhamento entre elas (adaptado de [SM97]).*

Assim, podemos sobrepor ambas para criar correspondências entre espaços ou caracteres da primeira sequência e espaços ou caracteres da segunda sequência. Além disso, um alinhamento não pode admitir um espaço de uma sequência ser alinhado com um espaço na outra sequência. Os espaços podem ser inseridos inclusive no começo ou no fim das sequências.

É importante poder atribuir uma *pontuação* a um alinhamento pois isto nos permite ter uma ideia da sua qualidade. Esta pontuação pode ser calculada da seguinte forma: cada coluna do alinhamento possui uma pontuação parcial e a pontuação final do alinhamento é a soma de todas as pontuações parciais atribuídas às colunas. Se dois caracteres na coluna de alinhamento forem iguais atribuímos uma pontuação +1, se forem diferentes a pontuação será -1 e, se houver um espaço, a pontuação será -2 (a escolha por estes pesos é bastante comum na prática). Um exemplo do cálculo da pontuação em um alinhamento é mostrado mostrado na figura 5.1. O alinhamento possui 9 colunas com o mesmo caractere, 1 coluna com caracteres diferentes, e 1 com um espaço obtendo uma pontuação final de 6.

Esta pontuação serve para definir o conceito de *similaridade*. Dadas duas sequências s e t, a similaridade $\text{sim}(s,t)$ é dada pela pontuação final do alinhamento entre s e t. Em geral, podem existir vários alinhamentos com pontuações máximas.

O problema do alinhamento possui duas versões dependendo se desejamos encontrar alinhamentos que envolvem as sequências inteiras ou somente segmentos dessas sequências. A primeira versão chama-se de *alinhamento global* enquanto que a segunda é conhecida como *alinhamento local*.

5.1.1 Alinhamento global

Dadas duas sequências, o alinhamento global tenta encontrar correspondências em todos os caracteres de ambas as sequências e é útil quando as sequências são similares e quase do mesmo tamanho. O algoritmo para resolver o problema de alinhamento global em Biologia Computacional foi proposto por Needleman em 1970 ([NW70]). A solução, baseada em programação dinâmica, calcula todas as similaridades de prefixos arbitrários de duas sequências s e t. De forma precisa, seja m o tamanho de s e n o tamanho de t. Existem $m+1$ possíveis prefixos de s e $n+1$ prefixos de t, incluindo a palavra vazia. Podemos então dispor os cálculos em uma matriz de tamanho $(m+1)\times(n+1)$ onde cada célula (i,j) contém a similaridade entre os prefixos $s[1,i]$ e $t[1,j]$.

A figura 5.2 mostra a matriz que corresponde a $s = AAAC$ e $t = AGC$. Note que a primeira fila e a primeira coluna são inicializadas com penalidades de espaços. Isto se deve a que existe somente um alinhamento possível se uma das sequências é vazia: Basta adicionar tantos espaços quantos caracteres houver na outra sequência. A pontuação para este alinhamento é $-2k$, onde k é o comprimento da sequência não vazia.

Com respeito às outras células, fazemos a seguinte observação: Para calcular a pontuação da célula (i,j), somente é necessário olhar para as células $(i-1,j)$, $(i-1,j-1)$ e $(i,j-1)$. A razão é que somente existem três formas de alinhar $s[1,i]$ e $t[1,j]$, e cada uma

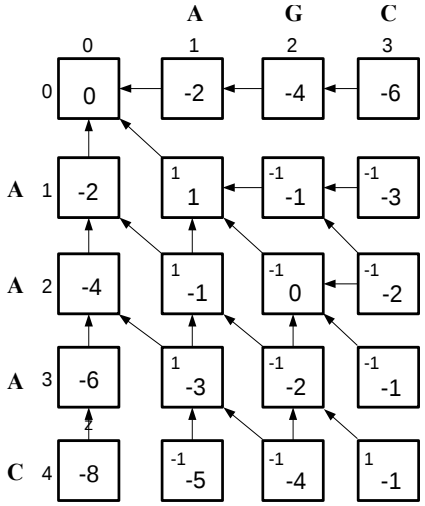


Figura 5.2: *Matriz contendo alinhamentos ótimos. O valor na esquina superior esquerda da célula (i, j) indica se $s[i] = t[j]$. Índices de filas e colunas começam em zero (adaptado de [SM97])*

usa um dos valores prévios. De fato, para alinhar $s[1,i]$ e $t[1,j]$ temos as seguintes opções:

- Alinhar $s[1,i]$ com $t[1,j-1]$ e emparelhar um espaço com $t[j]$, ou
- Alinhar $s[1,i-1]$ com $t[1,j-1]$ e emparelhar $s[i]$ com $t[j]$, ou
- Alinhar $s[1,i-1]$ com $t[1,j]$ e emparelhar $s[i]$ com um espaço

Estas opções são exaustivas porque não podem existir dois espaços emparelhados na última coluna do alinhamento. Como consequência, a similaridade pode ser determinada pela seguinte função:

$$\text{sim}(s[1,i],t[1,j]) = max\begin{cases}\text{sim}(s[1,i],t[1,j-1])-2\\ \text{sim}(s[1,i-1],t[1,j-1])+\delta(s[i]\rightarrow t[j])\\ \text{sim}(s[1,i-1],t[1,j])-2,\end{cases} \tag{5.1}$$

Usamos aqui a mesma notação vista na equação 4.3 e estamos adotando $\delta(s[i] \rightarrow t[j])$ para nos referir à pontuação da substituição de $s[i]$ por $t[j]$, que no caso de DNA, é por exemplo: $+1$ se $s[i] = t[j]$; -1 se $s[i] \neq t[j]$. Na equação 4.3 busca-se minimizar custos associados a distâncias enquanto que na equação 5.1 busca-se, de forma dual, maximizar uma pontuação associada ao grau de semelhança entre os prefixos $s[1,i]$ e $t[1,j]$. Os valores de $\delta(s[i] \rightarrow t[j])$ estão escritos nos cantos superiores esquerdos de cada célula na figura 5.2.

Se armazenarmos os valores $\text{sim}(s[1,i],t[1,j])$ numa matriz de programação dinâmica a, temos que

$$a[i,j] = max\begin{cases}a[i,j-1]-2\\ a[i-1,j-1]+\delta(s[i]\rightarrow t[j])\\ a[i-1,j]-2\end{cases}$$

Observe que a similaridade entre s e t é:

$$sim(s,t) = sim(s[1,m],t[1,n]) = a[m,n] \tag{5.2}$$

Como de costume neste tipo de técnica, é importante preencher ordenadamente a matriz de maneira que os valores $a[i-1,j]$, $a[i-1,j-1]$ e $a[i,j-1]$ já estejam computados ao se calcular cada $a[i,j]$. A forma de preenchimento pode ser, por exemplo, fila a fila, da esquerda para a direita em cada fila.

É possível ainda construir o alinhamento ótimo a partir dos resultados da matriz a. As setas na figura 5.2 serão úteis nesta tarefa. Tudo que precisa ser feito é começar na célula (m,n) e seguir as setas até chegar na célula $(0,0)$. Cada seta utilizada fornece uma coluna do alinhamento. De fato, considere uma seta que sai da célula (i,j). Se esta seta é horizontal, corresponde a uma coluna com um espaço em s emparelhada com $t[j]$; se a seta é vertical, corresponde a $s[i]$ emparelhada com um espaço em t; finalmente, a seta diagonal significa que $s[i]$ e $t[j]$ foram emparelhados. Observe que a primeira sequência, s, está sempre colocada no eixo vertical. Logo, um alinhamento ótimo pode ser facilmente construído da direita para a esquerda se a matriz a foi preenchida da forma descrita anteriormente.

5.1.2 Alinhamento local

O alinhamento local entre duas sequências s e t é um alinhamento global entre um segmento de s e um segmento de t. O algoritmo, proposto por Smith e Waterman ([SW81]) é uma variação do algoritmo de alinhamento global. A diferença reside na interpretação dos valores da matriz a: Cada célula (i,j) armazena a maior pontuação de um alinhamento global entre um sufixo de $s[1,i]$ e um sufixo de $t[1,j]$. A primeira fila e coluna são inicializadas com zeros.

Para qualquer célula (i,j), sempre existe um alinhamento entre os sufixos vazios de $s[1,i]$ e $t[1,j]$, que tem pontuação zero; assim a matriz terá células maiores ou iguais a zero. Depois da inicialização a matriz pode ser preenchida da forma usual com $a[i,j]$ dependendo do valor das três células previamente computadas. A recorrência resultante é

$$a[i,j] = max \begin{cases} a[i,j-1]+g \\ a[i-1,j-1]+\delta(s[i] \rightarrow t[j]) \\ a[i-1],j]+g \\ 0, \end{cases}$$

onde o parâmetro g ("gap penalty") especifica a penalidade da correspondência de um espaço a um caractere. A recorrência é basicamente a mesma do alinhamento global mas possui uma quarta possibilidade, não disponível no alinhamento global, ao de um alinhamento vazio.

Para obter os alinhamentos locais basta encontrar a célula com maior valor na matriz. Esta será a pontuação de um alinhamento local ótimo. Qualquer célula que contém este valor pode ser usada como ponto inicial para obter aquele alinhamento. O resto do alinhamento é obtido seguindo as setas mas parando assim que uma célula não contém uma seta de saída. Alternativamente, podemos parar assim que encontrarmos uma célula com valor zero.

5.1.3 Alinhamento múltiplo

A noção de alinhamento múltiplo é uma generalização natural do caso com duas sequências. Um alinhamento múltiplo que envolve o conjunto de sequências $s_1,...,s_k$ é

obtido mediante a inserção de espaços nas sequências de maneira que todas elas possuam o mesmo tamanho. É comum colocar as novas sequências em uma lista vertical de forma que os caracteres - ou espaços - em posições correspondentes ocupem a mesma coluna. A figura 5.3 mostra o alinhamento múltiplo de quatro sequências de aminoácidos (o alinhamento múltiplo é mais comum em sequências de aminoácidos):

$$
\begin{array}{l}
MQPI\ LLL \\
ML\,R{-}\,LL{-} \\
MK{-}I\ LLL \\
MP\,PVLI\,L
\end{array}
$$

Figura 5.3: *Alinhamento múltiplo de quatro sequências de aminoácidos. A correspondência das letras com aminoácidos é como segue: M representa a Metionina, Q representa a Glutamina, P representa a Prolina, I representa a Isoleucina, L representa a Leucina, R representa a Arginina, K representa a Lisina e V representa a Valina. (adaptado de [SM97])*

Da mesma forma que nos outros tipos de alinhamento, também é possível atribuir uma pontuação a um alinhamento múltiplo. Uma das métricas mais conhecidas é a *soma dos pares* ou SP. Esta métrica calcula a pontuação final do alinhamento múltiplo somando todas as pontuações parciais dos alinhamentos dois-a-dois das sequências de entrada. Suponha, por exemplo, que temos em uma coluna de um alinhamento múltiplo os seguintes valores: $I, \_, I, V$. Logo, a métrica SP sobre esta coluna é calculada da seguinte forma:

$$
\begin{aligned}
SP - score(I, \_, I, V) = & p(I,-) + p(I,I) + p(I,V) + \\
& p(-,I) + p(-,V) + p(I,V)
\end{aligned}
$$

onde $p(a,b)$ é a pontuações para os símbolos a e b. Note que esta função pode incluir um pontuação específica quando a ou b são espaços.

Dada esta métrica podemos ser capazes de computar os alinhamentos de pontuação máxima para um conjunto de sequências. Estes alinhamentos podem ser obtidos utilizando a técnica de programação dinâmica, da mesma forma como foi feito nos outros dois tipos de alinhamento. Infelizmente o problema de alinhamento múltiplo é NP-difícil e, em particular, o algoritmo baseado na métrica SP tem complexidade $O(k^2 2^k n^k)$ (esta complexidade depende de que todas as sequências tenham tamanho $O(n)$). Isto torna este tipo de solução inviável para aplicações práticas.

Existem, no entanto, heurísticas que produzem soluções razoáveis (ou seja, não têm qualquer garantia sobre a qualidade do alinhamento) e possuem tempos de execução menores.

Podemos citar, por exemplo, a heurística *estrela* cuja ideia gira em torno de escolher uma das sequências como o centro s_c e logo alinhar o restante das sequências s_i, $i \neq c$ ao redor de s_c. A complexidade deste algoritmo é $O(kn^2 + k^2 l)$ onde l é um limite superior nos comprimentos dos alinhamentos e n é o comprimento das sequências de entrada (ou seja, todas as sequências tem comprimento $O(n)$).

Por outro lado, existe também a heurística de *árvore*. Suponha que temos um conjunto de k sequências e uma árvore com exatamente k folhas com uma correspondência um-a-um entre as folhas e as sequências. Se atribuímos sequências aos nós interiores da árvore, podemos computar o peso de cada aresta, que corresponde à similaridade entre

duas sequências nos nós incidentes à aresta. A soma de todos estes pesos é a pontuação da árvore com respeito a esta atribuição particular das sequências aos nós. O problema de encontrar uma atribuição das sequências que maximizam a pontuação da árvore é conhecido como *problema de alinhamento de árvores*. O alinhamento estrela pode ser visto como um caso particular do alinhamento de árvore.

O problema de alinhamento de árvores também é NP-difícil mas existem otimizações de espaço e tempo que produzem algoritmos mais práticos. Existem também algoritmos de aproximação com boas taxas de aproximação que são projetados para o caso quando os pesos das arestas são definidos em termos de distância ao invés de similaridade. Em trabalhos mais recentes na área tem surgido algoritmos que filtram a entrada para descartar regiões de pouco interesse nas sequências de entrada ([PSdL+09]) e assim diminuir drasticamente o tempo de execução de algoritmos de alinhamento múltiplo como o GLAM2 ([FHSW04]). Por outro lado, tem surgido também sistemas de alinhamento múltiplo mais eficientes como o KALIGN2 ([LFS09]) que implementam o alinhamento de árvore mas que utiliza um algoritmo de busca inexata eficiente ([Rob]) para calcular a similaridade entre duas sequências.

5.1.4 Comparação de sequências em bancos de dados

Os algoritmos de programação dinâmica para ambos os tipos de alinhamento apresentados tem complexidade computacional $O(|s| \times |t|)$ onde s e t são duas sequências de símbolos. Infelizmente esta complexidade torna o algoritmo inviável quando as sequências de entrada são muito grandes. Este caso ocorre com bastante frequência pois existem cadeias muito grandes a serem comparadas (o cromossoma 1 do genoma humano possui aproximadamente 249×10^6 caracteres).

Dada esta complexidade, foram propostas várias heurísticas que aceleram a busca de similaridade em grandes bancos de dados. Em geral, estas heurísticas possuem uma complexidade de tempo e espaço difíceis de calcular. Por outro lado, estas heurísticas trabalham com critérios de custo mais sofisticados que os apresentados até agora que somente consideram valores -1, 0 e $+1$. A razão por trás desses critérios é que as heurísticas precisam de custos que reflitam de maneira precisa a relação entre as proteínas garantindo assim a sua terminação. Os critérios são representados por *matrizes de substituição* que determinam a relação entre as proteínas.
Nas próximas seções vamos apresentar de maneira de breve os tipos mais comuns de matrizes de substituição assim como o BLAST que é a heurística mais conhecida para encontrar alinhamentos locais.

Matrizes de substituição

Os aminoácidos, que são os componentes básicos das sequências proteicas, possuem propriedades bioquímicas que favorecem a sua substituição. Por exemplo, existem aminoácidos que preferem ser substituídos por outros que apresentam o mesmo tamanho ou que possuam a tendência de se associar com água.

Uma matriz de substituição é indexada em suas linhas e em suas colunas por símbolos do alfabeto Σ e contém em cada posição (x, y) a pontuação associada à substituição do símbolo x pelo símbolo y. Adotamos a mesma notação $\delta(x \to y)$ vista na equação 4.3 para representar a matriz de substituição de um símbolo x do alfabeto Σ por um símbolo y.

Para construir uma matriz de substituição é recomendável observar diretamente a probabilidade de substituição entre aminoácidos para ter uma noção clara das taxas reais de substituição. Este método é utilizado para calcular as matrizes PAM propostas por

Margaret Dayhoff ([DN73]) em 1973. O acrônimo PAM quer dizer *Point Accepted Mutations* ou *Percent of Accepted Mutations* e se refere ao fato que a matriz PAM reflete uma quantidade de evolução que produz, em média, uma quantidade de mutações por centena de aminoácidos.

O método proposto por Dayhoff é produzido levando em conta alinhamentos que possuem pelo menos 85% de identidade. Para sequências que possuem identidades mais fracas foram propostas as matrizes BLOSUM(Block Substitution Matrix) propostas por Henikoff e Henikoff ([HH92]). As matrizes BLOSUM são calculadas a partir blocos gerados de alinhamentos múltiplos de cadeias com pouca similaridade. Para reduzir o viés de segmentos com alta similaridade estes são aglomerados caso tenham uma identidade acima de um certo threshold. A matriz BLOSUM62 possui um threshold de 62%.

Os bioquímicos costumam dividir os vinte aminoácidos em grupos, de acordos com diversas propriedades comuns. Na figura 5.4, pode-se observar que, a grosso modo, a matriz de substituição BLOSUM62 reflete com pontuações positivas as substituições onde os aminoácidos envolvidos pertencem ao mesmo grupo. É o caso de substituições envolvendo dois aminoácidos dentre: os aromáticos (F,W,Y); os básicos (H,K,R); Serina, Treonina ou Prolina (S,T,P); os ácidos (D,E) ou aqueles do grupo amida (N,Q); os alifáticos do subgrupo (A,G); os alifáticos do subgrupo (M,I,L,V). Em geral, são negativas as pontuações de aminoácidos envolvidos em grupos distintos dos mencionados, inclusive quando se substitui um aminoácido do subgrupo (A,G) por outro do subgrupo (M,I,L,V).

As matrizes BLOSUM são as mais utilizadas por heurísticas como a do BLAST.

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W
C	**9**	-1	-1	-3	0	-3	-3	-3	-4	-3	-3	-3	-3	-1	-1	-1	-1	-2	-2	-2
S	-1	**4**	1	-1	1	0	1	0	0	0	-1	-1	0	-1	-2	-2	-2	-2	-2	-3
T	-1	1	**4**	1	-1	1	0	1	0	0	0	-1	0	-1	-2	-2	-2	-2	-2	-3
P	-3	-1	1	**7**	-1	-2	-1	-1	-1	-1	-2	-2	-1	-2	-3	-3	-2	-4	-3	-4
A	0	1	-1	-1	**4**	0	-1	-2	-1	-1	-2	-1	-1	-1	-1	-1	-2	-2	-2	-3
G	-3	0	1	-2	0	**6**	-2	-1	-2	-2	-2	-2	-2	-3	-4	-4	0	-3	-3	-2
N	-3	1	0	-2	-2	0	**6**	1	0	0	-1	0	0	-2	-3	-3	-3	-3	-2	-4
D	-3	0	1	-1	-2	-1	1	**6**	2	0	-1	-2	-1	-3	-3	-4	-3	-3	-3	-4
E	-4	0	0	-1	-1	-2	0	2	**5**	2	0	0	1	-2	-3	-3	-3	-3	-2	-3
Q	-3	0	0	-1	-1	-2	0	0	2	**5**	0	1	1	0	-3	-2	-2	-3	-1	-2
H	-3	-1	0	-2	-2	-2	1	1	0	0	**8**	0	-1	-2	-3	-3	-2	-1	2	-2
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	**5**	2	-1	-3	-2	-3	-3	-2	-3
K	-3	0	0	-1	-1	-2	0	-1	1	1	-1	2	**5**	-1	-3	-2	-3	-3	-2	-3
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	**5**	1	2	1	0	-1	-1
I	-1	-2	-2	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	**4**	2	3	0	-1	-3
L	-1	-2	-2	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	**4**	1	0	-1	-2
V	-1	-2	-2	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	**4**	-1	-1	-3
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	**6**	3	1
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	**7**	2
W	-2	-3	-3	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	**11**

Figura 5.4: *Matriz de pontuação BLOSUM62. Salvo raras exceções, posições dos quadrados cinza têm pontuação positiva e as demais negativa.*

BLAST

Sejam s e t duas sequências de símbolos. Vamos denominar s por *sequência de referência* e t *sequência secundária*. Sejam w_s e w_t dois k-gramas quaisquer de s e t respetivamente. Seja também w uma palavra qualquer de $\Sigma*$ de tamanho k. A heurística BLAST ([AGM+90]) está dividida em três etapas:

1. **Geração de Pares de Alta Pontuação**: Seja w_s um k-grama da sequência s (existem $|s|-k+1$) e uma palavra w. O par (w_s, w) é pontuado a partir da pontuação do alinhamento natural entre w_s e w coluna a coluna considerando a matriz de substituição. A pontuação é a somatória das pontuações individuais das colunas de w_s e w. Se a pontuação for maior ou igual ao parâmetro T (definido pelo usuário) então o par (w_s, w_Σ) é um Par de Alta Pontuação (PAP) e é armazenado numa lista que será processada na etapa 2. A figura 5.5 mostra um PAP de pontuação 10 formado pelo k-grama w_s ="SLM", w ="SVM". Para $T = 10$ o PAP é considerado para a segunda etapa.

   Esta etapa produz uma lista com todos os PAPs possíveis, fixados k, s e T. Na etapa seguinte essa lista é procurada na sequência secundária para encontrar as *sementes* de um alinhamento local.

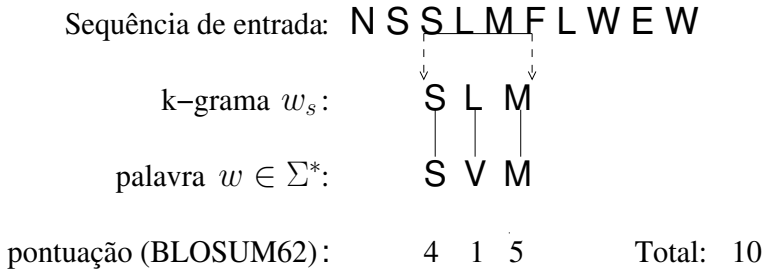


Figura 5.5: *Exemplo da etapa 1 da heurística BLAST com $k = 3$ e $T = 9$. O k-grama w_s ="SLM" da sequência de referência forma um PAP com w ="SVM". O PAP tem pontuação 10 ao se utilizar a matriz BLOSUM62 e será considerado na etapa 2 pois a sua pontuação é maior ou igual a T.*

2. **Localização de sementes** Seja (w_s, w) um PAP da lista produzida na etapa 1 e seja i tal que $w_s = s[i, i+k-1]$. Na etapa 2, o algoritmo tenta encontrar ocorrências j de w na sequência secundária t. Esta busca poder ser feita em tempo constante através de uma tabela de espalhamento de todos os k-gramas de t. O par (i, j) é chamado de semente, pois é semente da expansão para um alinhamento local e é associado ao PAP (w_s, w). Esta etapa conclui com uma lista de sementes que serão expandidas na etapa 3.

   A figura 5.6 mostra a semente associada ao PAP (SLM,SVM) mostrado na figura 5.5.

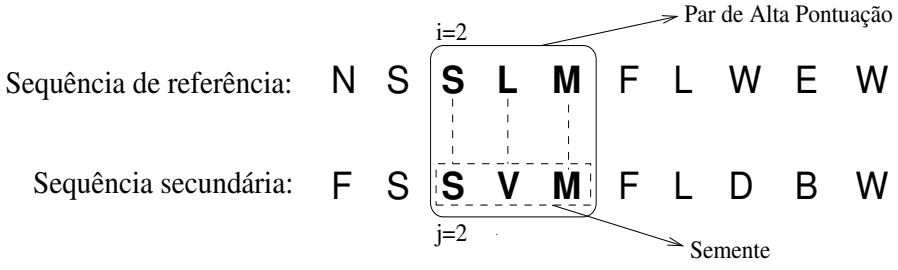


Figura 5.6: *Etapa 2 do BLAST. É mostrada a semente do PAP* (SLM,SVM) *da figura 5.5*

3. **Expansão** Nesta etapa, os alinhamentos locais associados às sementes obtidas na etapa anterior são expandidos em ambas as direções. Para cada semente, parte-se do alinhamento local formado pelo próprio PAP a ela associado. Fixada uma direção, a cada passo da expansão é avaliado o acréscimo ao alinhamento de uma nova coluna formada por um símbolo de cada sequência. Sua pontuação acumulada é

atualizada, bem como a máxima pontuação de um dos alinhamentos locais. A iteração deste processo termina quando se atinge a extremidade de uma das sequências ou quando a diferença da pontuação acumulada para a máxima pontuação exceder a *queda máxima* permitida X. Os índices associados aos dois símbolos que compõem a coluna acrescida que tiver produzido o alinhamento local de máxima pontuação definem um par de coordenadas que vai delimitar na respectiva direção o alinhamento local procurado.

A composição das expansões em ambas as direções resulta no alinhamento de um *Par de Segmentos de Pontuação Maximal (PSPM), sujeito à máxima queda* X.

Como mostra a figura 5.7, a expansão da semente do PAP (SLM,SVM) da figura 5.6 com $X = 3$ obtém o PSPM (SSLMFL,SSVMFL). Sua expansão à esquerda (índices decrescentes) termina porque as sequências terminam, resultando no par de coordenadas $(1,1)$. Sua expansão à direita termina porque a queda da pontuação acumulada por conta do acréscimo da coluna que alinha W a D excede a queda máxima 3, resultando no par de coordenadas $(6,6)$. Observe que este par de coordenadas seria $(9,9)$ se a queda máxima fosse 4 ou mais. O PSPM obtido a partir da semente em questão possui pontuação 24 e coordenadas $(1,1)$ e $(6,6)$.

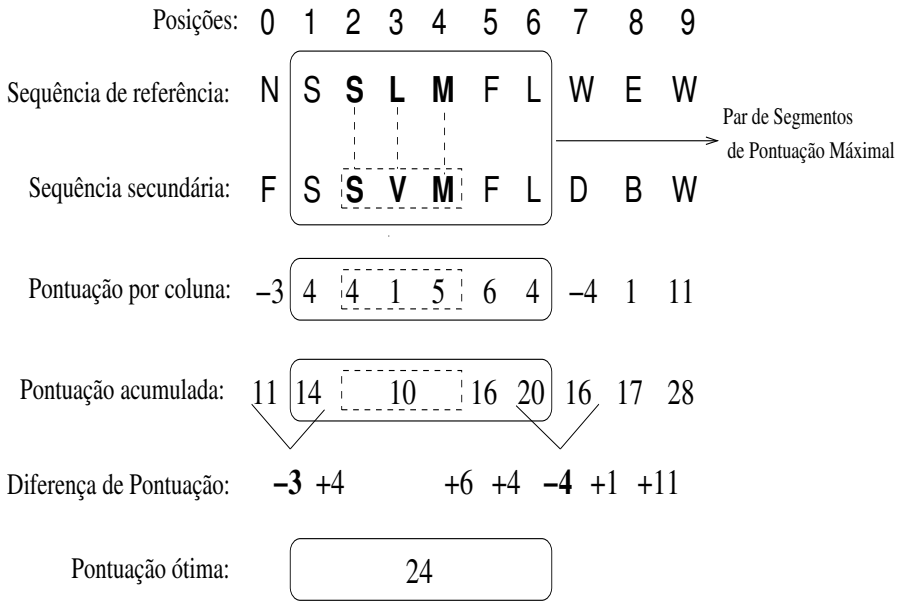


Figura 5.7: *Etapa 3 do BLAST. O Par de Segmentos de Pontuação Maximal (PSPM) sujeito à queda máxima* 3 *obtido a partir da semente da figura 5.6 é mostrado acima e possui pontuação* 24 *e coordenadas* $(1,1)$ *e* $(6,6)$. *Já o PSPM sujeito à queda máxima* 4 *tem pontuação* 32 *e coordenadas* $(1,1)$ *e* $(9,9)$

O BLAST não leva em conta a inserção de espaços nos alinhamentos locais associados aos PSPMs. Considerar essa possibilidade leva à implementação de um algoritmo de programação dinâmica que possui uma complexidade computacional maior.

O BLAST com espaços encontra alinhamentos locais executando a solução de programação dinâmica. Essa execução é controlada inserindo um parâmetro adicional que controla o tamanho da faixa de busca.

BLAST com espaços

Como foi dito anteriormente, o BLAST não encontra alinhamentos locais com espaços. No entanto, anos depois ele foi melhorado para considerar essa possibilidade ([AMS+97]).

Essa melhoria possui as seguintes heurísticas adicionais:

- No BLAST com espaços exige-se duas restrições para quaisquer dois PAPs que pertençam à mesma diagonal: eles não podem ter sobreposição e a distância entre eles deve ser pelo menos A que é um parâmetro fornecido pelo usuário.
- Costumeiramente, o BLAST ordena os PSPMs obtidos segundo suas respectivas pontuações e lista os k primeiros.
- A partir de um PAP é possível gerar um alinhamento local com espaços. Esse alinhamento é produzido utilizando um algoritmo de programação dinâmica que é restrito a uma faixa do espaço de busca das duas sequências s e t. O parâmetro X controla o tamanho da faixa de busca.

  Este novo tipo de extensão é realizado ainda em ambas as direções da semente e possui um efeito na escolha de T: O valor deste parâmetro pode ser incrementado acelerando assim a computação dos alinhamentos locais. Embora o valor de T é maior, os outros PAPs são encontrados porque agora são admitidos espaços a partir de um PAP.

A figura 5.8 mostra a execução do BLAST com espaços para duas sequências biológicas.

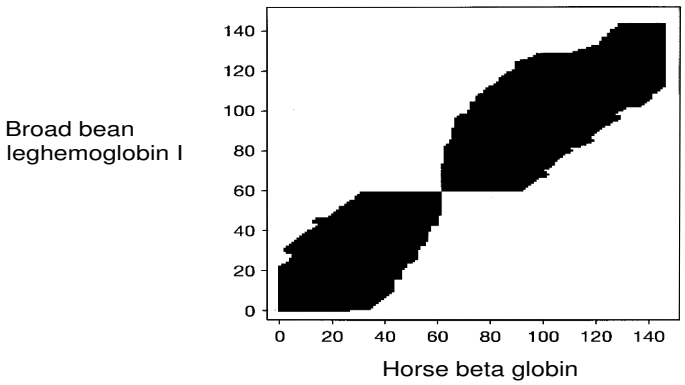


Figura 5.8: *Resultado da execução do BLAST com espaços em duas sequências biológicas. A figura mostra as faixas da matriz de soluções de programação dinâmica onde foi realizada a busca. Adaptado do original em [AMS+97].*

Nesta seção vimos como são obtidos os alinhamentos globais e locais para sequências biológicas. Como a complexidade computacional é quadrática, inviabiliza como solução para vários tipos de entradas comuns em Bioinformática. A heurística BLAST foi proposta para minimizar o problema reduzindo a complexidade.

Para alinhar pares de documentos de texto podemos reutilizar as ideias do BLAST apresentadas até agora. Para tanto, é necessário definir a similaridade entre pares de documentos e resolver alguns problemas como a geração dos PAP e das sementes.

Na continuação apresentamos o BLASTD, uma heurística que estende as ideias do BLAST para encontrar alinhamentos locais com espaços para pares de documentos.

5.2 Alinhamento de pares de documentos de texto

O alinhamento de documentos de texto não é um problema recente, ele tem sido abordado em diferentes contextos. Em Processamento de Linguagem Natural documentos de

texto são alinhados para construir tradutores ([MS99]) ou para encontrar segmentos de texto compartilhados [Gle15, OHR].
Na continuação fazemos uma breve revisão do problema do alinhamento local de pares de documentos de texto.

5.2.1 Matriz de substituição ternária

No caso de alinhamento de documentos, em vez de uma matriz de substituição de aminoácidos como a BLOSUM62 usada para se alinhar proteínas, deve-se usar uma matriz de substituição de termos. O desenvolvimento de uma matriz de substituição como a BLOSUM62, requer a existência de dados de treinamento formados por um bom conjunto de alinhamentos de referência. Pares de aminoácidos distintos que possuem uma pontuação positiva tipicamente pertencem a um mesmo grupo funcional e substituições envolvendo estes pares são observadas mais frequentemente nestes alinhamentos. Matrizes de substituições de termos como estas já existem à disposição de grandes companhias que têm feito fortuna através de máquinas de busca e são usadas no aprimoramento de seus algoritmos através de expansão de consulta. Contudo, informação deste tipo é tratada como segredo industrial e não é pública.

Uma possível alternativa é aquela adotada no alinhamento de DNA visto na seção 5.1. Na equação 5.1, a pontuação $\delta(s[i] \rightarrow t[j])$ para se alinhar numa mesma coluna os símbolos $s[i]$ e $t[j]$ foi adotada como sendo $+1$ se $s[i] = t[j]$ e -1 se $s[i] \neq t[j]$. Isto equivale a uma matriz de substituição com $+1$ na diagonal principal e -1 nas demais posições, valores estes que são arbitrários e podem ser parametrizados. Assim sendo, este esquema generaliza para um *sistema de pontuação básico* para a substituição de símbolos, de forma que a

identidade de símbolos é recompensada com a mesma pontuação positiva e a

substituição de um símbolo por outro é penalizada com a mesma pontuação negativa.

Na referida seção, o alfabeto Σ é o conjunto de quatro nucleotídeos presentes numa cadeia de DNA, mas o mesmo sistema de pontuação básico pode ser aplicado a um caso em que o alfabeto Σ é o conjunto de termos usados num conjunto de documentos. Numa tal aplicação, pares de termos idênticos numa coluna de um alinhamento recebem a mesma pontuação enquanto que pares de termos distintos recebem uniformemente um outro valor, não importando se os termos são muito diferentes ou se apenas uma letra diferencia um do outro. Pode-se refinar a granularidade de uma tal classificação binária, introduzindo uma terceira pontuação para um par de termos diferentes mas similares. Assim, dados dois símbolos x e y, definimos as seguintes relações entre x e y:

Igualdade – quando os símbolos x e y são idênticos;

Similaridade – quando x e y não são idênticos mas similares segundo algum critério;

Disparidade – quando x e y não são nem idênticos nem similares.

Isto evidentemente permite definir um *sistema de pontuação ternário* para a substituição de símbolos, com sua correspondente matriz de substituição de símbolos, ambos parametrizados por três valores inteiros: a pontuação t_{igual} de um par de símbolos iguais;

a pontuação $t_{similar}$ de um par de símbolos similares; e a pontuação t_{dispar} de um par de símbolos díspares. Assim temos que:

$$\delta(x \to y) = \begin{cases} t_{igual}, \text{ se } x = y; \\ t_{similar}, \text{ se } x \text{ e } y \text{ são similares;} \\ t_{dispar}, \text{ se } x \text{ e } y \text{ são díspares.} \end{cases} \tag{5.3}$$

Convém observar que estes parâmetros não devem possuir valores inteiros quaisquer. Isto porque uma coluna com símbolos díspares deve ser pontuada negativamente e sua pontuação deve ser menor que a do caso em que os símbolos são similares; enquanto que uma coluna com símbolos iguais deve ser pontuada positivamente, e sua pontuação deve ser superior à do caso em que os símbolos são similares:

$$t_{igual} > t_{similar} > t_{dispar} \qquad \text{e} \qquad t_{igual} > 0 > t_{dispar}.$$

Parece-nos mais adequado que símbolos similares sejam pontuados positivamente, mas a influência da adoção de uma pontuação não positiva nos alinhamentos de documentos vale ser estudada. Além da própria escolha dos três parâmetros da pontuação acima mencionados, é parte crítica de um sistema de pontuação ternário a definição do próprio *critério de similaridade*, a respeito do qual passamos a discutir no caso de nossa aplicação de interesse.

5.2.2 Critério de similaridade de termos

Uma motivação para a introdução de uma terceira classe de classificação para pares de termos são os erros frequentemente em documentos produzidos por um Reconhecedor Ótico de Caracteres (OCR). Isto se deve aos erros de classificação das imagens e, erros ainda maiores no caso de imagens de textos antigos, onde a tipografia é muito distinta da tipografia atual provavelmente usada nas imagens de treinamento do OCR e os variados problemas de conservação dos escritos introduzem manchas ou falhas na tinta nos próprios escritos. Seja pois a *taxa de erro* β um real tal que $0 \leq \beta < 1$. A taxa de erro, um parâmetro a ser definido pelo usuário, determina o valor máximo admissível para a distância de edição entre palavras consideradas similares. Desta forma, dadas a distância de edição $d(x, y)$ vista na seção 4.1 e a taxa de erro β, definimos um *primeiro critério de similaridade* segundo o qual duas palavras

$$x \text{ e } y \text{ são similares quando } d(x, y) \leq \beta \cdot max\{|x|, |y|\}, \text{ e } x \neq y.$$

A adoção deste primeiro critério de similaridade permite definir uma matriz de substituição parametrizada por quatro valores: a taxa de erro; a pontuação de um par de termos iguais; a pontuação de um par de termos similares; e a pontuação de um par de termos díspares.

Além dos erros ortográficos introduzidos por um sistema OCR ou pelo próprio homem, razões de natureza semântica ou sintática motivam eventuais alterações no critério de similaridade. Por exemplo, o autor de uma nova edição ou um copista de um documento antigo podem eventualmente decidir pela correção de um erro de concordância presente no texto original, sem que isto introduza qualquer alteração semântica. Tal como o estudo linguístico nos ensina, para uma palavra w, seu *radical* $\rho(w)$ representa o *núcleo semântico* da palavra w. A função inversa ρ^{-1} mapeia o radical $\rho(w)$ no conjunto $\Phi(w)$ formado por todas as palavras que possuem este mesmo radical.

A este conjunto $\Phi(w)$ chamamos de *núcleo* de w e ele forma uma classe de equivalência. Assim,

$$\Phi(w) = \rho \circ \rho^{-1}(w) = \{w' \mid w \text{ e } x' \text{ possuem o mesmo radical}\}.$$

Temos então um *segundo critério de similaridade*, segundo o qual palavras

x e y são similares quando x e y possuem o mesmo radical, e $x \neq y$.

Termos diferentes com o mesmo radical são similares por este segundo critério. No caso de uma língua rica em declinações como é o caso do Latim, isto é particularmente interessante e tal critério de similaridade não depende de outro parâmetro senão a própria língua em que o texto foi escrito, podendo ainda ser estendido com a incorporação de um dicionário de sinônimos ou *thesaurus*, bem como algum dicionário de variações ortográficas.

Convém observar que os dois critérios de similaridade vistos acima não são excludentes mas complementares, como se pode verificar no exemplo apresentado na figura 5.9, que analisa parte do núcleo do termo `venit` construído a partir de dados coletados do projeto *Index Thomisticus Treebank* [MMPR]. O par das declinações verbais `venientem` e `venissemus` possui distância de edição 7, que é 70% do comprimento de `venissemus`. Apesar de distantes, as duas palavras são similares pelo segundo critério pois possuem o mesmo radical `venio`. Por outro lado, a substituição de um simples e por c, um erro típico de OCR, transforma `venissemus` numa palavra fora do vocabulário latino: `venisscmus`. Assim, este segundo par de palavras não é um par de palavras similares pelo segundo critério pois `venisscmus` não possui radical conhecido, apesar de ter uma distância de edição de apenas 10% em relação a `venissemus`.

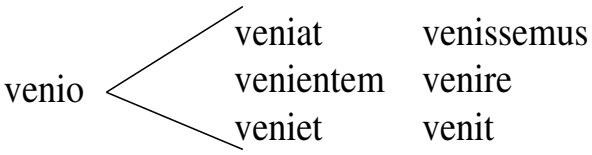


Figura 5.9: *Parte do núcleo de* `venit`*, cujo radical* `venio` *corresponde ao verbo* `vir` *em português. A figura exibe algumas das declinações verbais deste verbo de quarta conjugação. Para qualquer β entre* 10% *e* 69%, `venissemus` *e* `venientem` *são díspares pelo primeiro critério de similaridade e similares pelo segundo, ao passo que* `venissemus` *e* `venisscmus` *são similares pelo primeiro critério e díspares pelo segundo.*

Por um lado, a adoção de uma taxa de erro não inferior a 10% torna as palavras deste último par similares pelo primeiro critério, apesar de serem díspares pelo segundo; por outro lado, qualquer taxa de erro inferior a 70% torna as palavras deste último par díspares pelo primeiro critério, apesar de serem similares pelo segundo. Já a adoção de uma taxa de erro de 70% torna `venissemus` similar tanto a `venisscmus` quanto a `venientem`, mas também a palavras tão diversas quanto `verbum`, `adesse`, `electus`, `permissis`, `iniurias` e `veritatis`.

À simples troca do termo `venivi` para `veniebam` – uma simples mudança de tempo verbal de pretérito perfeito para pretérito imperfeito, onde se mantém a conjugação de `venio` na primeira pessoa do singular – temos associada uma distância de edição 6, que é 75% do comprimento de `veniebam`.

Demonstrado que os dois critérios são complementares, vem naturalmente a pergunta sobre a possibilidade e a conveniência de compô-los. Uma maneira bastante imediata de

fazê-lo se dá pela adoção de um *terceiro critério de similaridade*, onde se definem como similares as palavras que são similares por algum dos dois primeiros critérios. Dependendo dos dados a serem usados pela aplicação, porém, uma composição mais elaborada pode ser mais conveniente. Caso se queira comparar dois textos obtidos de forma automática por um sistema OCR que tenha sido aplicado às imagens de dois escritos antigos, por exemplo, parece-nos mais adequada uma modelagem onde se adota um *quarto critério de similaridade*, onde as palavras

> x e y são *similares* quando $x \neq y$ e:
>
> - existe palavra x' não díspar de x pelo primeiro critério;
> - existe palavra y' não díspar de y pelo primeiro critério;
> - e x' e y' não são díspares pelo segundo critério.

Se esta modelagem do problema nos parece mais realista, ela também apresenta um desafio computacional que nos parece bastante interessante: aquele de implementar de forma eficiente este critério de similaridade. Por restrições de tempo e recursos, este quarto critério não veio a ser implementado na corrente versão de nosso sistema. Ademais, também a implementação corrente do segundo critério precisa de melhorias, principalmente pela dificuldade de se conseguir um vocabulário completo da língua latina usada nos referidos escritos.

5.2.3 Similaridade de documentos

Sejam R e S dois documentos definidos de acordo com a seção 4.2 e que possuem tamanhos $m = |R|$ e $n = |S|$, respectivamente. Denominamos R o *documento de referência* e S o *documento secundário*. A similaridade $\text{sim}(R, S)$ entre os documentos R e S pode ser calculada através da equação 5.1. Naquele exemplo, pontua-se uma coluna que alinha os símbolos $s[i]$ e $t[j]$ por $+1$ se $s[i] = t[j]$ e -1 se $s[i] \neq t[j]$. Como visto acima, este sistema de pontuação básico equivale a uma matriz de substituição com $+1$ na diagonal principal e -1 nas demais posições, mas generaliza pela equação 5.3 para uma matriz de substituição $\delta(x \rightarrow y)$ definida por um sistema de pontuação ternário, que pressupõe a adoção de um critério de similaridade e que depende da adoção de parâmetros $t_{igual}, t_{similar}$ e t_{dispar}. Ademais, as pontuações da inserção e da remoção adotadas no exemplo são iguais a -2. Em nosso caso, a pontuação de inserção e remoção é dada pelo parâmetro t_ϵ, negativo, onde ϵ representa o documento vazio. Assim, t_ϵ, $t_{igual}, t_{similar}$ e t_{dispar} são parâmetros definidos pelo usuário e permitem reescrever a equação 5.1 como segue:

$$\text{sim}(R[1,i], S[1,j]) = max \begin{cases} \text{sim}(R[1,i], S[1,j-1]) + t_\epsilon \\ \text{sim}(R[1,i-1], S[1,j-1]) + \delta(R[i] \rightarrow S[j]) \\ \text{sim}(R[1,i-1], S[1,j]) + t_\epsilon \end{cases} \tag{5.4}$$

Esta similaridade $\text{sim}(R, S) = \text{sim}(R[1,m], S[1,n])$ é a máxima pontuação de um alinhamento global entre os documentos R e S, obtida como soma das pontuações das colunas de um alinhamento ótimo, cada uma delas associada à substituição de um possível termo presente na posição superior da coluna do alinhamento pelo possível termo presente na posição inferior.

Neste trabalho desejamos alinhar trechos de documentos que não necessariamente possuem o mesmo tamanho. Assim, é do nosso interesse encontrar segmentos de dois

documentos que sejam similares entre si. Em outras palavras, desejamos encontrar alinhamentos locais de grande similaridade entre dois documentos.

Da mesma forma que no algoritmo de Smith-Waterman visto na seção 5.1.2, podemos armazenar a computação da maior pontuação de um alinhamento entre um sufixo de $R[1,i]$ e um sufixo de $S[1,j]$ na posição $M[i,j]$ de uma matriz M de tamanho $(m+1)\times(n+1)$. A célula $M[i,j]$ da matriz é calculada pela seguinte fórmula de recorrência:

$$M[i,0] = i \tag{5.5}$$

$$M[0,j] = j \tag{5.6}$$

$$M[i,j] = max\begin{cases} M[i,j-1] + t_\epsilon \\ M[i-1,j-1] + \delta(R[i] \rightarrow S[j]) \\ M[i-1,j] + t_\epsilon \\ 0 \end{cases}, \qquad \text{se } i > 0 \text{ e } j > 0 \tag{5.7}$$

Cada $M[i,j]$ computado através da fórmula 5.7 ainda requer a computação da distância de edição entre os termos $R[i]$ e $S[j]$ e são $m \times n$ as posições da matriz M computadas através desta fórmula. Ainda que os tamanhos dos documentos por nós analisados não inviabilizem a execução do algoritmo de Smith-Waterman para a busca de um alinhamento local de máxima pontuação, optamos por uma solução mais eficiente e mais escalável oferecida por uma heurística como a utilizada pelo BLAST visto que também nos interessa a obtenção de alinhamentos locais sub-ótimos, que requereriam modificações não triviais no algoritmo de Smith-Waterman e que produziriam uma piora significativa na complexidade computacional do algoritmo. Nas próximas subseções exibimos algumas modificações executadas nas etapas do BLAST para a busca dos alinhamentos locais de documentos com mais alta pontuação.

5.2.4 BLASTD: Uma heurística tipo BLAST que alinha documentos

Esta subseção descreve os problemas e as soluções encontradas para uma adaptação das etapas do BLAST com espaços para encontrar alinhamentos locais em pares de documentos. Além de adaptar o conceito de Pares de Alta Pontuação (PAPs) para as sequências de termos, ela descreve os métodos eficientes adotados para encontrar sementes de palavras nos documentos a partir dos PAPs. Finalmente, é apresentado o conceito de Pares de Multi-Segmentos (PMS) para nos referir ao resultado da expansão das sementes.

Etapa 1: Geração da lista de Pares de Alta Pontuação

No BLAST para sequências biológicas, os PAPs (w_s, w) são gerados a partir dos k-gramas w_s da sequência de referência s e de palavras w quaisquer de tamanho k que alinhem com alta pontuação, ao menos T. O número das possíveis palavras de tamanho k é $|\Sigma|^k$, ou seja: 20^k, no caso de proteínas; e 4^k, no caso de DNA. No caso de sequências de termos, o tamanho do alfabeto é ao menos tão grande quanto o tamanho do dicionário (de termos), de forma que se torna inviável utilizar a mesma técnica para gerar a lista de PAPs na primeira etapa do BLAST. Para contornar este problema, introduzimos uma simplificação no problema. Sejam K_R e K_S são os conjuntos de k-gramas de R e de S, respectivamente. Em nossa simplificação, limitamos as possíveis escolhas dos PAPs a

$\Pi = \{(w_R, w_S) \in K_R \times K_S \mid w_R = w_S$ ou w_R e w_S diferem em exatamente uma posição e os termos envolvidos são similares$\}$.

Lembremos que a pontuação para um par de k-gramas de documentos é análoga à pontuação de k-gramas definida para o BLAST: cada par de termos que pertence a uma coluna do alinhamento canônico do par de k-gramas é pontuada utilizando a mesma matriz de substituição de termos adotada na equação 5.4 e a pontuação total do par de k-gramas é a soma das pontuações das colunas. Assim, a pontuação de um PAP $(w_R, w_S) \in \Pi$ pode assumir apenas um dos valores:

- kt_{igual}, no caso em que $w_R = w_S$;
- $(k-1)t_{igual} + t_{similar}$, no caso em que w_R e w_S diferem em exatamente uma posição e os termos envolvidos são similares.

Além disso, qualquer par $(w_R, w_S) \in K_R \times K_S \setminus \Pi$ possuirá pontuação não superior a

$$\max((k-2)t_{igual} + 2t_{similar}, (k-1)t_{igual} + t_{dispar}) < (k-1)t_{igual} + t_{similar} < kt_{igual}.$$

Donde, podemos nos limitar a dois valores valores de T:

$$T = \begin{cases} kt_{igual}, \text{ ou} \\ (k-1)t_{igual} + t_{similar}, \end{cases} \tag{5.8}$$

e a lista de PAPs a ser devolvida é:

$$Q = \begin{cases} \{(w, w) \mid w \in K_R \cap K_S\}, \text{ se } T = kt_{igual}; \\ \Pi, \text{ se } T = (k-1)t_{igual} + t_{similar}. \end{cases} \tag{5.9}$$

Gerar $\{(w, w) \mid w \in K_R \cap K_S\}$ é fácil. A seguir exibimos dois algoritmos para gerar Π. O primeiro leva em conta o primeiro critério de similaridade visto na subseção 5.2.2 e usa as técnicas de busca aproximada definidas na seção 4.3. O segundo aplica-se ao segundo critério de similaridade visto na mesma seção.

1. Geração de PAPs por distância de edição Nesta etapa utilizamos as técnicas de Busca Aproximada definidas na seção 4.3 para gerar os k-gramas w. O algoritmo 9, mostra como são gerados esses k-gramas utilizando o filtro $FastFilter$.

Entre as linhas 1-4 os termos do dicionário Γ_R são iterados para obter os termos similares utilizando o filtro $FastFilter$ com base no dicionário Γ_S. Esses termos são armazenados na tabela de espalhamento H que é consultada depois para gerar os k-gramas w. A execução do $FastFilter$ demanda, além dos dicionários Γ_R e Γ_S, as tabelas de espalhamento dos sacos de símbolos η_R e η_S para ambos os documentos e o índice de k-gramas Ψ_S do documento secundário.

Em posse dos termos similares para os termos de Γ_S o algoritmo passa a gerar novos k-gramas w para cada k-grama w_R do documento de referência (linhas 6-15). O conjunto Q coleta os PAPs (w_R, w) e é devolvido no fim do algoritmo.

Observe que podemos armazenar a tabela de espalhamento H em um sistema de cache para assim evitar a sua computação de forma repetitiva. Essa tabela depende

Algoritmo 9 Geração de pares por distância de edição(*ApproxExpand*)

Entrada:
K_R, K_S índices de k-gramas dos documentos R e S
Γ_R, Γ_S dicionários de termos dos documentos de referência e secundário
η_R, η_S índice de sacos de palavras dos dicionários Γ_R e Γ_S
Ψ_S índice de k-gramas dos termos do dicionário Γ_S
β taxa de erro
Saída:
lista de PAPs

```
1  Inicializar tabela de espalhamento H
2  para todo t ∈ Γ_R faça
3     H[t] ← FastFilter(t, η_R(t), Γ_S, Ψ_S, η_S, β)
4  fim
5  Q ← ∅
6  para todo w_R ∈ K_R faça
7     para todo x ∈ w_R faça
8        para todo x' ∈ H(x) faça
9           Substitua x por x' no k-grama w_R e gere w
10          se w ∈ K_S então
11             Q ← Q ∪ {(w_R, w)}
12          fim
13       fim
14    fim
15 fim
16 devolva Q
```

tam	pestilentes	homines	interficere
tam	pestilcntes	homines	interficere

Figura 5.10: *Geração de PAPs por distância de edição para o k-grama* `tam pestilentes homines interficere`. *Ao procurar termos próximos ($\beta = 30$) a* `pestilentes`, *o algoritmo 9 encontrou* `pestilcntes`, *que deve conter um erro de OCR. O k-grama da linha superior pertence à obra "B.P. Francisci Assisiatis Opuscula", escrita por Luca Wadding em 1623; e o da linha inferior foi extraído de uma digitalização da obra "De Conformitate", escrita em torno do ano 1380 por Bartolomeu de Pisa.*

unicamente da taxa de erro β pois os dicionários não possuem uma variabilidade alta.

2. Geração de PAPs por radicais A ideia é gerar k-gramas w com termos similares obtidos a partir de um dicionário de radicais. O termo similar é verificado no dicionário Γ_S e o novo k-grama w é verificado em K_S.

Algoritmo 10 Geração PAP por radicais(*LemmaExpand*)

Entrada:
K_R, K_S índices de k-gramas de R e de S
Γ_S dicionário dos termos de S
Φ o núcleo Φ é $\rho \circ \rho^{-1}$, dado o dicionário de radicais ρ
Saída:
lista de PAPs

```
1  Q ← ∅
2  para todo w_R ∈ K_R faça
3    para todo x ∈ w_R faça
4      para todo x' ∈ Φ(x) ∩ Γ_S faça
5        Substitua x por x' no k-grama w_R e gere w
6        se w ∈ K_S então
7          Q ← Q ∪ {(w_R, w)}
8        fim
9      fim
10   fim
11 fim
12 devolva Q
```

A figura 5.11 mostra um exemplo da geração de PAP por expansão de radicais.

venit	ad	me	et
veniat	ad	me	et

Figura 5.11: *Geração de PAPs por radicais no BLASTD para o k-grama* `venit ad me et`. *O algoritmo LemmaExpand gerou o PAP da figura porque encontrou no núcleo* Φ(`venit`) *o termo* `veniat`. *O k-grama da linha superior pertence ao Evangelho segundo Lucas e o inferior ao Evangelho segundo João, dois livros presentes na Vulgata Clementina. O dicionário de radicais foi coletado no projeto Index Thomisticus Treebank.*

A figura 5.12 mostra a pontuação por coluna para os pares das figuras 5.10 e 5.11.

Etapa 2: Localização de sementes

Nesta etapa os PAPs gerados na etapa 1 são localizados no documento secundário S. Considere o dicionário de termos Γ_S de tamanho $n_\Gamma = |\Gamma_S|$. Atribuindo a cada palavra de Γ_S um inteiro no intervalo $[1, n_\Gamma]$, pode-se facilmente mapear cada k-grama w_S num identificador único, por exemplo através de uma função de espalhamento (perfeita ou não),

tam	pestilentes	homines	interficere
tam	pestilcntes	homines	interficere
2	1	2	2

(a)

venit	ad	me	et
veniat	ad	me	et
1	2	2	2

(b)

Figura 5.12: *Pontuação dos pares mostrados nas figuras 5.10 e 5.11, para* $t_{igual} = 2$ *e* $t_{similar} = 1$. *A linha inferior mostra a pontuação coluna a coluna, totalizando 7 em cada subfigura.*

e construímos uma tabela de espalhamento onde este identificador único é associado à lista das posições onde w_S ocorre em S. Essa estrutura constitui o *índice de k-gramas de S* e pode ser pré-computada e armazenada em disco para $k \in \{2, 3, 4\}$.
Uma vez obtida essa estrutura, cada PAP (w_R, w) obtido na etapa 1 produz uma *semente* para cada ocorrência do k-grama w em S, já que a semente possui uma localização determinada pelas posições de w_R e w nos documentos R e S, respectivamente.
A figura 5.13 mostra as sementes para os PAPs mostrados na etapa 1.

venit 12594	ad 12595	me 12596	et 12597
veniat 5641	ad 5642	me 5643	et 5644
1	2	2	2

(a)

tam 215	pestilentes 216	homines 217	interficere 218
tam 783307	pestilcntes 783308	homines 783309	interficere 783310
2	1	2	2

(b)

Figura 5.13: *Sementes para os PAPs das figuras 5.10 e 5.11. Suas coordenadas iniciais são respectivamente* $(215, 783307)$ *e* $(12594, 5641)$.

Etapa 3: Expansão de sementes

Nesta etapa as sementes são expandidas para gerar Pares de Multi-Segmentos (PMS), uma forma de representar um alinhamento com espaços, tal como se pode ver na figura 5.14. *Um multi-segmento de um documento é formado por: uma sequência de segmentos posicionados e contíguos, ordenados de acordo com suas posições; uma sequência de inserções de espaços, ordenada pelos pontos de inserção: cada inserção é feita antes do primeiro segmento*

ou depois de cada um dos segmentos posicionados.

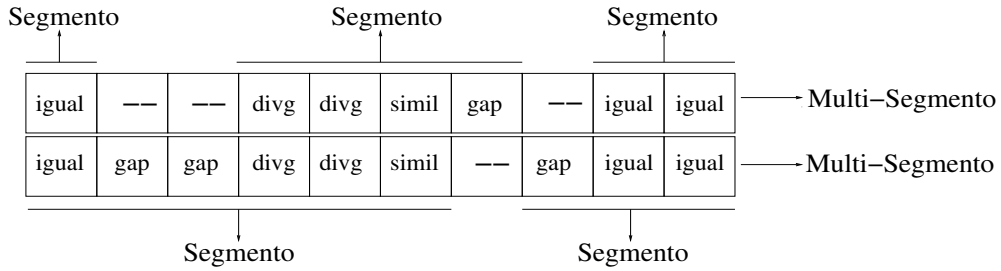


Figura 5.14: *Par de Multi-Segmentos (PMS). O multi-segmento superior contém: três segmentos contíguos; dois espaços inseridos entre estes os dois primeiros e um entre os dois últimos. O multi-segmento inferior contém: dois segmentos contíguos; um espaço que os separa. Em cada coluna sem espaço são mostradas as relações entre os termos da coluna, sendo que um termo inserido ou removido é anotado com "gap".*

Seja h um multi-segmento de um documento d. Pelo fato dos segmentos posicionados serem contíguos, sua concatenação, respeitada a ordem, forma um segmento no documento com *início* $s(h)$ *na primeira posição do primeiro segmento* e *fim* $e(h)$ *na última posição do último*. Esta concatenação $d[s(h), e(h)]$ é o *segmento original do multi-segmento* h. Se além disso forem concatenados os espaços nas devidas posições, obtemos uma sequência num alfabeto estendido por este espaço e que forma a *resultante do multi-segmento*, cujo comprimento é o *comprimento* $|h|$ *do multi-segmento* h, isto é, a soma dos comprimentos dos segmentos constituintes com a quantidade de espaços inseridos.

Assim, definimos um *Par de Multi-Segmentos (PMS)* como sendo um *par ordenado de multi-segmentos de mesmo comprimento* e o *alinhamento do PMS* em questão é a matriz de formada pelo par ordenado das resultantes dos respectivos multi-segmentos. Observe que este alinhamento pode conter uma coluna com dois espaços. Por mais comprido que seja, um segmento posicionado de um documento pode ser representado por dois inteiros apenas: seu início e seu comprimento, de forma que um alinhamento local pode ser representado de forma muito compacta por um PMS. Ademais, os PMSs permitem estender o conceito de Par de Segmentos de Pontuação Maximal (PSPM) do BLAST original para o de *Par de Multi-Segmentos de Pontuação Maximal (PMSPM)* usado no BLAST com espaços para sequências biológicas, de forma que os PMSPM são encontrados a partir das sementes localizadas na etapa 2.

Na figura 5.15, vemos um alinhamento de dois segmentos posicionados de dois documentos, com intervalos das posições nos segmentos originais $[203, 222]$ e $[783294, 783313]$, respectivamente. Cada PMS possui dois segmentos posicionados separados por um espaço e o comprimento deles é 2). O primeiro PMS contém os segmentos dados pelos intervalos de posições $[203, 213]$ e $[214, 222$, e o segundo PMS contém os segmentos dados pelos intervalos $[783294, 783311]$ e $[783312, 783313]$.

Antes da busca dos PMSPM a partir das sementes localizadas na etapa 2, estas passam por um processo de filtragem que remove as sementes redundantes numa mesma diagonal e que produzem o mesmo alinhamento quando expandidas. Então, todas as sementes de uma mesma diagonal são ordenadas de acordo com sua posição e sementes de uma mesma diagonal tais que seus k-gramas posicionados se sobrepõem são consideradas conexas. Cada componente conexa é agrupada e uma semente central de cada grupo é escolhida para ser expandida. As demais sementes do grupo são descartadas. A figura 5.16 mostra como funciona esse passo.

A expansão das sementes centrais segue os mesmos moldes do BLAST com espaços. A

nam 203	persequentes 204	eos 205	a 206	malignis 207	spiritibus 208	agitati 209	magnum 210	esse 211	obsequium 212	dei 213	--	dicent 214
nam 783294	perseriuentes 783295	eos 783296	a 783297	malignis 783298	spiritibus 783299	agitati 783300	magnum 783301	esse 783302	obsequium 783303	1 783304	1'0 783305	dicent 783306
2	1	2	2	2	2	2	2	2	2	-1	-3	2

tam 215	pestilentes 216	homines 217	interficere 218	et 219	delere 220	de 221	terra 222
tam 783307	pestilcntes 783308	homines 783309	interficere 783310	et 783311	--	de 783312	terra 783313
2	1	2	2	2	-3	2	2

Figura 5.15: *PMS encontrado a partir da expansão da semente da figura 5.13a ($k = 4$, $t_{igual} = 2$, $t_{similar} = 1$, $t_{dispar} = -1$,$t_{\bar{e}} = -3$, $X = 10$). Observe que foram inseridos espaços em ambas as direções da expansão.*

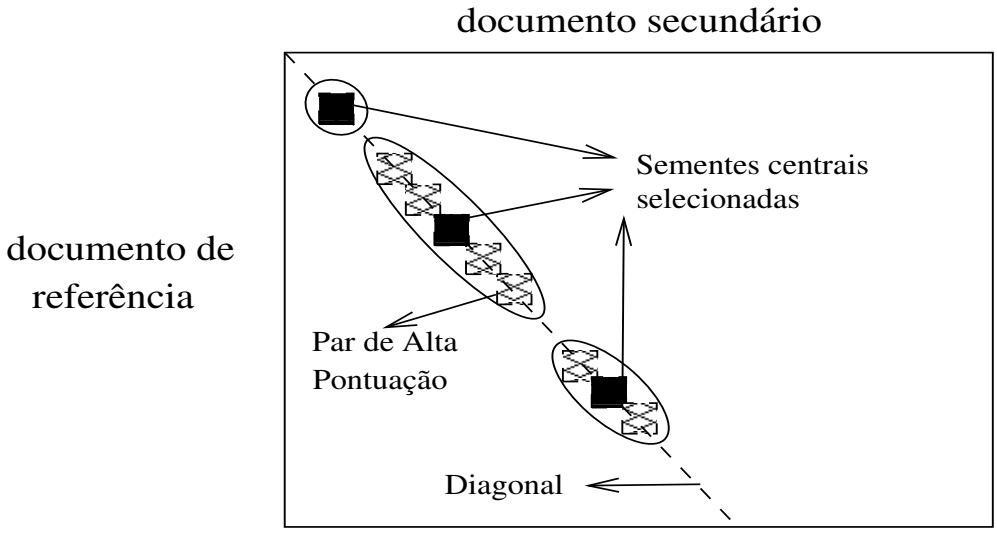


Figura 5.16: *Agrupamento de sementes por diagonal. Sementes que se sobrepõem na mesma diagonal conectadas e cada componente conexa tem uma única semente central escolhida para ser expandida.*

solução de programação dinâmica é aplicada na direção esquerda e direita. O parâmetro X controla a queda máxima da pontuação maximal em cada expansão. As figuras 5.15 e 5.17 mostram exemplos das expansões das sementes da figura 5.13.

O resultado da expansão em ambos os sentidos é um alinhamento local de pontuação maximal, naturalmente representado por um PMSPM.

si 12592	quis 12593	--	venit 12594
si 5638	quis 5639	sitit 5640	veniat 5641
2	2	-3	1

ad 12595	me 12596	et 12597
ad 5642	me 5643	et 5644
2	2	2

Figura 5.17: *PMS encontrado a partir da expansão da semente da figura 5.13b ($k = 4$, $t_{igual} = 2$, $t_{similar} = 1$, $t_{dispar} = -1$, $t_{\bar{\epsilon}} = -3$, $X = 10$). A expansão direita não foi adiante porque a pontuação acumulada foi negativa e ultrapassou o limiar X.*

As figuras 5.18 e 5.19 representam as regiões computadas das matrizes de programação dinâmica dos PMSPMs das figuras 5.15 e 5.17, respectivamente. Os retângulos mostrados nessas figuras representam os PMSPMs obtidos. Além disso, as figuras exibem as regiões das matrizes com as expansões ulteriores às coordenadas das soluções maximais.

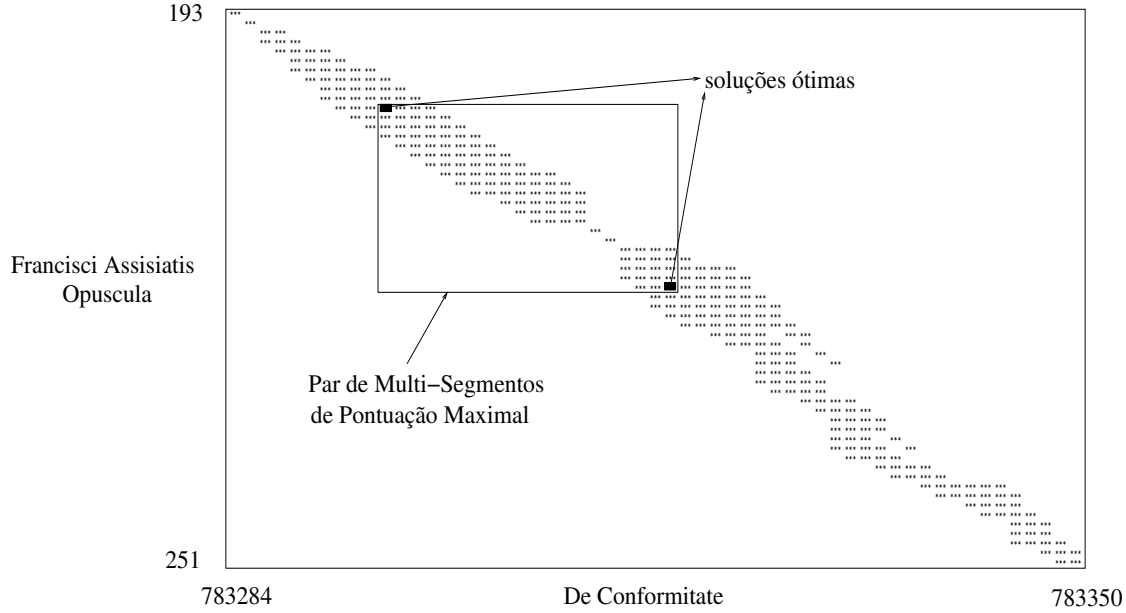


Figura 5.18: *Matriz de programação dinâmica para o PMSPM da figura 5.15*

Nesta seção foi mostrado o funcionamento do BLASTD para encontrar alinhamentos locais de documentos de texto. Mostramos o processo de criação de Pares de Alta Pontuação utilizando técnicas de Busca Aproximada. Em seguida mostramos um índice compacto para encontrar a posição das sementes no documento secundário. Finalmente foi mostrada a expansão das sementes até encontrar alinhamentos locais de pontuações maximais representados por Pares de Multi-Segmentos de Pontuação Maximal.

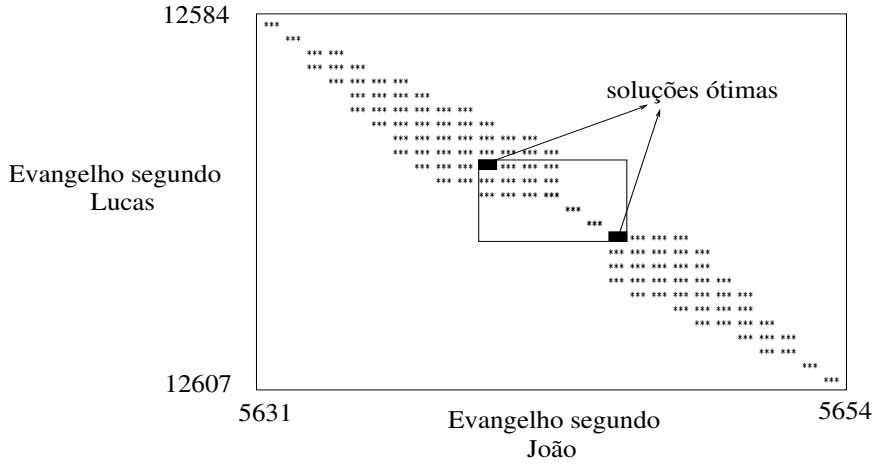


Figura 5.19: *Matriz de programação dinâmica para o PMSPM da figura 5.17*

Embora o alinhamento local de pares de documentos seja uma ferramenta importante, ela não é capaz de mostrar se dois ou mais documentos compartilham segmentos em comum com o documento de referência. Para isto é necessário aproveitar os alinhamentos locais encontrados pelo BLASTD e em seguida realizar alguma forma de alinhamento múltiplo. Na continuação mostramos um algoritmo que resolve o problema do alinhamento múltiplo para documentos de texto.

5.3 Alinhamento múltiplo de documentos

Em Bioinformática o alinhamento local de pares de sequências biológicas é aplicado em sequências que possuem uma relação biológica conhecida e assim o alinhamento fornece os segmentos que ambas as sequências têm em comum.

No entanto, no caso do alinhamento múltiplo, a intuição é oposta: descobrir segmentos em comum entre sequências que não possuem uma relação biológica clara [Gus97].

No caso de documentos de texto o alinhamento de pares de documentos revela segmentos em comum. Mas para o caso do alinhamento múltiplo a necessidade não está focada na descoberta de novas relações entre documentos mas em descobrir coberturas do documento de referência pelos documentos secundários. Diferente das sequências biológicas, no caso de documentos de texto a relação geralmente já é conhecida mesmo pelo tópico do assunto que os documentos tratam ou pelos autores dos documentos.

Neste trabalho estamos interessados em descobrir quais são os segmentos em comum de uma coleção de documentos que previamente possuem uma relação conhecida.

5.3.1 Alinhamento Múltiplo de Documentos pela algoritmo Estrela

As técnicas de Alinhamento Múltiplo para sequências biológicas foram descritas na seção 5.1.3, e uma solução exata para o problema geral que use as técnicas usuais de programação dinâmica é exponencial no número de sequências. Dentre as heurísticas normalmente utilizadas, o algoritmo estrela é polinomial, possui complexidade computacional razoável, é bastante simples e possui garantias de aproximação em relação à solução ótima quando a pontuação usada nas colunas é a da somas por pares. O algoritmo constrói progressivamente um alinhamento múltiplo a partir do conjunto de alinhamentos das diversas sequências contra um mesmo documento de referência.

Utilizamos o mesmo procedimento para encontrar alinhamentos múltiplos de documentos de texto a partir de alinhamentos locais produzidos pelo BLASTD.

Seja $\mathcal{C}$ uma coleção de documentos e seja $R \in \mathcal{C}$ o *documento de referência* pré-determinado. Vamos denotar por $\mathcal{S} = \mathcal{C} \setminus \{R\}$ o conjunto de *documentos secundários* que são alinhados contra R. A partir da saída dos alinhamentos locais obtidos por BLASTD para cada par (R, S), onde $S \in \mathcal{S}$, são construídos *blocos de alinhamento* que correspondem ao alinhamento múltiplo de diversos segmentos das sequências de $\mathcal{C}$.

Mais formalmente, *um bloco b é um par* (h, L) onde h é o *multi-segmento de referência*, com segmentos contíguos de R separados por espaços, e L é um conjunto de *multi-segmentos secundários*, onde cada multi-segmento $l \in L$ é formado por segmentos contíguos de uma sequência de $\mathcal{S}$ e tal que o par (h, l) seja sempre um PMS. A figura 5.20 ilustra estes conceitos com um bloco onde cada multi-segmento tem comprimento 8.

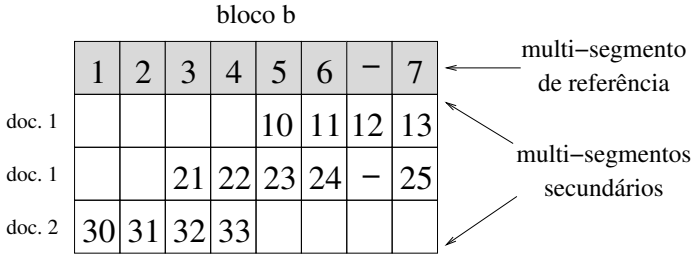


Figura 5.20: *Exemplo de um bloco com 3 multi-segmentos secundários alinhados a um multi-segmento de referência. Cada número de 1 a 43 corresponde ao identificador de um termo. O segundo dos multi-segmentos secundários é formado pelos dois segmentos dados pela sequências de termos 21,22,23,24 e pela sequência com o termo 25; e por dois espaços que precedem o primeiro segmento e mais um que sucede o mesmo segmento. As posições nos documentos não estão informadas na ilustração.*

Um bloco de alinhamento b representa o alinhamento múltiplo de vários Pares de Multi-Segmentos (PMS). Dado um bloco de alinhamento $b = (h, L)$, definimos as *funções* $g(b) = h$ *e* $L(b) = L$ obtém o multi-segmento de referência e um conjunto de multi-segmentos secundários do bloco, respectivamente. Definimos também a *função início do bloco b por* $s(b) = s(h)$ e a *função fim do bloco b por* $e(b) = e(h)$, as posições do início e do fim do multi-segmento de referência h, respectivamente. Observe que um par de multi-segmentos (PMS) é no fundo um caso particular de um bloco em que o conjunto de multi-segmentos secundários contém um único multi-segmento. Assim, o bloco b integra os $|L|$ pares de multi-segmentos (h, l) tais que $l \in L$ e representa um alinhamento múltiplo dos $|L|+1$ multi-segmentos $\{h\} \cup L$. Supondo pontuação estritamente negativa para espaços, os PMSs (h, l) que possuem espaços em uma das pontas de seus multisegmentos ou possuem dois espaços numa mesma coluna não são PMSPMs. É o caso de cada cada um dos três PMSs expostos na figura 5.20. Ao se eliminar tais colunas de seus respectivos alinhamentos e de seus PMSs, obtêm-se PMSs que podem ser PMSPMs. Observe que da eliminação de tais colunas nos três PMSs da figura 5.20, obtém-se os três PMSs da figura 5.21, que também ilustra o conceito de intersecção de blocos.

Dados dois blocos b e b' com a mesma sequência de referência R, dizemos que *os dois blocos b e b' se intersectam quando* $s(b') \leq e(b)$ e $s(b') \leq e(b)$.

Definimos s seguir duas operações sobre blocos usadas na exposição do algoritmo estrela:

União de blocos Sejam $b = (h, L)$ e $b' = (h', L')$ dois blocos que se intersectam. Definimos a *união dos blocos* $b \sqcup b'$ como sendo o bloco $b'' = (h'', L'')$ obtido da união h'' dos multi-segmentos h e h', e da união L'' das adequações de L e de L' à referência h''.

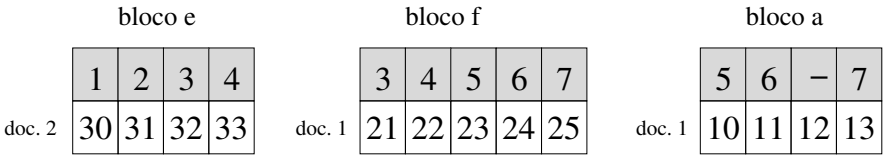


Figura 5.21: *Exemplo de blocos que se intersectam. Cada número de 1 a 33 corresponde ao identificador de um termo. A sequência de referência para os três blocos é $R = 1,2,3,4,5,6,7$. As posições nos documentos não estão explicitamente informadas na ilustração, mas elas são univocamente determinadas na sequência de referência. Os blocos e e f se intersectam, bem como os blocos a e f. Já os blocos a e e não se intersectam.*

O novo multi-segmento de referência h'' é formado pela união dos segmentos originais posicionados com a inserção de espaços em cada ponto de inserção dada pelo máximo de espaços inseridos no mesmo ponto em cada multi-segmento de referência. Espaços nunca são eliminados: uma vez *gap*, sempre *gap*. Por exemplo, a união dos segmentos de referência dos blocos a e f e e da figura 5.21 resulta no segmento de referência do bloco b da figura 5.20. (De fato, $b = (a \sqcup f) \sqcup e$.)

Uma vez obtida a nova referência h'', cada multi-segmento l de L (ou de L') é estendido para o multi-segmento $\epsilon(l)$ tendo por referência a transformação que leva de h (ou de h') ao multi-segmento de referência h''.

A cada ponto de inserção em h (ou em h') que recebeu uma certa quantidade de espaços, a mesma quantidade de espaços é inserida no ponto correspondente do multi-segmento l. Ademais, a cada inserção de símbolos de R em uma das pontas de h (ou de h') corresponde a inserção da mesma quantidade de espaços na extremidade correspondente de l. Assim, as colunas do alinhamento associado ao PMS (h,l) (ou (h',l)) são preservadas no alinhamento associado ao novo PMS $(h'',\epsilon(l))$ e todas as demais colunas são formadas unicamente por espaços ou são espaços acrescidos nas pontas do PMS secundário. Por fim,

$$L'' = \{\epsilon(l) \mid l \in L \cup L'\}.$$

A figura 5.22 mostra um exemplo deste procedimento.

Quebra de bloco Para efeito de visualização, um bloco b pode ser quebrado de acordo com um particionamento das colunas em grupos de colunas consecutivas. Por exemplo, pode-se fazê-lo de forma a eliminar trechos de multisegmentos secundários formados apenas por espaços nas extremidades dos respectivos multi-segmentos secundários. Neste caso, um novo grupo de colunas é iniciado sempre que se atinge uma extremidade de um dos multi-segmentos secundários. A figura 5.23 mostra um exemplo de uma tal quebra de bloco.

Como visto acima, as colunas de um alinhamento local entre uma sequência de referência e uma sequência secundária são preservadas após sucessivas uniões de blocos. Ademais, as outras colunas contêm apenas espaços ou contêm ainda um espaço no multi-segmento secundário que foi inserido em uma de suas pontas. Seja pois $p = (h,l)$ um PMS. Definiu-se acima o alinhamento associado ao PMS p. Definimos agora o *alinhamento local do PMS p* como sendo o PMS $\lambda(p)$ obtido de p pela eliminação das colunas do alinhamento de p que sejam extensões por espaço nas pontas da sequência secundária ou que tenham espaços nas duas linhas. Ademais, dado um bloco $b = (h,L)$, definimos o *conjunto*

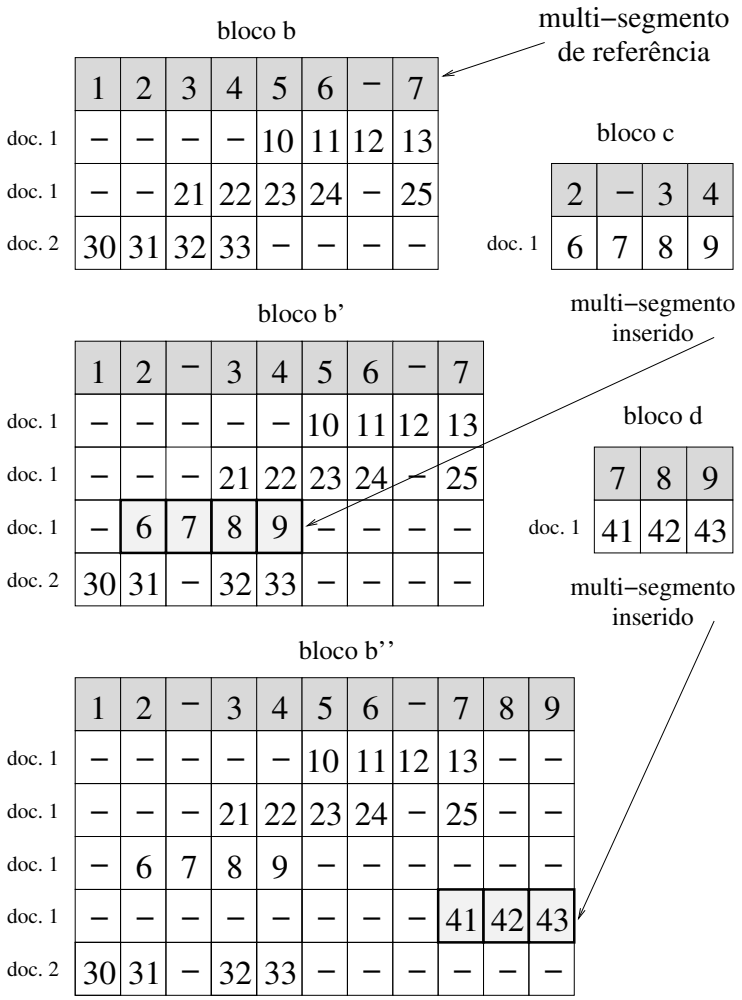


Figura 5.22: *Operação de união de blocos. Os números inteiros de 1 a 43 identificam termos dos documentos. Temos duas uniões representadas: $b' = b \sqcup c$ e $b'' = b' \sqcup d$. Observe que a união $b \sqcup c$ acrescenta um novo multi-segmento secundário em b' mesmo que o novo multi-segmento possa ser combinado com o primeiro multi-segmento secundário de b para formar um único multi-segmento. Observe também que esta união com c provoca a inserção de espaço no multisegmento de referência de b para formar o de b' bem como em cada um dos multi-segmentos secundários oriundos do bloco b. Analogamente, a união $b'' = b' \sqcup d$ não combina multi-segmentos secundários.*

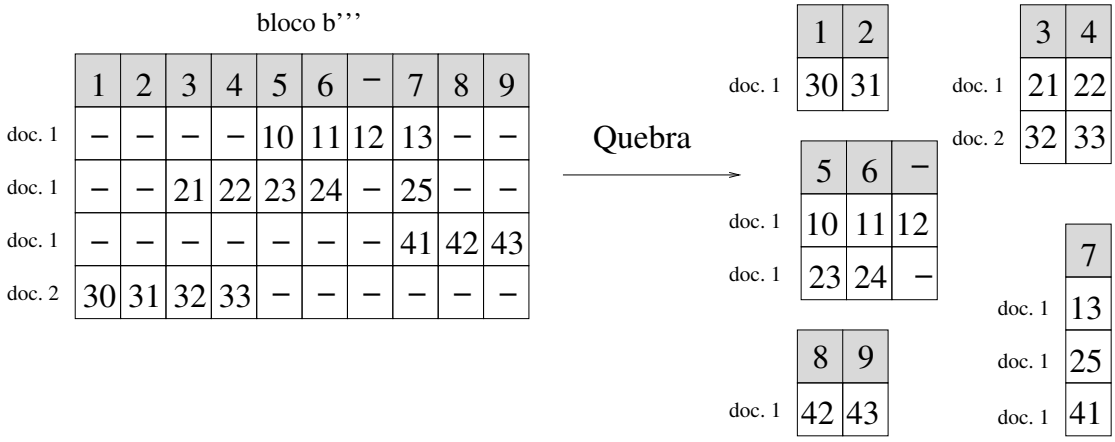


Figura 5.23: *Quebra de blocos de alinhamento. O bloco $b''' = b \sqcup d$ é quebrado em vários blocos. (Os blocos b e d estão expostos na figura 5.22.) Observe que os blocos obtidos pela quebra podem conter espaços nos seus multi-segmentos de referência.*

dos alinhamentos locais de b como sendo

$$\lambda(b) = \{\lambda(p) \mid p = (h, l), l \in L\}.$$

Observe que λ aplica o bloco b da figura 5.20 nos alinhamentos dos blocos a, f e e da figura 5.21. Ademais, $b = (a \sqcup f) \sqcup e$. Podemos enunciar o seguinte lema:

Lema 3 (Conservação dos alinhamentos locais)**.** *Dado um conjunto de alinhamentos locais A que unidos ordenadamente resultam num bloco de alinhamentos b, então o conjunto* $\lambda(b)$ *dos alinhamentos locais de b é o próprio conjunto de alinhamentos locais A, isto é:*

$$\lambda(b) = A.$$

Uma vez definidas as operações sobre blocos de alinhamento, mostramos o Algoritmo 11 que obtém um bloco de alinhamento formado a partir de alinhamentos locais entre o documento de referência $R \in \mathcal{C}$ e as sequências secundárias $S \in \mathcal{S} = \mathcal{C} - \{R\}$. O conjunto Q guarda um conjunto de blocos correspondentes aos alinhamentos locais já calculados, unindo blocos que se intersectam.

Algoritmo 11 Algoritmo Estrela(*StarAlignment*)

Entrada:
$\mathcal{C}$ coleção de documentos
R documento de referência, presente em $\mathcal{C}$
Saída:
blocos de regiões conservadas

```
1  Q ← ∅
2  para todo S ∈ C \ {R} faça
3     A_S ← Alinhamentos locais entre R e S obtidos pelo BLASTD
4     B_S ← Blocos gerados a partir de A_S
5     Q ← Q ∪ B_S
6     enquanto existir um par de blocos b e b' em Q que se intersectam faça
7        Q ← Q \ {b, b'}
8        Q ← Q ∪ {b ⊔ b'}
9     fim
10 fim
11 {Os blocos de Q são maximais, não se intersectam e λ(Q) = ⋃_{S∈S} A_S}
12 devolva Q e a Quebra de Q
```

Devido ao lema anterior, à linha 6 do algoritmo 11 temos a propriedade invariante de que $\lambda(Q)$ é a união dos A_S já calculados, já que os alinhamentos locais perdidos à linha 7 são recuperados à linha 8. Ao final deste laço, também temos que não existem dois blocos em Q que se intersectem e, ao final do laço da linha 2, também temos que todos os alinhamentos de A_S possíveis já foram calculados. Ademais, eles foram unidos entre si até formarem os blocos maximais presentes em Q, e que também não se intersectam. Neste ponto,

$$\lambda(Q) = \bigcup_{S \in \mathcal{S}} A_S.$$

A figura 5.24 mostra o exemplo de um bloco de alinhamento local múltiplo obtido como quebra em colunas da saída do algoritmo 11 para os três documentos antigos com

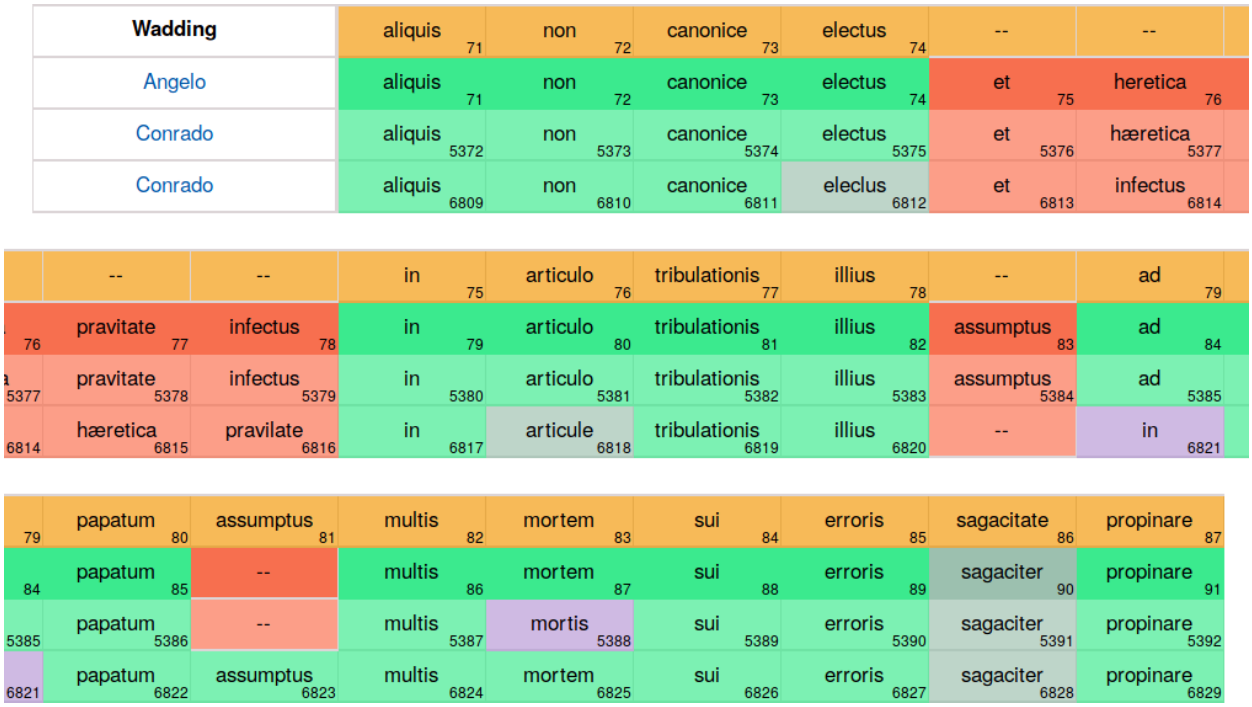


Figura 5.24: *Exemplo de grupo de 22 colunas do alinhamento múltiplo onde participam os alinhamentos locais AW1, WC5 e WC8 exibidos na figura 5.25 ($k = 4$, $t_{igual} = 3$, $t_{similar} = 1$, $t_{dispar} = -1$, $t_{\bar{e}} = -2$, $X = 10$). As 22 colunas foram manualmente divididas em três partes para disposição na figura.*

alinhamentos locais representados na figura 5.25. O bloco da figura 5.24 exibe um grupo de colunas que participam dos alinhamentos locais AW1, WC5 e WC8 exibidos na figura 5.25. É de posse da visualização de diversos blocos comparativos como estes é que o pesquisador poderá estudar com mais profundidade os documentos de interesse. Por exemplo, na figura 5.24 pode-se notar que o texto compilado por Wadding não apresenta a locução "et haeretica pravitate infectus" ("e infectado com uma depravação herética"), ao contrário dos demais trechos. Ademais, a troca de "articulo" por "articule" à posição 6818 de Conrado se não constitui um erro ortográfico pode ser devido a um erro de OCR, o que merece ser inspecionado na imagem ou no documento original. Convém também notar que a locução "ad Papatum assumptus" ("ao Papado ascendido") presente no texto de Wadding não se repete literalmente nos demais trechos expostos, muito embora em dois trechos apareçam as mesmas palavras com "assumptus" na primeira posição.

Ainda a respeito do exemplo da figura 5.24, convém observar que também Bartolomeu de Pisa expõe em sua obra um texto que alinha com este bloco de alinhamento da figura 5.24: "Et quod aliquis non canonice electus et haeretica pravitate infectus, in articulo tribulationis illius ad papatum assumptus, multis mortem sui erroris sagaciter propinare moliretur." Além disso, Marcos de Lisboa escreveu em sua obra do século XVI: "*E o que for não canonicamẽte electo, mas sospeito de heresia, ou intruso em o Pontificado em a conjunção destes trabalhos, será obedecido e sagazmente converterà muitos ao mortifero erro seu.*"

Se nos interessa também saber o quanto do texto de Wadding é coberto por alinhamentos locais contra os documentos analisados, a própria figura 5.25 nos mostra que diversas regiões do texto compilado por Wadding são cobertas apenas por alinhamentos contra Verba Conradi. Ademais, como diversos trechos de Verba Conradi aparecem repetidos, possivelmente também devido a citações e comentários feitos pelo próprio Paul Sabatier, fica a pergunta de que poderíamos nos preocupar em formular o seguinte pro-

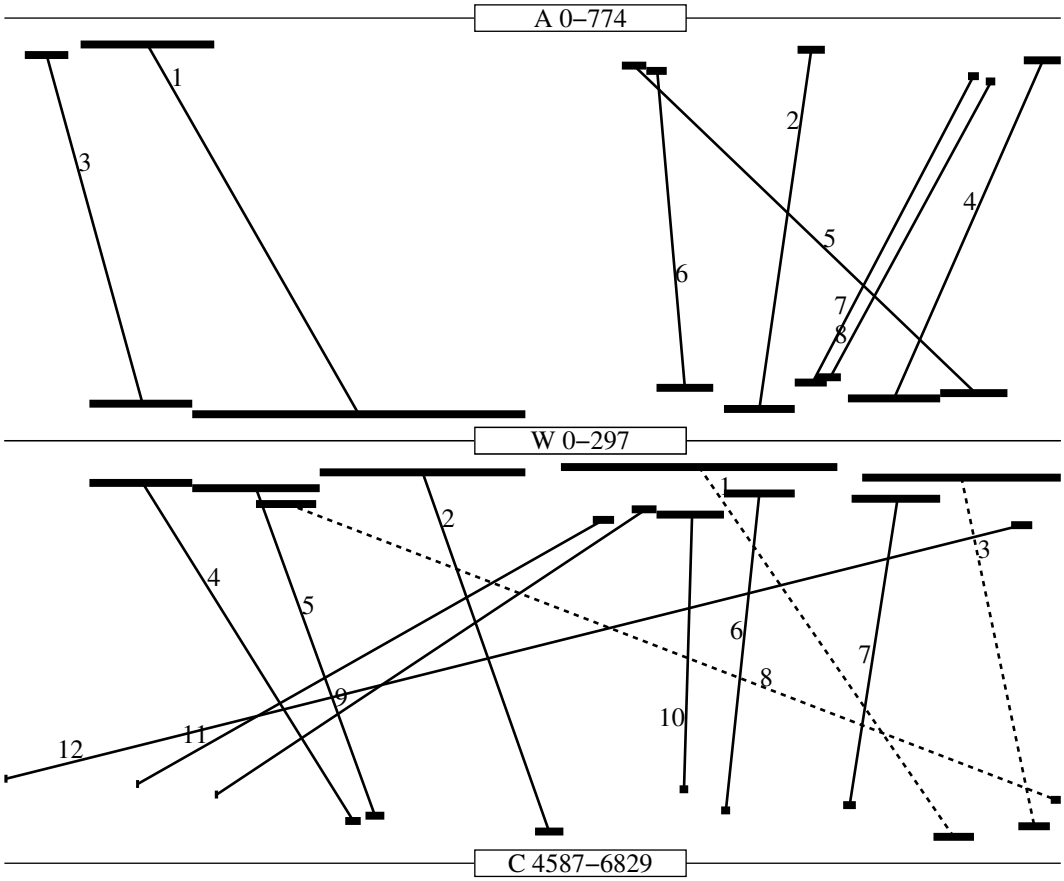


Figura 5.25: *Alinhamentos locais dos textos de Ângelo Clareno (A) e artigo com as Palavras de Conrado (C) contra o documento de Wadding (W) – ($k = 4$, $t_{igual} = 3$, $t_{similar} = 1$, $t_{dispar} = -1$, $t_{\bar{e}} = -2$, $X = 10$). Os alinhamentos locais são denotados: AW1, AW2, . . . , AW8; ou WC1, WC2, . . . , WC12.*

blema de interesse:

> Dada uma coleção de intervalos (formados pelas posições do segmento de referência original de um alinhamento local), cada um deles pontuados (pelo seu comprimento, pelo número de pares de termos iguais ou similares no alinhamento local, ou pela própria pontuação do alinhamento local), obter uma subcoleção de intervalos que não se intersectem e que tenha o máximo total de pontuação.

Se as pontuações forem sempre unitárias e os intervalos forem vistos como vértices de um grafo em que as arestas conectam intervalos que se intersectam, o problema a que se reduz é de encontrar um subconjunto vértices independentes de máxima cardinalidade, um problema NP-difícil. Contudo, para o tipo de grafo em questão, grafo de intervalos, uma solução polinomial é conhecida, mesmo para pontuações positivas quaisquer. No caso dos intervalos das posições em Wadding dos alinhamentos locais contra Conrado, aqueles associados aos cinco alinhamentos WC4, WC5, WC2, WC1 e WC3 obtêm a máxima pontuação quando esta é dada pelo comprimento dos intervalos.

A este conjunto de cinco intervalos, porém, não se pode acrescentar o intervalo relativo a WC7, pois ele intersecta o relativo a WC3, ainda que os três termos "capiti conformentur fiducialiter" ("que sejam conformados fielmente à Cabeça") sejam cobertos apenas por aquele intervalo, de forma que isto motiva-nos a indagar se não é possível reformular o problema de forma a aceitar também alguma forma composição de intervalos que se sobreponham mutuamente.

Voltando ainda ao nosso exemplo, um exame mais atento revela que os intervalos WC1 e WC3 não pertencem às Palavras de Conrado propriamente ditas. É que o documento com as Palavras de Conrado usado nas figuras 5.24 e 5.25, ao contrário dos outros dois, é saída automática de um sistema OCR que processou as 22 páginas do artigo de Paul Sabatier em que foram publicadas as Palavras de Conrado, de forma que os alinhamentos WC1, WC3, e WC8 envolvem comentários e citações de outras obras feitas por Sabatier em seu artigo mas não as Palavras de Conrado por ele compiladas. Assim, o mais bem pontuado conjunto de alinhamentos contra as Palavras de Conrado que produzem intervalos no texto de Wadding que não se sobrepõem é o formado pelos alinhamentos WC4, WC5, WC2, WC11, WC9, WC10, WC6, WC7 e WC12. Observe que os intervalos WC11, WC9 e WC12 cobrem trechos do texto de Wadding não cobertos pelo texto de Ângelo Clareno, ao passo que os intervalos dos alinhamentos AW7, AW8 e AW5 do texto de Ângelo Clareno contra o de Wadding cobrem trechos não cobertos pelos alinhamentos contra Conrado. Note que os intervalos AW7 e AW8 também se intersectam.

5.4 Conclusões

Neste capítulo mostramos como é realizado o alinhamento de documentos de texto considerando erros entre os termos. Foi exposta a concepção do algoritmo BLASTD, uma de implementação tipo BLAST com espaços para alinhar pares de documentos de texto. Para encontrar os alinhamentos locais foram utilizadas técnicas de Busca Aproximada para encontrar as sementes de alinhamento. Ademais, com base na saída do BLASTD foi proposto um algoritmo para encontrar alinhamentos múltiplos de documentos que se baseia no algoritmo estrela para sequências biológicas e foi mostrado como esse algoritmo agrupa alinhamentos locais em blocos de alinhamento.

Capítulo 6

Treinamento de OCR baseado em alinhamentos locais

A principal componente de sistemas OCR é o classificador, que atribui a uma imagem um caractere de texto. Geralmente, estes classificadores são treinados sobre um conjunto de pares ordenados formados por uma imagem e um caractere de texto associado. Além da informação das imagens, outros recursos estranhos a um classificador propriamente dito podem ser usados para apoiar o reconhecimento, como por exemplo a utilização de informação linguística para corrigir, a posteriori, possíveis inconsistências nas palavras reconhecidas pelo classificador. No que diz respeito ao classificador em si, é parte crítica no processo de elaboração do sistema OCR a etapa de treinamento do classificador.

Neste capítulo mostramos uma aplicação do alinhamento local de documentos de texto apresentados no capítulo 5 para melhorar a qualidade de um sistema OCR através do retreinamento do classificador a partir dos erros detectados a posteriori. Essa aplicação foi projetada para usar o Tesseract que é um dos sistemas OCR de código aberto mais conhecidos na sua categoria.

Mostramos em seguida brevemente o funcionamento do Tesseract para logo mostrar o projeto do método que treina este classificador utilizando alinhamento local de textos.

6.1 Uma breve introdução ao Tesseract

O Tesseract é um sistema OCR de código aberto que foi desenvolvido pela HP entre os anos 1984 e 1994, a partir do projeto de doutorado [SW87] de Ray Smith. A motivação do projeto era melhorar a acurácia pois os sistemas desenvolvidos à época tinham uma performance ruim. Em 1995, depois que o projeto finalizou, o sistema participou de uma competição importante na área para medir a sua acurácia. A sua participação obteve bons resultados mas somente se tornou um projeto de código aberto em 2005.

Somente em 2007, na conferência ICDAR, a arquitetura do Tesseract foi apresentada pelo próprio Smith [Smi07]. O sistema implementa um fluxo de processamento de imagens bastante comum para este tipo de problemas: segmentação de linhas, palavras e caracteres. Dado que na época da sua criação a HP construiu um sistema próprio de binarização, o Tesseract então supõe que a imagem já tenha sido previamente tratada.

A segmentação de linhas é bastante robusta e pode reconhecer linhas com inclinações, o que permite conservar a qualidade da imagem. Por outro lado, ainda na etapa de segmentação de linhas, o Tesseract é capaz de identificar caracteres muito pequenos a partir das linhas ascendentes e descendentes da região sendo processada.

Com respeito à segmentação de palavras, estas são classificadas em dois tipos: aquelas que tem caracteres regulares, ou seja, a largura é a mesma, fixada, entre todos os caracteres; e aquelas que possuem caracteres com largura variável, também dita proporcional. Caso a região possua caracteres regulares então a segmentação de caracteres utiliza o valor da largura encontrada para essa região. Se a região for do segundo tipo, a segmentação verifica os espaços horizontais entre os caracteres daquela região para produzir uma segmentação.

Neste ponto do fluxo, ambos os tipos de regiões são processados pelo classificador para obter o texto correspondente. No entanto, as regiões com caracteres de largura proporcional são processadas para gerar mais hipóteses de segmentação. As regiões de segmentação são geradas quebrando os caracteres que têm ligações a partir de marcadores no contorno da imagem como mostra a figura 6.1.

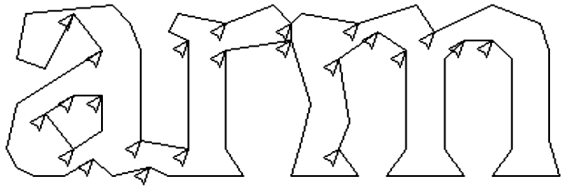


Figura 6.1: *Pontos de quebra (adaptado de [Smi07]). A imagem mostra também um segmento de quebra selecionado entre o 'r' e 'm'.*

Qualquer região de segmentação que piorar a pontuação do resultado é ignorada, mas não é descartada de maneira que possa ser utilizada na etapa de associação de segmentos.

A etapa de associação de segmentos utiliza o algoritmo de busca A* (busca em profundidade) no grafo de segmentação das possíveis combinações das regiões para encontrar caracteres candidatos. O processo não constrói o grafo inteiro, mas mantém uma tabela de espalhamento com os estados visitados. A busca A* procede retirando novos estados candidatos de uma fila de prioridade e avaliando os estados mediante a classificação das combinações de regiões. Uma parte importante do sucesso deste algoritmo se deve ao classificador de caracteres que pode reconhecer caracteres quebrados (como é mostrado na figura 6.2).

Com respeito às características das imagens, o Tesseract usa segmentos da aproximação poligonal da região que contém um caractere. Essa abordagem possui uma desvantagem quando tenta reconhecer caracteres deteriorados ou desconhecidos. Para contornar este problema, são obtidos um número maior de segmentos do caractere desconhecido e esses segmentos são emparelhados com segmentos de um caractere "protótipo". A figura 6.2 é um exemplo deste procedimento. Nessa figura, a imagem da esquerda é o protótipo enquanto a imagem do meio é um caractere desconhecido. Na terceira imagem, os segmentos de linha contínua são as características do caractere protótipo, enquanto que, a linha pontilhada mostra os segmentos do caractere desconhecido. Ao todo, são 5 segmentos que não foram emparelhados de maneira precisa.

Na etapa de treinamento, os segmentos do conjunto de treinamento são aglomerados e as classes de caracteres são associadas a cada segmento. Dado um caractere desconhecido, o procedimento de classificação é um sistema de votação simples onde o protótipo vencedor é aquele que tiver maior número de segmentos em comum com a imagem do caractere desconhecido. Para um segmento de um protótipo as características extraídas são tuplas (x, y, α, l) onde x, y são as coordenadas do segmento, α é o angulo, e l é o comprimento do segmento. Para um segmento de um caractere desconhecido as características são tuplas (x, y, α). Geralmente, para um caractere protótipo, são extraídos entre 50 e 100

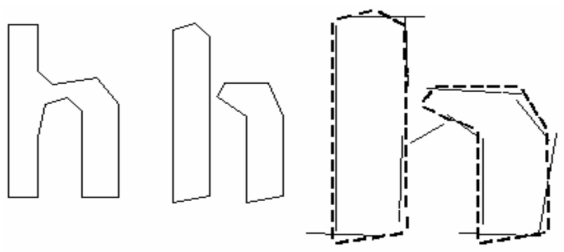


Figura 6.2: *Características extraídas pelo Tesseract (adaptado de [Smi07]). A primeira imagem à esquerda mostra um caractere protótipo, enquanto que a segunda imagem mostra um caractere desconhecido composto por dois pedaços. Na terceira imagem são mostradas as características do caractere desconhecido emparelhados com o protótipo.*

tuplas de características, enquanto que para um caractere desconhecido são extraídas entre 10 e 20 características.

6.1.1 Informação linguística

Ao segmentar uma palavra o Tesseract utiliza informação linguística para obter uma nota para essa segmentação. As hipóteses de segmentação são pontuadas de acordo com informação linguística disponível no Tesseract. Essa informação linguística é composta por vários tipos de dicionários: palavras mais frequentes, palavras com caracteres numéricos, palavras com caracteres maiúsculos, e palavras com caracteres minúsculos. O sistema então escolhe a melhor palavra disponível em cada categoria. Os caracteres de cada palavra são ponderados de acordo com a distância com o seu protótipo e o comprimento do polígono que contém a palavra, logo a palavra com a melhor soma ponderada dos seus caracteres é selecionada como a segmentação final.

6.2 Treinamento semiautomático

O treinamento de um sistema OCR é uma tarefa bastante árdua pois é necessário obter grandes quantidades de imagens anotadas com caracteres de texto. Nesta seção apresentamos um método semiautomático para treinar o classificador a partir de informação de conjuntos de imagens.

O método, desenvolvido em Python e mostrado na figura 6.3, é um laço que a cada etapa executa o reconhecimento de um conjunto de imagens cadastrado no sistema. A saída do reconhecimento são arquivos em formato HOCR para cada imagem que compõe o conjunto. O formato HOCR é um padrão para representar informação da saída de um sistema OCR utilizando notação HTML. O arquivo nesse formato inclui informação sobre as páginas, parágrafos, linhas e palavras da página que foi processada. A informação é composta, principalmente, pelas coordenadas do retângulo que contém algum elemento do layout (página, parágrafo, linha ou palavra). Um exemplo de formato HOCR é mostrado na listagem 6.1:

```
1 <span
2    class='ocr_line'
3    id='line_1_2'
4    title="bbox 236 104 1633 185; baseline 0.004 -26; x_size 74;
5           x_descenders 19; x_ascenders 21">
6      <span class='ocrx_word'
```

```
7             id='word_1_4'
8             title='bbox 236 105 341 178; x_wconf 78;'>
9               <strong>plus</strong></span>
10      <span class='ocrx_word'
11            id='word_1_5'
12            title='bbox 353 104 490 160; x_wconf 73;'>(olito</span>
13      <span class='ocrx_word'
14            id='word_1_6'
15            title='bbox 504 105 720 175; x_wconf 74;'>(oluctur ,</span>
16      <span class='ocrx_word'
17            id='word_1_7'
18            title='bbox 732 107 900 162; x_wconf 65;'>noüræ</span>
19      <span class='ocrx_word'
20            id='word_1_8'
21            title='bbox 913 108 1173 165; x_wconf 68;'>Rclwionis</span>
22      <span class='ocrx_word'
23            id='word_1_9' title='bbox 1190 120 1238 163; x_wconf 99;'>
24              &amp;</span>
25      <span class='ocrx_word'
26            id='word_1_10' title='bbox 1251 108 1447 164; x_wconf 66;'>
27              <strong>allatum</strong></span>
28 </span>
```

Listagem 6.1: *Exemplo de arquivo em formato HOCR*

Por exemplo, entre as linhas 8 e 9, o sistema OCR reconheceu a palavra "plus" contida no retângulo definida pelas coordenadas (236,105) e (341,178), que correspondem ao canto superior esquerdo e inferior direito respectivamente.

Contudo, essa informação não é suficiente para o propósito de treinar o classificador pois, para treiná-lo, precisamos de um arquivo com pares ordenados que associa imagens com caracteres de texto e, ainda, poder corrigir tal arquivo com os caracteres corretos obtidos a partir de um alinhamento local.

Assim, fizemos alterações no Tesseract para produzir informação adicional no formato HOCR. Essa informação inclui, para cada palavra, as posições dos retângulos que correspondem aos caracteres que compõem essa palavra. Na listagem 6.2, mostramos um trecho de um arquivo HOCR que inclui essa informação adicional. Entre as linhas 8 e 14 dessa listagem é mostrada a informação dos retângulos de cada caractere que compõe a palavra "TOMVS".

```
1 <span class='ocr_line'
2       id='line_1_1'
3       title="bbox 587 37 1633 94;
4       baseline 0.006 -18; x_size 58;
5       x_descenders 16; x_ascenders 9">
6   <span class='ocrx_word'
7       id='word_1_1'
8       title='bbox 587 38 850 76; x_wconf 83;
9             x_bboxes 587 38 622 76
10                     644 41 677 74
11                     701 39 744 75
12                     770 41 802 76
13                     828 40 850 73'>
14                <strong>TOMVS</strong>
15  </span>
16  <span class='ocrx_word'
17      id='word_1_2'
18      title='bbox 906 37 1274 76; x_wconf 74;
19             x_bboxes 906 37 941 74
```

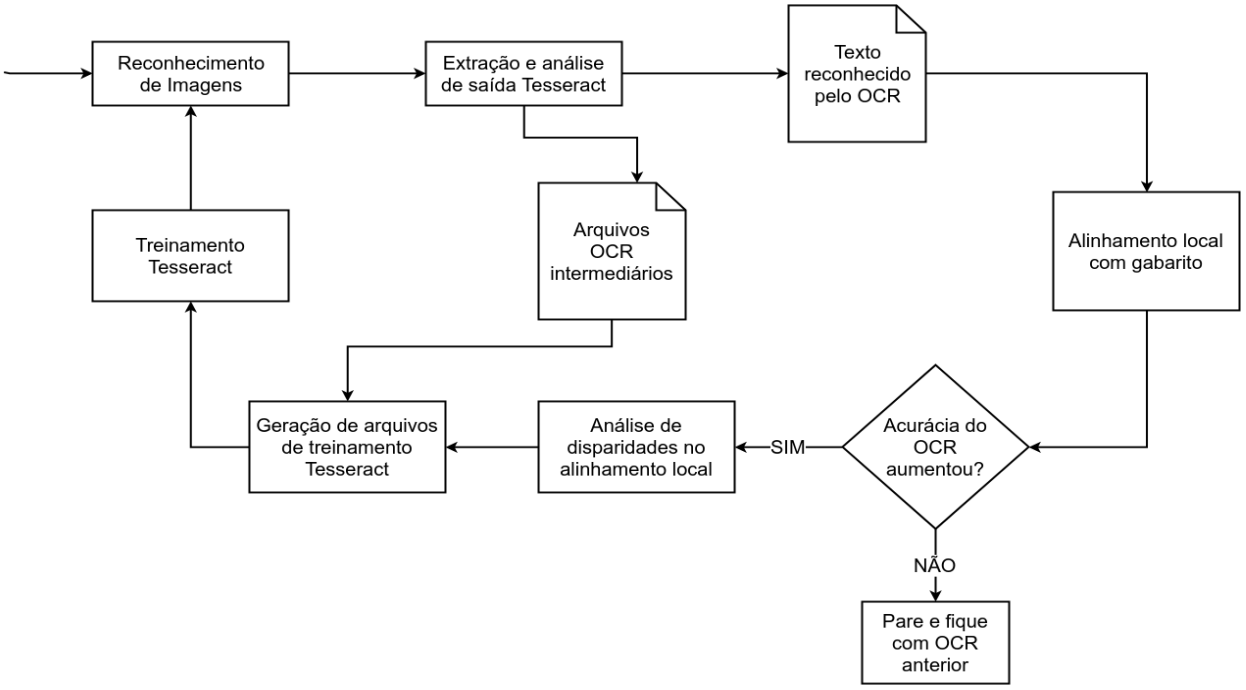


Figura 6.3: *Treinamento do Tesseract*

```
20                          967 38 995 73
21                          1019 38 1054 74
22                          1083 39 1115 74
23                          1142 39 1159 75
24                          1178 42 1210 76
25                          1237 41 1258 76
26                          1265 63 1274 75'>
27                      <strong>TERTIVS.</strong>
28   </span>
29 </span>
```

Listagem 6.2: *Arquivo HOCR incluindo os retângulos dos caracteres em cada palavra*

A saída HOCR nesse formato estendido é interpretada para produzir dois arquivos:

- Um arquivo de texto, que vamos chamar de *arquivo intermediário*, que lista os caracteres reconhecidos pelo OCR e as coordenadas do retângulos que contém esses caracteres. Um exemplo desse tipo de arquivo é mostrado na listagem 6.3.
- Um segundo arquivo de texto com os caracteres que fazem parte de todas as palavras reconhecidas no arquivo HOCR. Para facilitar o alinhamento com o gabarito, são inseridos espaços entre cada par de símbolos consecutivos. Um exemplo de deste arquivo de texto é mostrado na listagem 6.4.

```
T 587 38 622 76 0
O 644 41 677 74 0
M 701 39 744 75 0
V 770 41 802 76 0
S 828 40 850 73 0
T 906 37 941 74 0
E 967 38 995 73 0
R 1019 38 1054 74 0
```

```
T 1083 39 1115 74 0
I 1142 39 1159 75 0
V 1178 42 1210 76 0
S 1237 41 1258 76 0
. 1265 63 1274 75 0
```

Listagem 6.3: *Trecho de um arquivo intermediário com os caracteres de texto e suas coordenadas na imagem de entrada*

```
T O M U S T E R T I V S .
```

Listagem 6.4: *Trecho do arquivo de texto criado a partir de um arquivo HOCR*

Neste ponto na execução do fluxo da figura 6.3, foram produzidos os textos e arquivos intermediários para o conjunto de imagens que está sendo processado, é necessário então, verificar a acurácia do OCR comparando os resultados com o arquivo de gabarito.

A *acurácia* de um sistema OCR para uma imagem é determinada pela quantidade relativa de caracteres de texto que são iguais aos do gabarito para aquela imagem.

Dentro dos algoritmos apresentados no capítulo 5, o algoritmo de alinhamento local BLASTD pode nos ajudar a obter uma medida entre o texto produzido por um sistema OCR e o gabarito. A aplicação do algoritmo é direta, basta colocar no sistema todos os textos obtidos pelo OCR junto com o gabarito para, em seguida, produzir o alinhamento. Uma vez obtido o alinhamento, podemos corrigir os documentos intermediários (listagem 6.3, por exemplo) para finalmente produzir os arquivos de treinamento do classificador, a ser retreinado a partir principalmente das colunas onde há disparidade entre o gabarito e a saída do classificador.

É adequado reservar uma parte dos dados para treinamento e avaliação, como de costume na área (mas não discutiremos aqui).

O modelo treinado é atualizado dentro do sistema para ser utilizado na iteração seguinte complementando o modelo do Tesseract vindo de fábrica correspondente à língua configurada para o conjunto de imagens.

O sistema de treinamento de OCR, exposto na figura 6.3, foi testado em um conjunto de imagens que possui três páginas da obra *Beati Patris Francisci Assiciatis Opuscula* de Luke Wadding escrita em 1623 (veja seção 7.1). Para cada imagem foi produzida uma imagem binária utilizando as técnicas descritas no capítulo 3 com exceção da segmentação de caracteres. O gabarito utilizado para este experimento foi obtido a partir de um documento de texto obtido por um motor OCR e que foi corrigido manualmente. O gabarito final sofreu a inserção de espaços entre cada par de caracteres consecutivos para viabilizar o alinhamento com os textos do conjunto de imagens.

A figura 6.4, mostra visualmente o alinhamento local de uma página obtida pelo sistema OCR e o gabarito para o conjunto de imagens pertencente à obra *Beati Patris Francisci Assiciatis Opuscula*. Ao todo, existem 1749 colunas no alinhamento, 72.4% das quais correspondem a igualdade (1267 caracteres), 24.3% são colunas díspares (426 caracteres) e 3.2% são inserções ou remoções (56 caracteres).

A taxa de erro obtida para este documento, 27.5%, é alta se comparada com a taxa de erro para documentos modernos. Claramente, o Tesseract não foi treinado para reconhecer a tipografia usada em 1623 e é uma oportunidade para utilizarmos o método de correção proposto.

Alguns padrões de correção de caracteres merecem ser comentados. Na figura 6.5, foram destacadas disparidades, remoções e inserções do alinhamento local mostrado na figura 6.4. Podemos observar que para as colunas destacadas em azul a correção é direta: substituir o caractere do texto OCR (linha inferior) pelo gabarito (linha superior). Por

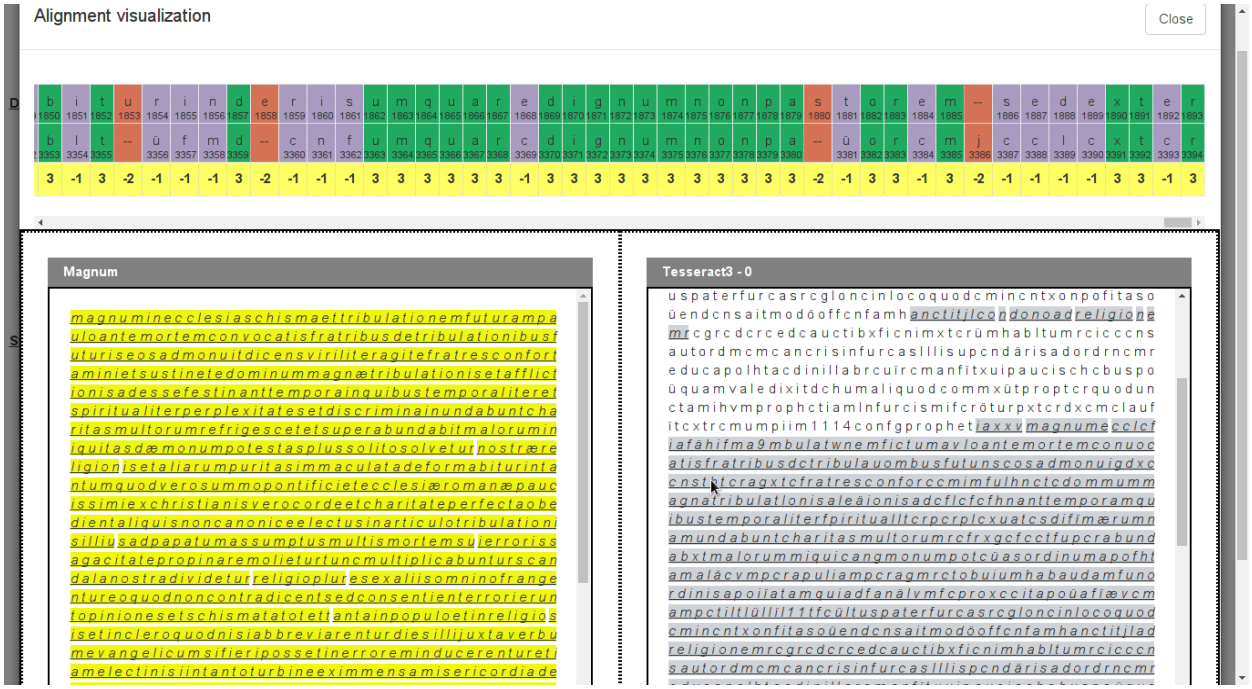


Figura 6.4: *Alinhamento local obtido entre um texto obtido pelo Tesseract e o gabarito desse texto. Os parâmetros utilizados ($t_{igual} = 3$, $t_{similar} = 0$, $t_{dispar} = -1$, $t_{\bar{e}} = -2$, queda máxima 10 e taxa de erro 20%), permitem identificar as disparidades de caracteres que são a fonte de correção dos arquivos de treinamento.*

outro lado, as colunas destacadas com uma caixa vermelha constituem correções de vários caracteres. Nas quatro caixas destacadas na figura 6.4 existem os seguintes padrões de correção:

- Substituir “üfm” por “urin”
- Substituir “cnf” por “eris”
- Substituir “ü” por “st”

Esses padrões começam com uma inserção de um caractere no documento secundário (que corresponde com o texto obtido pelo sistema OCR) e terminam com uma dispari-

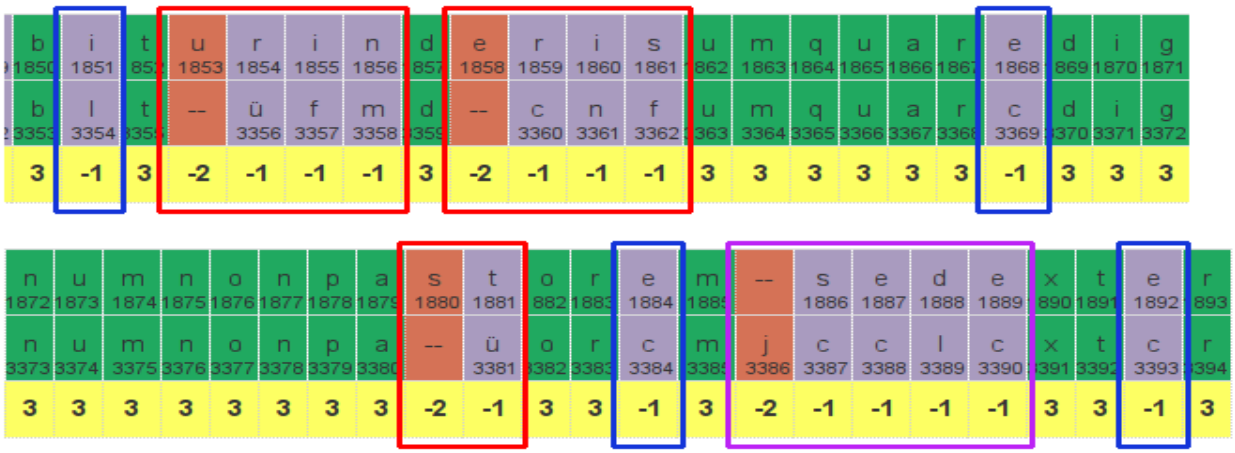


Figura 6.5: *Padrões de correção no alinhamento local de textos para treinamento do Tesseract*

dade. Podemos associar esse tipo de padrão a um erro de segmentação do sistema OCR ao tentar quebrar caracteres grudados.

Por outro lado, ainda na figura 6.5, a caixa de cor roxa situada à direita corresponde a um padrão onde um caractere foi removido do gabarito. Nesse caso, a remoção é descartada e todos os caracteres díspares do texto OCR são substituídos pelo gabarito ("cclc" por "sede").

Ambos os tipos de padrão de correção podem ser aproveitados no treinamento do sistema OCR utilizando as colunas díspares para corrigir o arquivo de treinamento. O Tesseract possibilita criar um arquivo com substituições de sequências de caracteres que corrijam a posteriori os resultados do algoritmo de segmentação em casos difíceis envolvendo a segmentação de caracteres grudados.

Dificulta bastante o treinamento o fato de que não temos acesso aos dados de treinamento do Tesseract. Se eles fossem disponibilizados, bastaria acrescentar os arquivos de treinamento corrigidos. A falta dos dados de treinamento persiste inclusive na versão mais recente do Tesseract (4.0). Existe uma interface que permite acréscimos de complementos ao conjunto de treinamento, mas ao testar a arquitetura não conseguimos fazer o OCR aprender com os novos dados, apesar de seguir a documentação disponível. É trabalho futuro resolver este problema.

Vale a pena ressaltar que para uma página do texto de *Verba Conradi* impresso em 1903 [Sab03], o Tesseract apresenta um comportamento melhor. Foi realizado um alinhamento entre essa página e o seu gabarito. O resultado possui 1338 caracteres (colunas), dos quais 96.9% são colunas de igualdade (1296 caracteres), 0.7% são colunas díspares (9 caracteres) e 2.5% são colunas de remoção (33 caracteres). Esta página foi impressa utilizando tipografia Times Roman que, naturalmente, é bastante similar à Times New Roman. Isto facilita a obtenção de caracteres de treinamento pelo Tesseract.

Existem outras tentativas de treinar o Tesseract utilizando alinhamento de palavras [Bos09]. Este trabalho aplica também técnicas de alinhamento de caracteres para treinar o Tesseract, mas não implementou o circuito mostrado na figura 6.3 o que deixa o método manual. É possível também utilizar um corretor ortográfico para introduzir mais alterações em regiões onde ambas palavras não possuem diferenças. Isto poderia ajudar a melhorar o treinamento do motor OCR. Por outro lado, o corretor ortográfico poderia ser também de ajuda para selecionar como documento de referência aquele que possui menor distância relativa a um texto ortograficamente correto.

6.3 Conclusões

Neste capítulo mostramos o projeto de um método que treina de forma semiautomática um sistema de OCR para viabilizar o reconhecimento em textos antigos. O sistema utiliza as técnicas de processamento de imagens e alinhamento local mostrados nos capítulos 3 e 5.

O método utiliza um fluxo onde combina etapas de processamento de um conjunto de imagens e o alinhamento local do texto produzido pelo sistema OCR com um gabarito para assim corrigir a saída do sistema OCR e produzir arquivos de treinamento para a próxima iteração.

O sistema OCR escolhido para treinar foi o Tesseract. Foi verificado que para o documento de texto de uma imagem do século 17 pertencente à obra *Beati Patris Francisci Assiciatis Opuscula* de Luke Wadding, o Tesseract obteve uma taxa de erro inicial de 27.5%. Mostramos como existem vários padrões de correção sugeridos pelo alinhamento local

do gabarito com o texto OCR. Estes padrões são aproveitados para inserir novos casos de treinamento ao Tesseract. Ainda é necessário investigar ajustes no método de treinamento do Tesseract para fazer com que realmente aprenda.

Com respeito ao método de segmentação de imagens com caracteres grudados do Tesseract, a busca A* é uma estratégia gulosa que pode encontrar soluções não ótimas para o problema da segmentação desse tipo de imagens. De fato, o próprio autor reconhece que algumas combinações de regiões podem ser ignoradas mas afirma que, ao usar o A*, não é necessário manter o grafo de segmentação inteiro. Desta forma, acreditamos que o algoritmo de programação dinâmica proposto para o problema da segmentação de caracteres no capítulo 3 possa oferecer uma contribuição original em relação ao Tesseract, mesmo que este sistema esteja em desenvolvimento a mais de duas décadas.

Capítulo 7

Experimentos

7.1 Conjuntos de dados

Nosso conjunto de dados é composto por imagens e texto que formam parte de um grande corpo de trabalhos efetuados por frades franciscanos pertencentes à Ordem dos Frades Menores (O.F.M.) fundada por São Francisco de Assis (1182-1226). O fundador deixou vários manuscritos que chegaram até nós em forma impressa. Além dele, os seus seguidores também foram prolíficos escritores produzindo documentos que tratam sobre a vida de São Francisco assim como da regra de vida que ele quis testemunhar para os seus seguidores.

7.1.1 Luke Wadding

O primeiro autor franciscano de nosso interesse é o historiador e teólogo Luke Wadding, O.F.M., que nasceu em Waterford, Irlanda, a 16 de outubro de 1588 e morreu em Roma, a 18 de novembro de 1657 [Cleb]. Conforme a Enciclopédia Católica descreve [Cleb], Luke Wadding era filho de Walter Wadding e Anastásia Lombard, parente próximo de Pedro Lombard, Arcebispo de Armagh, cidade do norte da Irlanda. Seu irmão Ambrósio, jesuíta, ensinava filosofia em Dillingen, Bavaria. Seus piedosos pais, instruíam todos os seus filhos a recitarem diariamente o Ofício Breve da Santíssima Virgem, como relata Fr. Harold [Har62]. Aos treze anos já havia adquirido um bom conhecimento dos clássicos, e havia aprendido a escrever em latim com facilidade. Não tardou para ser enviado ao seminário irlandês de Lisboa, para estudar com os jesuítas.

Estes estudos se desenvolveram, porém, dentro da ordem dos frades menores, onde fez a profissão solene em 1605 e foi ordenado sacerdote em 1613. Em Lisboa, durante um capítulo provincial, depois de uma atuação acadêmica brilhante, o vigário-geral da ordem Antônio de Trejo, enviou-o a Salamanca para que viesse a ter maiores oportunidades. Ali dominou o hebraico e escreveu sua obra sobre a origem e excelência da língua, tendo recebido a cátedra de teologia do Colégio de São Francisco.

Wadding trabalhou como professor até 1618, e apesar de possuir apenas trinta anos, foi escolhido pelo rei Felipe III como teólogo para a embaixada que Felipe enviou a Paulo V em apoio à doutrina da Imaculada Conceição. Antônio de Trejo, vigário-geral da ordem e bispo de Cartagena, havia sido nomeado para a embaixada e era patrocinador e admirador de Wadding.

Produção literária

Saindo da Corte do rei católico a 1 de outubro de 1618, a embaixada chegou a Roma a 17 de dezembro. Em busca de material para o trabalho que lhe havia sido confiado, assim como para seus outros estudos, Wadding passou dias inteiros nas bibliotecas de Roma, visitando também as de Nápoles, Assis, Perugia, e outras cidades. A elaboração do parecer mais importante da comissão, a preparação das súplicas ao papa, e a resolução das dificuldades teológicas, todo este trabalho em grande medida recaiu sobre Wadding.

Em maio do ano 1620, os enviados voltaram às suas dioceses na Espanha, mas a Wadding foi pedido de permanecer em Roma para ajudar ao novo encarregado. Enquanto a comissão durou, foi seu mais acreditado conselheiro teológico. Felipe IV, em uma carta bastante amável, agradece-o profundamente pelos serviços. As três opuscula sobre a redenção, batismo, e morte da Santa Virgem (1655 - 1656) foram escritas como contribuições aos problemas apresentados à comissão.

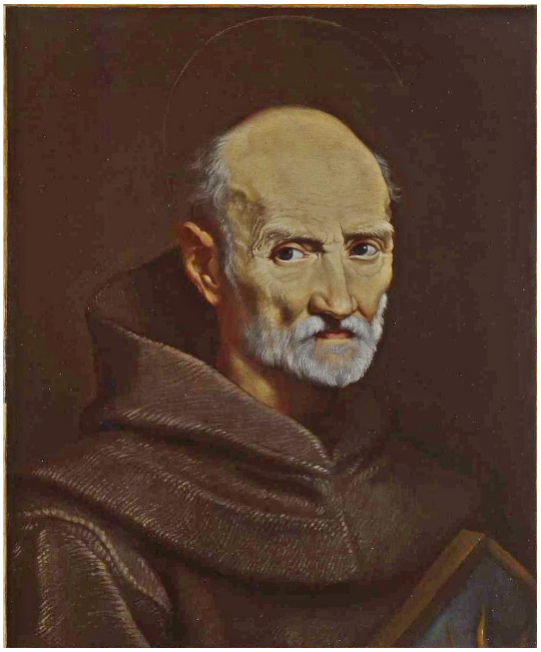


Figura 7.1: *Pintura a óleo de Luke Wadding por Carlo Maratta mantida na Galeria Nacional de Irlanda*

Além dos trabalhos requisitados pelo papa e pelo rei da Espanha, Wadding também se dedicou a atividades literárias da própria ordem, tendo encontrado apoio bastante eficaz no superior-geral, Benigno de Gênova, por cartas encíclicas à ordem, pediu em 1619 que em cada província alguém capaz se dedicasse a transcrever e remeter a Roma todos os documentos relativos à história da ordem. O material acumulado foi entregue a Wadding. Entre os mais destacados destes colaboradores encontram-se Bartolomeu Cimareli, trabalhando nos arquivos e bibliotecas do norte e do centro da Itália, e Jacobo Polius, nos da Alemanha. Como primícia, Wadding publicou em 1623 na Antuérpia uma edição completa com comentários de "Os escritos de São Francisco de Assis" [Wad23]. A obra foi publicada em latim, que Wadding tão bem dominava desde a infância, sob o título de *Beati Patris Francisci Assiciatis Opuscula*. Na cópia digitalizada da primeira edição desta obra clássica de Wadding, publicada em 1623, encontra-se esta profecia de São Francisco de Assis:

Magnum in Ecclesia schisma et tribulationem futuram.

Paulo ante mortem convocatis fratribus, de tribulationibus futuris eos admonuit, dicens: «Viriliter agite, Fratres, confortemini, et sustinete Dominum. Magnæ tribulationis et afflictionis adesse festinant tempora, in quibus temporaliter et spiritualiter perplexitates et discrimina inundabunt, charitas multorum refrigescet, et superabundabit malorum iniquitas. Demonum potestas plus solito soluetur, nostræ Religionis et aliarum puritas immaculata deformabitur, in tantum quod vero Summo Pontifici, et Ecclesiæ Romanæ paucissimi ex Christianis vero corde et charitate perfecta obedient. Aliquis non canonice electus, in articulo tribulationis illius ad Papatum assumptus, multis mortem sui erroris sagacitate propinare molietur. Tunc multiplicabuntur scandala, nostra diuidetur Religio, plures ex alijs omnino

frangentur, eo quod non contradicent, sed consentient errori. Erunt opiniones et schismata tot, et tanta in populo, et in Religiosis, et in Clero, quod nisi abbreviarentur dies illi juxta verbum Euangelicum (si fieri posset) in errorem inducerentur etiam electi, nisi in tanto turbine ex immensa misericordia Dei regerentur.

Regula et vita nostra tunc a quibusdam acerrime impugnabitur. Superuenient tentationes immensæ. Qui tunc fuerint probati, accipient coronam vitæ. Væ autem illis, qui de sola spe Religionis confisi tepescent, non resistent constanter tentationibus, ad probationem electorum permissis.

Qui vero spiritu ferventes ex charitate et zelo veritatis adhæredunt pietati, tanquam inobedientes et schismatici persecutiones et iniurias sustinebunt. Nam persequentes eos a malignis spiritibus agitati, magnum esse obsequium Dei dicent, tam pestilentes homines interficere et delere de terra. Erit autem tunc refugium afflictis Dominus, et saluabit eos, quia sperauerunt in eo. Et vt suo capiti conformentur, fiducialiter agent, et per mortem, vitam mercantes æternam, obedire Deo magis quam hominibus eligent; et mortem, nolentes consentire falsitati et perfidiæ, nullatenus formidabunt. Veritas tunc a quibusdam prædicatoribus operietur silentio, ab aliis conculcata negabitur. Vitæ sanctitas a suis professoribus habebitur in derisum, quare dignum non pastorem, sed exterminatorem mittet illis Dominus Jesus Christus». [Wad23, p.480-482]

Entre os breves cometários que Wadding fez da profecia transcrita acima, Wadding mencionou que parte dela aparecia na obra de frei Marcos de Lisboa, Crônicas da Ordem dos Frades Menores, cuja primeira parte foi publicada pelo frade português em 1557 [Lis62]. Transcrevemos a seguir, em português bastante arcaico, a primeira parte da profecia presente nas crônicas de frei Marcos de Lisboa.

E esta prophecia ficou escripta per frey Leão companheiro do santo Padre, que muitas vezes lha ouuiu de sua boca. "Virão tempos em que na igreja averà grandes perplexidades e duvidas no téporal e spiritual estado, e arrefecerà a charidade de muitos e crescera a malicia e o poder do demonio serà mais solto do costumado. E nestes tẽpos serà mui desfeada a fermosura e limpeza desta religião e das outras, e serà comprida a profana dissensaõ e apostasia de ambos os imperios, tanto q̃ ao summo Pontifice e a igreja Romana, muy poucos cõ verdadeira charidade obedecerão. E o que for não canonicamẽte electo, mas sospeito de heresia, ou intruso em o Pontificado em a conjunção destes trabalhos, serà obedecido e sagazmente converterà muitos ao mortifero erro seu. Então multiplicarseão os escandalos, e serà a Chistandàde divisa, porq̃ muitos não quererão contradizer mas consentirão ao erro. E serão tantas as opinioẽs, schismas e divisoẽs em a cleresia, religiosos e povo, q̃ se Deos não abreviara aquelles dias, os escolhidos cairaõ em grandes erros, se os Deos não livrara em tanto o impeto de males, por sua misericordia." [Lis62, livro 2, capítulo 27]

Frei Marcos de Lisboa julgava que esta profecia havia se cumprido no grande cisma do Ocidente. Durante o cisma, foi escrita uma das fontes bibliográficas usadas por Marcos de Lisboa, o *De conformitate vitae beati Francisci ad vitam Domini Iesu* [Pis86] mais conhecido como o *Livro das Conformidades*, de Bartolomeu de Pisa, de onde frei Marcos deve ter compilado a profecia.

7.1.2 Bartolomeu de Pisa

Frei Bartolomeu nasceu em Pisa antes de 1338 e viveu até o começo de 1400 [Man64]. Estava programada a sua ida para a Universidade de Cambridge para obter o grau de mestre, mas foi impossível chegar até a Inglaterra por causa da Guerra dos Cem Anos. O Papa Gregório XI concedeu-lhe o título de *magíster* depois que completou as disciplinas

em Bolonha [Mas12]. Foi professor na *studia* Franciscana na sua cidade natal de Pisa, em Siena, na Florência e em Pádua. Entre os poucos elementos de informação disponíveis sobre as suas atividades, o mais significativo diz respeito a sua pregação e produção literária. Além do *Livro das Conformidades*, ele produziu uma grande quantidade de texto sobre a vida da Virgem Maria, o *De Vita et laudibus Beatae Mariae Virginis* [Pis96], que pode ser considerada uma primeira tentativa de usar a noção de "conformidade" para compara a vida de um santo (no caso, a Virgem Maria) com a vida de Cristo, um método que atingirá o seu grau mais alto no trabalho posterior sobre a vida de São Francisco. Ele também produziu duas séries de sermões que foram publicados, um dos quais (dado em 1390 na Florência) foi dedicado ao discernimento moral (*de casibus conscientiae*, e o outro (1397 em Pisa) dizia respeito aos valores mundanos (*de contemptu mundi*).

O Livro das Conformidades

O livro compara a vida de São Francisco com a vida de Cristo, usando quarenta similaridades chamadas de "conformidades" para ilustrar a semelhança do Santo de Assis com o seu Mestre [Sho15]. Por exemplo: Jesus curou os leprosos; Francisco cuidou dos leprosos. Estas conformidades são arranjadas, de forma textual e gráfica, como frutos nos galhos de uma grande árvore, belamente ilustrado no começo das edições impressas.

Além da extensa utilização dos textos das Escrituras sagradas que servem para mostrar as conformidades, Bartolomeu também utiliza uma ampla série de textos de autores clássicas e cristãos, incluindo ambas as profecias cristãs e pagãs. O texto também fornece transcrições muito fieis dos primórdios de muitos documentos Franciscanos, incluindo desde escritos do próprio Francisco até textos hagiográficos primitivos sobre ele mesmo e seus seguidores.

Bartolomeu também usou trabalhos dos franciscanos Espirituais (este era um grupo dentro da O.F.M. que pregava uma observância mais estrita dos ensinamentos de São Francisco). Uma análise cuidadosa do *Livro das Conformidades* revela que o Mestre de Pisa citou o franciscano Espiritual Ângelo Clareno, reproduzindo os comentários de Clareno sobre a *Regra* de São Francisco assim como as extensas citações à *Primeira Regra* (*Regula non bullata*). Ele também cita o teólogo e dissidente Franciscano Ubertino de Casale, muito admirado entre os Franciscanos Espírituais.

Diversos destes escritos que circulavam entre os Franciscanos Espirituais foram bastante apreciados dentro do ramo observante dos frades menores, entre os quais destaca-se São Bernardino de Sena.

7.1.3 Ângelo Clareno

Ângelo Clareno nasceu em Fossombrone ao redor do ano 1247 e morreu em Santa María d'Aspro a 15 de Junho de 1337 [Cat]. Ingressou na ordem por volta do ano 1262. Acreditando que a regra de São Francisco não estava sendo observada e interpretada de acordo com a mente e o espírito do Padre Fundador, tornou-se eremita e, junto com alguns colegas dele, formou um novo braço da ordem conhecido como *Os Clarenos*. Mediante a bula *Dominus Noster Jesus Christus* do Papa Sisto IV, os Clarenos foram reunidos ao corpo principal da ordem e colocados sob obediência ao Ministro Geral.

O grupo liderado por Clareno era pequeno e logo levantou suspeitas na Ordem dos Frades Menores, que não estavam preparados para seguir a observância extrema da regra de São Francisco adotada por Clareno. Assim, ele se tornou perseguido pela sua própria ordem e foi forçado a viajar pela Grécia, Armênia e por diferentes cidades da Itália até

que, em 1311, chegou em Avignon para responder acusações de heresia das quais foi absolvido. Em 1337 retirou-se ao eremitério de Santa Maria d'Aspro onde faleceu em Junho do mesmo ano.

Ângelo Clareno escreveu *Expositio regulae Fratrum Minorum* [Cla23] em 1323, com o principal objetivo de propor o retorno aos primórdios do Franciscanismo tendo em mente a ideia do desprendimento e da não apropriação das coisas. A discussão em torno da Regra de São Francisco visava conduzir os franciscanos contemporâneos ao uso simples (pobre) das coisas evitando também receber bens fixos, pressuposto que tem origem nos documentos legislativos da Ordem. Clareno propôs a observância pura da Regra Franciscana, tendo o Testamento do Padre Fundador como guia [Agu10].

7.1.4 Conrado de Offida

Frei Conrado nasceu em Offida no ano 1237 e faleceu em 1306 [AHS99]. Conrado foi ordenado sacerdote depois de ter abandonado os seus estudos e dedicar-se a oficios mais humildes [Clea]. Ele modelou a sua vida tendo como referência a São Francisco e foi especialmente observante da pobreza. Durante a sua longa vida religiosa usou o mesmo hábito e sempre andou descalço. Quando o Irmão Léo, colega e confessor de São Francisco, estava falecendo pediu a companhia do Frei Conrado e entregou-lhe os seus escritos. Juntou-se com Ângelo Clareno e outros colegas para formar parte dos Franciscanos Espirituais. Em 1294 obteve permissão do Papa Celestino V para separar-se do corpo principal da ordem e fundar os *Celestinos* que observava a Regra de São Francisco de Assis em toda a sua pureza.

Palavras de Conrado [Sab03] é uma obra atribuída a ele e pode ser encontrada em um manuscrito, o Codex 1/25, da biblioteca de Santo Isidoro em Roma. Esse manuscrito foi datado entre 1318 e 1350 e parece ter circulado entre as regiões de Roma, Umbria e Toscana, onde ficaram localizados vários conventos franciscanos durante o século quatorze [Nim87]. A obra também pode ser encontrada na Compilação Barcelona escrita na segunda metade do século quatorze [Nim87]. Não existe evidência sobre a data em que a obra foi escrita. Dada proximidade entre o Irmão Léo e Frei Conrado, é difícil discernir se o texto foi de fato escrito por Conrado de Offida (de fato o Irmão Léo é citado como fonte direta em alguns trechos do manuscrito) [AHS99].

7.1.5 Os textos a serem comparados

Esta seção faz uma comparação envolvendo quatro textos em latim:

1. a profecia *Magnum in Ecclesia schisma et tribulationem futuram* elencada por Luke *Wadding* entre os escritos de Francisco de Assis em *Beati Patris Francisci Assiciatis Opuscula*, 1623 [Wad23, p.480-482];

2. um trecho do primeiro capítulo *Expositio regulae Fratrum Minorum* [Cla23, p.44-48] de *Angelo* Clareno, que se põe a explicar o primeiro capítulo da Regra franciscana;

3. o décimo oitavo fruto descrito por *Bartolomeu* de Pisa no livro II de *De conformitate vitae beati Francisci ad vitam Domini Iesu* [Pis86, p.166-168], que apresenta a profecia de um cisma feita por São Francisco de Assis (quando o livro foi aprovado pelo capítulo geral da ordem franciscana em 1396, faziam já 18 anos que tinha se iniciado o Grande Cisma do Ocidente, com uma linha de papas em Avignon se contrapondo à de Roma);

4. o texto completo de *Verba Conradi* [Sab03, p.370-391], cujos capítulos 10-12 contêm a região de interesse.

Os referidos textos serão aqui chamados simplesmente de *Wadding*, *Angelo*, *Bartolomeu*, *Conradi*, e os documentos correspondentes serão representados respectivamente pelas iniciais: W, A, B, ou C. O documento W, nossa sequência de referência, começa com as 294 palavras do texto da profecia *Magnum in Ecclesia schisma et tribulationem futuram* visto acima. Em sua parte final estão acrescidos também os termos presentes na breve seção final com algumas referências e comentários acrescentados por Wadding em sua primeira edição de 1623. Em contrapartida, os documentos secundários A, B e C possuem 775, 655 e 3576 palavras, respectivamente. O segmento $C[2238, 3354]$ do documento C corresponde aos capítulos 10, 11 e 12 de `Conradi`. Enquanto os textos *Wadding*, *Angelo*, *Bartolomeu* foram manualmente verificados com as imagens das páginas originais, o texto *Conradi* não sofreu este processo de verificação, apresentando vários erros de OCR, ainda que as notas de rodapé tenham sido eliminadas e as palavras que sofreram separação silábica tenham sido novamente reunificadas.

A seguir mostramos, de forma sintética, os resultados envolvendo um *alinhamento múltiplo destas quatro sequências, tendo W como sequência de referência*, para o conjunto paramétrico: $k = 4$, $t_{igual} = 3$, $t_{similar} = 1$, $t_{dspar} = -1$, $t_{\tilde{e}} = -2$, queda máxima $X = 10$, pontuação mínima de um PAP 10 e taxa de erro $\beta = 30\%$.

Os alinhamentos locais do documento de referência W contra os documentos secundários A, B e C produzidos por BLASTD são 8, 7 e 7, respectivamente. Os segmentos envolvidos em cada alinhamento local estão graficamente representados por barras horizontais pretas na figura 7.2. Os alinhamentos locais de W contra A, B e C são representados pelas arestas que ligam cada par de segmentos, um de W, outro de um documento secundário: A, B ou C.

Cada aresta de número n ligando W a X (onde X é A, B ou C) na figura 7.2 representa o alinhamento WXn, de forma que os 22 alinhamentos são nomeados como: $WA1, \ldots, WA8, WB1, \ldots, WB7, WC1, \ldots, WC7$. A enumeração dos alinhamentos WXn segue uma ordenação baseada em pontuações decrescentes, de forma que o alinhamento $WA1$ é o alinhamento local de mais alta pontuação entre W e A.

Lembramos que o primeiro termo de um documento é o de índice 0, e cada segmento envolvido em um alinhamento local é definido por um intervalo de índices no documento em questão, em W, ou no documento secundário. A tabela 7.1 relata estes intervalos de índices para cada um dos 22 alinhamentos locais. Os alinhamentos estão ordenados segundo as ocorrências dos segmentos no documento de referência W. Para segmentos da sequência W equivalentes, adota-se uma ordenação segundo pontuações decrescentes. Além dos intervalos e das pontuações, são relatadas também a quantidades de pares de símbolos iguais, similares, díspares, ou gaps que compõem o alinhamento. As linhas horizontais na tabela separam grupos de alinhamentos com segmentos equivalentes no documento de referência W, e os negritos destacam os alinhamentos de maior pontuação em cada grupo.

Como o texto *Conradi* não sofreu uma verificação com as imagens de forma a eliminar os erros de OCR, mas as sequências *Angelo* e *Bartolomeu* sim, é de se esperar que diversas colunas do alinhamento múltiplo o termo do documento W se alinhe com similaridade diante do correspondente termo do documento secundário C mas se alinhe com igualdade diante dos correspondentes termos dos documentos secundários A e B. Pode-se observar este fenômeno várias vezes, por exemplo, no alinhamento múltiplo que envolve os alinhamentos locais $WA1$, $WB1$ e $WC1$. Isto implica que grupos de alinhamentos equivalentes pelo segmento de referência devem ter a pontuações alteradas caso

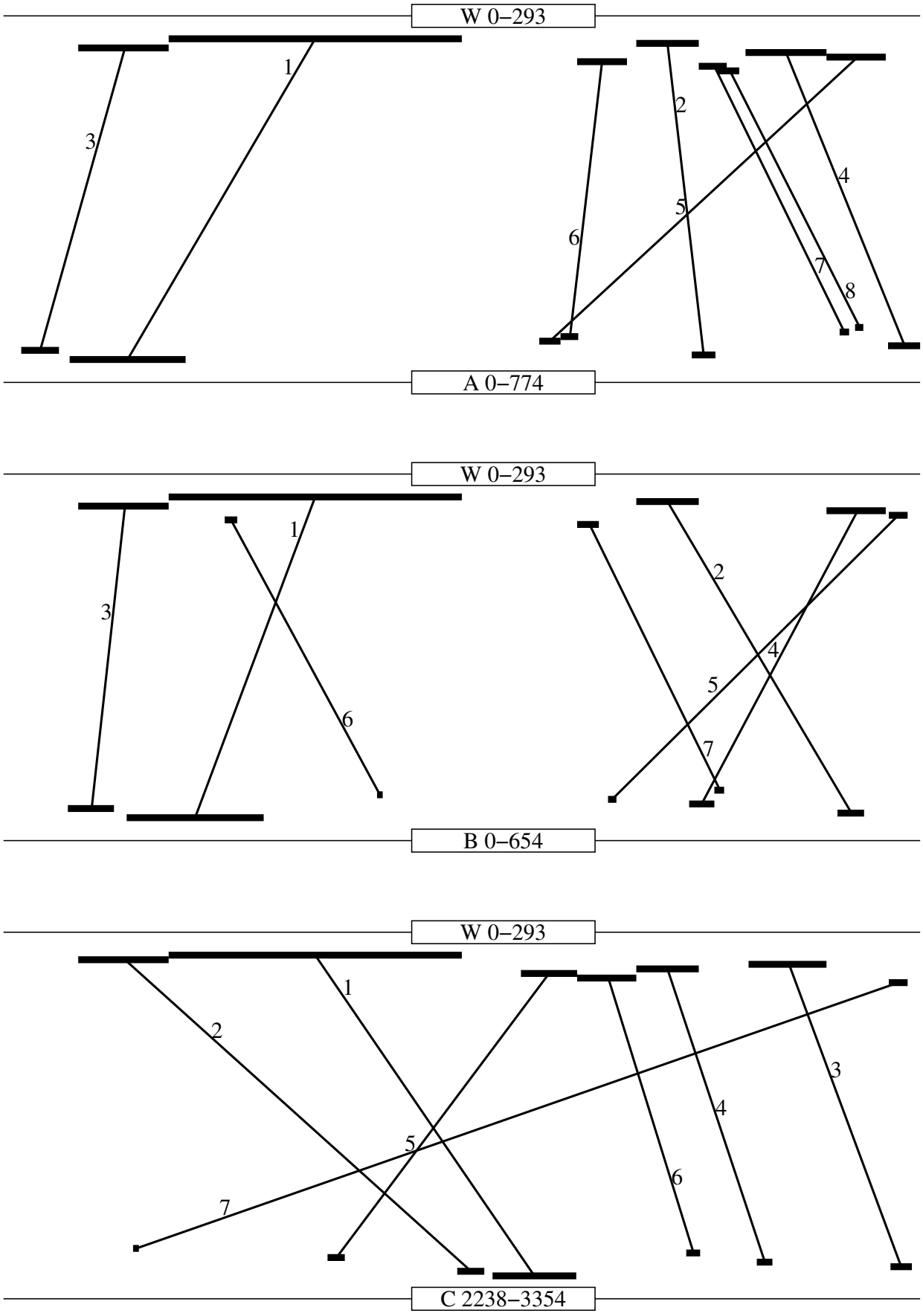


Figura 7.2: *Os 22 alinhamentos locais de W contra A, B e C são representados pelas 22 arestas, cada uma ligando dois segmentos representados por barras horizontais pretas: um de W; outro de um documento secundário A, B ou C.*

Alinhamento	interv. W		interv. sec.		pontua.	igual	similar	dispar	gap
WA3	24	52	15	46	**56**	20	7	1	5
WC2	24	52	2791	2823	52	19	8	1	6
WB3	24	52	46	78	46	18	7	3	6
WB1	53	146	88	185	**178**	67	15	6	16
WA1	53	146	56	153	156	63	15	8	20
WC1	53	146	2834	2935	141	61	14	12	22
WB6	71	74	267	270	12	4	0	0	0
WC5	166	183	2633	2653	**32**	13	2	3	3
WC6	184	202	3070	3086	**23**	11	3	1	6
WA6	184	199	471	485	19	8	5	0	5
WB7	184	190	508	514	11	4	1	2	0
WA2	203	222	582	601	**60**	20	0	0	0
WB2	203	222	596	614	51	18	0	1	1
WC4	203	222	3156	3174	42	16	1	1	3
WA7	223	231	707	714	**16**	6	1	1	1
WA8	229	235	720	726	15	5	1	1	0
WA4	238	263	748	774	**55**	20	3	2	3
WC3	239	263	3319	3344	45	18	3	2	5
WA5	264	282	453	470	**35**	13	3	1	3
WB4	264	282	490	507	33	13	2	2	3
WB5	284	289	432	437	**16**	5	1	0	0
WC7	284	289	2396	2402	14	5	1	0	1

Tabela 7.1: *Os 22 alinhamentos locais de W contra A, B e C, ordenados pelas ocorrências dos segmentos no documento de referência W e, então, por ordem decrescente de pontuação.*

seja feito este processo de eliminação de erros de OCR em *Conradi*.

Estes mesmos 22 alinhamentos locais estão sinteticamente representados na figura 7.3, onde os segmentos de maior pontuação nas suas classes de equivalência são destacados com fundo laranja.

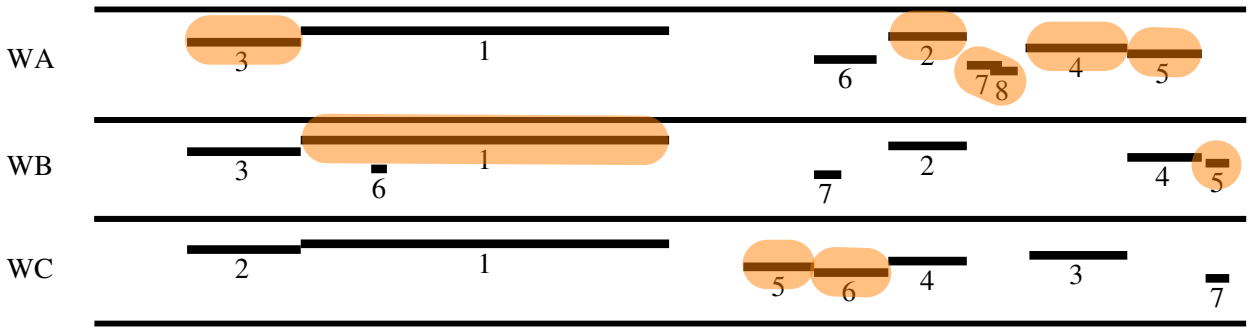


Figura 7.3: *Os 22 alinhamentos locais de W contra A, B e C são representados pelos 22 segmentos de W representados por barras horizontais pretas. Os destaques em laranja são aplicados aos segmentos de maior pontuação em sua classe de equivalência, que no entanto pode vir a ser revista depois de uma eliminação de erros de OCR do documento C.*

Da informação presente na figura 7.3 e seus 22 alinhamentos, pode-se observar que:

1. O segmento $W[166,183]$ é coberto unicamente pelo alinhamento $WC5$. São 18 as palavras que o compõe: "væ autem illis qui de sola spe religionis confisi tepescent non resistent constanter tentationibus ad probationem electorum permissis". Em relação aos termos correspondentes de C pelo alinhamento local, das dezoito palavras: três se alinham com disparidade; duas se alinham com similaridade (uma por conta de um erro de OCR); e treze se alinham com igualdade.

2. O segmento $W[223,235]$ é coberto unicamente por alinhamento contra o documento A, combinando-se os alinhamentos $WA7$ e $WA8$. São 13 as palavras que o compõe: "erit autem tunc refugium afflictis dominus et saluabit eos quia sperauerunt in eo".

3. Nenhum outro segmento de W é coberto unicamente por alinhamento contra um único documento secundário. Em particular, nenhum alinhamento local contra B cobre com exclusividade um segmento de W.

4. Os três segmentos $W[0,23]$, $W[147,165]$ e $W[290,293]$, respectivamente com 24, 18 e 4 palavras, são os únicos segmentos maximais de comprimento ao menos $k = 4$ que não são cobertos pelos 22 alinhamentos locais listados por BLASTD com os parâmetros acima. Os trechos em questão são respectivamente: "pavlo ante mortem conuocatis fratribus de tribulationibus futuris eos admonuit dicens viriliter agite fratres confortemini et sustinete dominum magnæ tribulationis et afflictionis adesse festinant", "regula et vita nostra tunc a quibusdam acerrime impugnabitur superuenient tentationes immensæ qui tunc fuerint probati accipient coronam vitæ", e "illis dominus iesus christus".

Um alinhamento múltiplo entre A, B e C pode trazer uma informação complementar que ajude a elucidar melhor as origens de *Wadding*. Até porque nenhum alinhamento local contra B cobre com exclusividade um segmento de W, surge a indagação a respeito de quanto do texto *Bartolomeu* é composto de citações dos textos mais antigos *Angelo* e

Conradi. Assim, convém analisar o *alinhamento múltiplo entre os três documentos associados, A, B e C, tomando B por referência.*

Bartolomeu de Pisa escreve um preâmbulo próprio de 42 palavras para apresentar a profecia feita por São Francisco de Assis, seguido de cópias de trechos das profecias propriamente ditas, tiradas de *Angelo* ou *Conradi*, e apresenta uma conclusão própria de 40 palavras, onde sinteticamente também descreve como as visões futuras associadas a esta profecia justificaram detalhes da redação do primeiro capítulo da Regra da ordem franciscana, comunicada por Cristo a São Francisco e por este ao frei Leão, seu redator. Entre as referências da obra de Bartolomeu de Pisa, consta a *Expositio regulae Fratrum Minorum* de Angelo Clareno. De fato, o texto *Angelo* explica de forma muito mais simples a gênese de *Bartolomeu* que *Conradi*, como fica claro a quem observa o resumo gráfico dos alinhamentos locais da sequência de referência *Bartolomeu* contra sequências secundárias *Angelo* e *Conradi* exposto na figura 7.4.

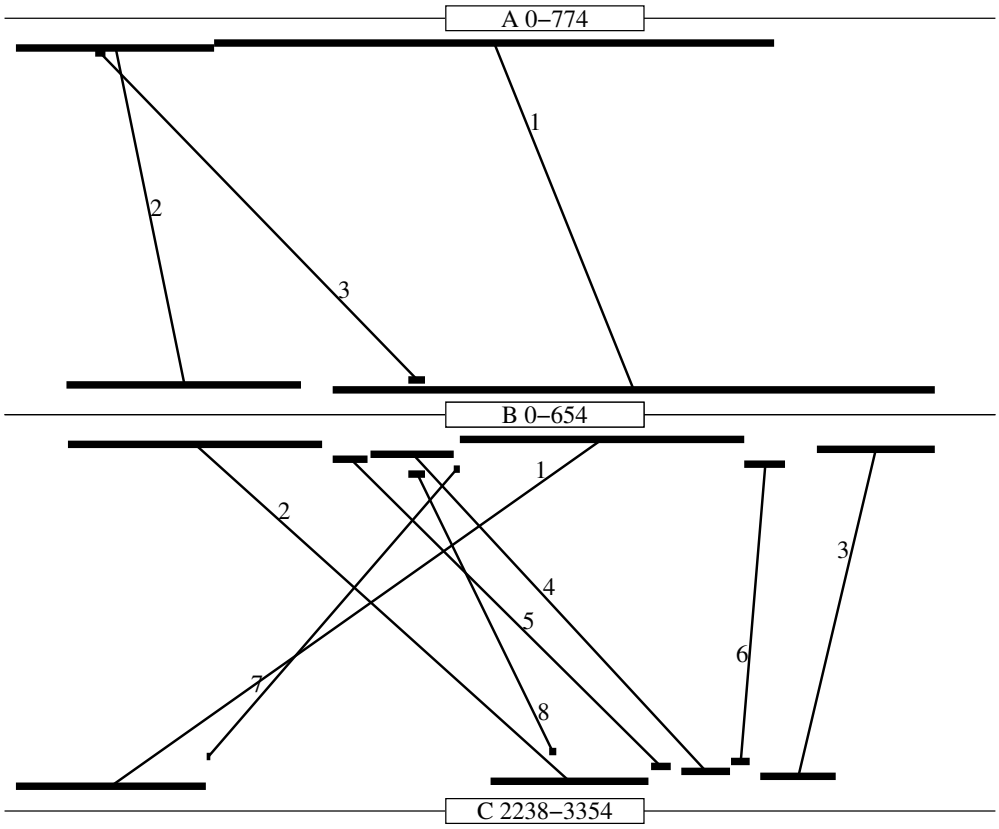


Figura 7.4: *Alinhamento múltiplo entre A, B e C, tomando B por referência. O documento A explica de forma muito mais simples a gênese de B que C explica.*

O alinhamento $AB1$ cobre em B tudo o que a combinação dos alinhamentos $BC5$, $BC4$, $BC1$, $BC6$ e $BC3$ cobre, mas esta combinação não cobre o segmento $B[516, 536]$, "persecutiones innumeras sustinebunt praedicebat etiam beatus franciscus sicut sotii retulerunt videlicet fratres bernardus massaeus leo et ceteri socii testabantur quod tunc", que é coberto por $AB1$. Com os dois alinhamentos locais $AB1$ e $AB2$ é possível cobrir praticamente todo o segmento de B obtido ao se eliminar o preâmbulo e a conclusão mencionados. Contudo, o segmento $B[196, 209]$, "regulae franciscus promittit obedientiam et reverentiam domino papae honorio et successoribus eius canonice intratibus", é coberto pelo alinhamento $BC2$ mas não pelo alinhamento $AB2$, o que leva a indagar se a gênese de B não se dá pela combinação de citações de C e A e pela junção dos trechos envolvidos nos alinhamentos $BC2$ e $AB1$. Neste contexto, parece-nos natural também supor a existência de A',

uma variante de A que apresenta este trecho faltante, da qual B descenderia unicamente. É natural, neste contexto, que nos perguntemos sobre a comparação entre A e C.

Na figura 7.5 vemos um resumo gráfico dos 6 *alinhamentos locais entre os dois documentos* A e C. Somente quatro segmentos de A não são cobertos por alinhamentos locais

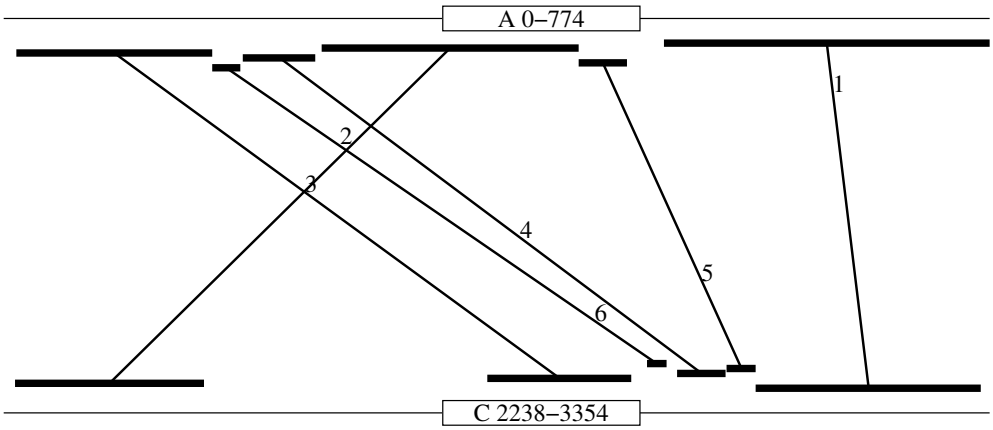


Figura 7.5: *Os* 6 *alinhamentos locais de A contra C são representados pelas* 6 *arestas, cada uma ligando dois segmentos representados por barras horizontais pretas. O trecho em C do alinhamento local AC*3 *encontra-se no cap. 10 enquanto que os trechos envolvidos nos demais alinhamentos locais pertencem ao cap. 12 de Verba Conradi.*

contra C: a introdução feita pelo segmento $A[0,9]$, "preterea sicut sotii sui referebant et fr leo scribit future", com 10 palavras; o segmento $A[186,187]$, "et venturi", com 2 palavras; o segmento $A[245,249]$, "dicebat enim sanctus franciscus et", com 5 palavras; e o segmento $A[490,518]$, "sanctus franciscus sicut sotii sui s fr bernardus et fr angelus et fr masseus et fr leo et ceteri sui sotii post ipsius transitum ad dominum testabantur quod tunc", com 29 palavras. Os três últimos segmentos participam do alinhamento $AB1$, e um alinhamento múltiplo de A, B e C que tem A por referência bem revela como eles alinham-se bem com os respectivos segmentos de B, o que reforça a prova de que o segmento de B que participa de $AB1$ foi copiado de A. Ademais, uma *busca aproximada* (com $\beta = 25\%$) pelo termo raro "masseus"[1] revela que os únicos documentos que o possuem são A e B, mas não C, o mesmo se dando ao se procurar pelo termo "bernardus".[2]

Como visto antes, o segmento $C[2238,3354]$ do documento C, com 1117 palavras, contém apenas os termos de *Conradi* que são correspondentes aos capítulos 10, 11 e 12. Destes três capítulos, os segmentos de C não cobertos por alinhamentos locais contra A são bem mais extensos: $C[2238,3250]$, com 13 palavras; $C[2465,2785]$, com 321 palavras; $C[2949,2966]$, com 18 palavras; $C[2989,3000]$, com 12 palavras; $C[3056,3056]$, com 1 palavra; $C[3345,3354]$, com 10 palavras. Dentro do segmento mais extenso, $C[2465,2785]$, encontra-se todo o capítulo 11, onde se relata uma aparição de Cristo a São Francisco de Assis e o diálogo que se dá entre ambos. Se devido à sua extensão não expomos aqui suas 321 palavras, expomos as 18 palavras do segmento $C[2949,2966]$, "regulæ frater franciscus promittit obedientiam et reverentiam domino papæ honorio ac successoribus ejus

[1] As palavras procuradas na busca expandida foram "masseus", "massiieus", "massaens", "massaeum", "massacus", "massaeus", "massneus", "massneum".

[2] As palavras procuradas na busca expandida foram "bernardus", "bernaruo", "gerardus", "jbernardinus", "bernardi", "bernarde", "berunrdus", "kernardus", "bcrnardum", "bcrnardus", "fernandus", "ijernardus", "berntirdus", "bernarrlus", "bernard", "rernardus", "liernardus", "bernardiis", "bemardus", "bernardum", "lïernardus", "bernardo", "leonardus", "berardus", "bernnruus", "burnardus", "bernardinus", "bernardnm", "bernarduni", "cerardus", "dernardus", "bertiardus", "bernaidus", "bernardîts", "bcruardus", "fsernardus", "ëcrnardus", "beruardus", "ternarius", "tîernardus".

canonice intrantibus il voluit inquam", e as 12 de $C[2989, 3000]$, "in hoc scilicet quod addidit canonice intrantibus 12[3] et in hoc voluit". Naturalmente que a existência destes trechos tão complexos servem para demonstrar que *Conradi* não foi historicamente gerado a partir de *Angelo*, mas muito mais provavelmente o contrário. Contudo, a demonstração de que *Angelo* descende de *Conradi* (ou uma variante próxima) é mais facilmente observada ao se debruçar sobre o conteúdo dos dois textos, mas também pelo contexto mais amplo do capítulo I da *Expositio regulae Fratrum Minorum* de Angelo Clareno, cuja finalidade é apresentar a Regra aos seguidores franciscanos. Se o segmento $A[0, 9]$ introduz as palavras de São Francisco com a sentença "preterea sicut sotii sui referebant et fr leo scribit future", imediatamente antes precede-o o seguinte texto no livro de Clareno: "Ex qua re signanter dicitur, quod domino pape Honorio et successoribus eius canonice intrantibus et ecclesie Romane promictit obedientiam et reverentiam, quam ipse tunc promisit pape Honorio et pontificibus sui temporis et ecclesie, intelliguntur promisisse."

Voltando agora à gênese do documento W, tal como resumidamente vimos na figura 7.3, o alinhamento $WC5$ cobre um segmento de W que não é coberto nem por B nem por A; a saber: $W[166, 183]$, "væ autem illis qui de sola spe religionis confisi tepescent non resistent constanter tentationibus ad probationem electorum permissis ", com 18 palavras. De fato, participa deste alinhamento local um trecho do último versículo do capítulo 11 de *Conradi*, que não se alinha com A, como vimos na figura 7.5.

Por outro lado, parece contradizer a hipótese de que A se origina de C o fato de que o segmento $W[264, 282]$, "veritas tunc a quibusdam prædicatoribus operietur silentio ab aliis conculcata negabitur vitæ sanctitas a suis professoribus habebitur in derisum", se alinhe com A pelo alinhamento local $WA5$ mas não se tenha encontrado nenhum alinhamento com C. Parte do alinhamento que falta é recuperado quando se alinha W com C reduzindo o tamanho de um k-grama de 4 para 3, como se pode ver na figura 7.6.

veritas 264	tunc 265	a 266	quibusdam 267	prædicatoribus 268	operietur 269	silentio 270	ab 271	aliis 272	conculcata 273	negabitur 274
veritas 3058	tune 3059	a 3060	--	prædicatoribus 3061	operietur 3062	silentio 3063	--	aut 3064	conculcata 3065	negabitur 3066
3	1	3	-2	3	3	3	-2	-1	3	3

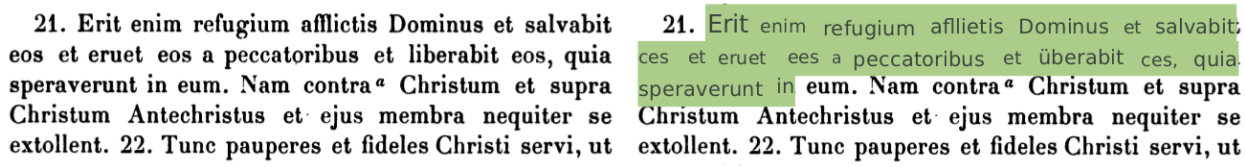
erit 223	autem 224	tunc 225	refugium 226	afflictis 227	dominus 228	et 229	saluabit 230
erit 3277	enim 3278	--	refugium 3279	afflietis 3280	dominus 3281	et 3282	salvabit 3283
3	-1	-2	3	-1	3	3	1

et 229	saluabit 230	eos 231	quia 232	sperauerunt 233	in 234
et 3290	überabit 3291	ces 3292	quia 3293	speraverunt 3294	in 3295
3	-1	-1	3	1	3

Figura 7.6: *Com k reduzido para* 3, *um novo alinhamento de W com C cobre parte dos segmentos de W cobertos pelos alinhamentos locais WA*5, *WA7 e WA*8. *As linhas superiores correspondem a W e as inferiores a C.*

A figura também exibe a recuperação de alinhamentos que cobrem as regiões de W cobertas pelos alinhamentos $AL7$ e $AL8$. No caso destes dois alinhamentos, é possível verificar que erros de OCR existentes em C impedem o reconhecimento destes alinhamentos quando $k = 4$. A figura 7.7 exibe a imagem do versículo 21 do capítulo 10 de *Conradi*, bem como o texto reconhecido pelo OCR, destacado em verde. Palavras como "afflictis", "liberabit" e "eos" foram reconhecidas incorretamente. Através da visualiza-

[3] Este número refere-se à numeração do versículo.



21. Erit enim refugium afflictis Dominus et salvabit eos et eruet eos a peccatoribus et liberabit eos, quia speraverunt in eum. Nam contra[a] Christum et supra Christum Antechristus et ejus membra nequiter se extollent. 22. Tunc pauperes et fideles Christi servi, ut

21. Erit enim refugium afflictis Dominus et salvabit eos et eruet eos a peccatoribus et liberabit eos, quia speraverunt in eum. Nam contra[a] Christum et supra Christum Antechristus et ejus membra nequiter se extollent. 22. Tunc pauperes et fideles Christi servi, ut

Figura 7.7: *Imagem do versículo 21 do capítulo 10 de* Conradi *[Sab03, p.390]. Palavras como "afflictis", "liberabit" e "eos" estão incorretamente reconhecidas pelo OCR da biblioteca digital, cuja saída está destacada em verde.*

ção do alinhamento múltiplo de três textos, a figura 7.8 exibe a região das 5 ocorrências da locução "et saluabit [eos]": uma em *W*; duas em *C*; e duas em *A*.

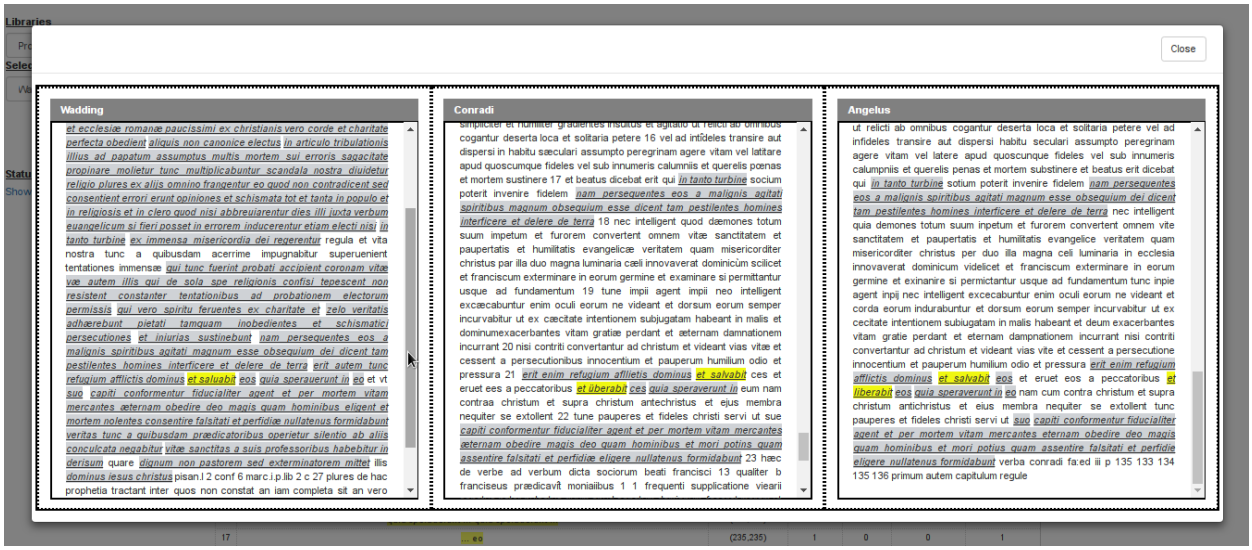


Figura 7.8: *Alinhamento múltiplo de W, C e A em torno das ocorrências da locução "et saluabit [eos]".*

7.2 Detecção de Plágio

Nesta seção mostramos a aplicação dos algoritmos de alinhamento local ao problema de detecção de plágio. Dados dois documentos de texto, o documento original e o documento suspeito, o problema de detecção de plágio consiste em determinar se o documento suspeito possui trechos do documento original.

Todo ano, desde 2010, é organizada a competição PAN (abreviatura para *Uncovering Plagiarism Authorship and Social Software Misuse*) dentro da conferência CLEF (Conference and Labs of the Evaluation Forum). O PAN consiste em três tarefas: detecção de plágio, verificação de autoria, e identificação do perfil do autor. A tarefa de detecção de plágio é dividida em duas sub-tarefas: recuperação da fonte; e alinhamento de texto. Na sub-tarefa de alinhamento de texto, os algoritmos precisam identificar trechos contíguos de comprimento máximo de texto reutilizado em um par de documentos. Para tanto, é fornecido junto à tarefa um corpus de trabalho. Examinaremos agora o corpus do PAN 2013.

7.2.1 Corpus

O corpus é composto de pares de documentos escolhidos entre 145 tópicos que foram processados de forma manual e automática de forma que os pares de documentos podem pertencer a cinco categorias: sem plágio, sem ofuscação, ofuscação aleatória, ofuscação cíclica e ofuscação por sumarização [FFN+].

Os pares de documentos sem plágio não compartilham trechos em comum, enquanto que a categoria sem ofuscação apresenta casos de plágio do tipo copiar e colar, puro e simples. Na ofuscação aleatória também são introduzidas uma quantidade de operações de texto aleatórias: embaralhamento de palavras, inserção ou remoção de palavras e frases, e substituição de palavras por sinônimos. A ofuscação cíclica produz textos em inglês que foram traduzidos em outras duas línguas para produzir o texto final de volta na língua inglesa e esta fora do nosso escopo. Finalmente, a ofuscação por sumarização possui documentos obtidos do corpus da Document Understanding Conference (2006) e requerem o uso de técnicas de sumarização fora do nosso escopo.

Pares da categoria sem plágio foram usadas para balizar nossa detecção. As categorias de ofuscação por sumarização e ofuscação por tradução cíclica não foram atacadas em nossos experimentos.

Ao todo, o corpus de teste do PAN 2013 possui 5.185 pares de documentos produzidos a partir de 3.169 documentos originais e 1.826 documentos suspeitos. Algumas estatísticas do corpus são mostradas na tabela 7.2.

	Caracteres	Palavras
Documentos suspeitos		
Comprimento mín.	657	110
Comprimento máx.	101483	16563
Comprimento médio	8711	1421
Documentos originais		
Comprimento mín.	529	20
Comprimento máx.	518181	91991
Comprimento médio	4487	727

Tabela 7.2: *Estatísticas do corpus de alinhamento de texto PAN 2013*

7.2.2 Medidas de desempenho

As medidas de desempenho para avaliar o resultado dos algoritmos participantes no PAN 2013 são baseadas na precisão e cobertura adaptadas ao problema da detecção de plágio. Esta adaptação permite detectar quão bem o alinhamento detectado corresponde às posições dos caracteres e comprimentos dos trechos dos valores de referência para ambos os documentos original e suspeito.

As medidas envolvem *casos* (reais) e *detecções* (relatos) de plágio que são definidos da seguinte maneira. Um *caso* de plágio é uma tupla $p = (r_{off}, r_{len}, s_{off}, s_{len})$ onde r_{off} e r_{len} são o começo e comprimento do trecho no documento original r compartilhado com o documento suspeito s; s_{off} e s_{len} são o começo e comprimento do trecho do documento suspeito. De forma análoga, usamos a tupla $q = (r'_{off}, r'_{len}, s'_{off}, s'_{len})$ para definir uma *detecção* de plágio encontrada pelo algoritmo de alinhamento.

Dadas duas tuplas p e q, onde p é um caso de plágio e q é uma detecção de plágio, dizemos que o caso de plágio p foi detectado por q se os intervalos de p e q se intersectam

mutuamente, tanto os intervalos relativos ao documento original quanto os intervalos relativos ao documento suspeito.

Se definirmos P como o conjunto de todos os casos de plágio e Q o conjunto das detecções de plágio, então a precisão e cobertura são definidas da seguinte maneira:

$$precision(P,Q) = \frac{1}{|Q|} \sum_{q \in Q} \frac{\bigcup (p \sqcap q)}{|q|}$$

e

$$recall(P,Q) = \frac{1}{|P|} \sum_{p \in P} \frac{\bigcup (p \sqcap q)}{|p|}$$

onde

$$p \sqcap q = \begin{cases} p \cap q, & \text{se } q \text{ detecta } p \\ 0, & \text{caso contrário} \end{cases} \tag{7.1}$$

Para o cálculo da intersecção $p \cap q$, p (ou q) é interpretado como união dos dois intervalos posicionados que o constituem.

De forma a reconhecer se um caso de plágio é detectado como um todo por uma única detecção ou em pedaços por diversas detecções que se complementam, é introduzida a medida de *granularidade* de detecções, é definida da seguinte forma:

$$gran(P,Q) = \frac{1}{|P_Q|} \sum_{p \in P_Q} |Q_p|,$$

onde $P_Q \subseteq P$ é o conjunto dos casos detectados pelas detecções P, e $Q_p \subseteq Q$ é o conjunto de detecções relatadas que detectam o caso p.

Finalmente, todas as medidas definidas acima são combinadas em uma medida composta denominada como *pontuação de detecção de plágio*:

$$plagdet(P,Q) = \frac{F_1}{log_2(1 + gran(P,Q))},$$

onde F_1 é a media harmônica da precisão e da cobertura entre P e Q.

7.2.3 Resultados

Adaptamos o algoritmo de alinhamento local BLASTD proposto no capítulo 5 para contemplar problemas de plágio. A adaptação ocorreu de duas formas. Por um lado, foi relaxado o critério de queda máxima para espaços e disparidades, isto é, o valor de um espaço ou uma disparidade pode ser maior ou igual à queda máxima. Isto para forçar o algoritmo BLASTD gerar alinhamentos locais que não contêm espaços ou disparidades. Por outro lado, foi introduzido um filtro que verifica a projeção dos alinhamentos locais no documento suspeito. O filtro remove os alinhamentos locais que são contidos em outro alinhamento local.

Foi criado um processo em lote para submeter todos os documentos originais e suspeitos no sistema. Em seguida foram criadas bibliotecas de documentos, uma para cada par de documentos anotados no conjunto de treinamento do corpus. De acordo com a definição de detecções de plágio mostrada na subseção anterior, é necessário relatar estes

levando em conta inclusive símbolos de pontuação ou espaços em branco no documento. Estes símbolos são removidos do documento quando este é indexado no sistema. Assim, é necessário corrigir as posições relatadas por um alinhamento local para refletir as verdadeiras posições em ambos os documentos original e suspeito.

Por outro lado, para cada categoria do conjunto de dados, foi utilizado um conjunto paramétrico específico, tem por objetivo ajustar o comportamento do algoritmo BLASTD aplicado a cada categoria.

Categoria	k-grama	t_{igual}	$t_{similar}$	t_{dispar}	$t_{\bar{\epsilon}}$	X	Pont. min.	β
Sem ofuscação	11	2	1	-30	-30	10	21	5%
Ofuscação aleatória	4	3	1	-1	-2	14	10	40%

Tabela 7.3: *Conjuntos paramétricos utilizados nas categorias da competição PAN 2013 usados em nossos experimentos.*

O parâmetro de disparidade e espaço são altos na categoria sem ofuscação pois desejamos obter alinhamentos locais sem colunas deste tipo. Com respeito aos parâmetros da categoria "sem ofuscação", estes foram projetados para forçar o BLASTD parar a execução quando encontra uma coluna com disparidade ou espaço. Na categoria "ofuscação aleatória", os parâmetros foram ajustados para readmitir colunas deste tipo. Ambos os fenômenos são controlados via as pontuações de disparidade e de espaço (t_{dispar} e $t_{\bar{\epsilon}}$).

Antes de mostrar os nossos resultados, desejamos atrair a atenção do leitor para o resultado de um algoritmo bastante similare ao nosso. Dentro dos algoritmos que usaram os dados do PAN 2013, os trabalho de Glinos [Gli14] é de nosso interesse.

O algoritmo de Ginos implementa Smith-Waterman (De forma a achar soluções subótimas, o algoritmo implementa também uma heurística que recursivamente explora o espaço de soluções. A exploração começa pela localização de uma solução subótima na matriz de programação dinâmica. Essa localização, que tem a forma de um retângulo que cobre ambos os documentos, cria quatro regiões na matriz que não cobrem nenhum dos dois documentos). O desempenho deste algoritmo é superior a todas as soluções submetidas durante a competição PAN 2013 e os resultados nas categorias de nosso interesse estão mostrados na tabela 7.4.

Categoria	PlagDet	Cobertura	Precisão	F_1	Granularidade	Tempo
Sem ofuscação	0.962	0.960	0.964	0.962	1.0	6:47.717
Ofuscação aleatória	0.795	0.707	0.973	0.819	1.041	5:09.562

Tabela 7.4: *Resultados do algoritmo de Glinos para o conjunto de teste do PAN 2013 (adaptado de [Gli14]) (O tempo é medido em minutos)*

Com respeito aos nossos resultados, na categoria "sem plágio" nenhuma detecção de plágio é encontrada, como esperado. Na categoria "sem ofuscação", a execução do BLASTD conseguiu obter cobertura de 99,7% e precisão de 96,8%. Os documentos suspeitos que compõem este conjunto de dados somente possuem inserções de trechos do documento original, sem inserção de palavras ou de sinônimos.

O nosso resultado na categoria "sem ofuscação" supera todas as medidas relatados por Glinos com exceção da granularidade. Com respeito ao tempo de execução, o nosso algoritmo rodou em 1:40 minutos superando o tempo de execução de Glinos que rodou em 6:47 minutos.

Categoria	PlagDet	Cobertura	Precisão	F_1	Granularidade	Tempo
Sem ofuscação	0.988	0.997	0.979	0.988	1.0	1:27.74
Ofuscação aleatória	0.572	0.816	0.813	0.811	1.678	1:32.80

Tabela 7.5: *Resultados da detecção de plágio do BLASTD para o conjunto de teste do PAN 2013*

Com respeito à categoria "ofuscação aleatória", são introduzidas substituições de palavras utilizando dicionário de sinônimos e são permitidas inversões, remoções e inserções de palavras. Os parâmetros que foram configurados tentam superar essa dificuldade incrementando a taxa de erro e permitindo introduzir espaços nos alinhamentos locais. Isto tem o efeito de aumentar a cobertura (inclusive superando o resultado relatado por Glinos). Devido à limitação de tempo, a introdução de um dicionário de sinônimos não foi feita nem tampouco uma estratégia para aglutinação de alinhamentos locais necessária para diminuir a granularidade de 1.678. Esta é a principal razão pela qual nosso algoritmo teve desempenho final inferior ao de Glinos.

7.2.4 Outras técnicas de detecção de plágio

O problema de detectar plágio é bastante comúm no domínio da engenharia de software para determinar a similaridade entre dois programas de computador. Os algoritmos mais conhecidos nesse tipo de aplicação são o Winnowing [SWA03] e Greedy String Tiling (GST) [Wis93].

A técnica de Winnowing produz para uma sequência de entrada os k-gramas de símbolos sem levar em conta os espaços. Em seguida, uma função de espalhamento atribui chaves para cada k-grama obtendo assim uma sequência de chaves. Essa sequência é convertida em segmentos de chaves de tamanho w e logo a chave de menor valor é selecionada em cada segmento. O algoritmo descarta as chaves repetidas e contíguas e a sequência de final de chaves configura uma espécie de digital de um documento de texto. O único trabalho que encontramos que chega a detalhar os algoritmos a ponto de descrever como é que detectada a similartidade de dois documentos a partir de suas digitais, fala em calcular a intersecção delas interpretadas como conjuntos de chaves mas aparentemente ignora eventuais repetições das chaves que as formam [WSR+17].

Com respeito ao algoritmo GST, este localiza segmentos de comprimento maximal que ocorrem em ambos documentos (o comprimento mínimo é um parâmetro do algoritmo). Uma vez que um segmento maximal é localizado os tokens que o conformam são marcados para não serem utilizados por segmentos de comprimento menor. O algoritmo termina quando não encontra mais segmentos de tamanho maximal e a as posições dos segmentos junto com os seus cumprimentos são retornadas pelo algoritmo. As posições e cumprimentos encontrados são denominados "tiles". Os tiles correspondem com alinhamentos entre os dois documentos.

O algoritmo de Winnowing é utilizado pela ferramenta MOSS para detectar plágio entre programas de computador. Esta ferramenta se encontra disponível como um serviço web e para utiliza-lo é necessário pedir o registro no serviço. Fizemos o pedido mas não tivemos nenhuma resposta.

Com respeito ao algoritmo GST, utilizamos a ferramenta JPlag que implementa o algoritmo GST com algumas otimizações de performance. Fizemos um pequeno teste desta ferramenta para ter uma noção melhor das suas capacidades na detecção de plágio de texto. A figura 7.9 mostra o resumo do JPlag para um caso de plágio sem obfuscação do conjunto de dados do PAN 2013. JPlag relatou 5 alinhamentos sendo que o maior,

mostrado na figura 7.10, contém 60 tokens.

suspicious-document00005.txt (8.171206%)	source-document01090.txt (14.651163%)	Tokens
suspicious-document00005.txt(30-31)	source-document01090.txt(1-1)	27
suspicious-document00005.txt(31-32)	source-document01090.txt(8-9)	23
suspicious-document00005.txt(32-35)	source-document01090.txt(9-9)	60
suspicious-document00005.txt(75-76)	source-document01090.txt(23-23)	10
suspicious-document00005.txt(79-79)	source-document01090.txt(24-24)	6

Figura 7.9: *Resumo dos alinhamentos encontrados pelo JPlag para um caso de detecção de plágio.*

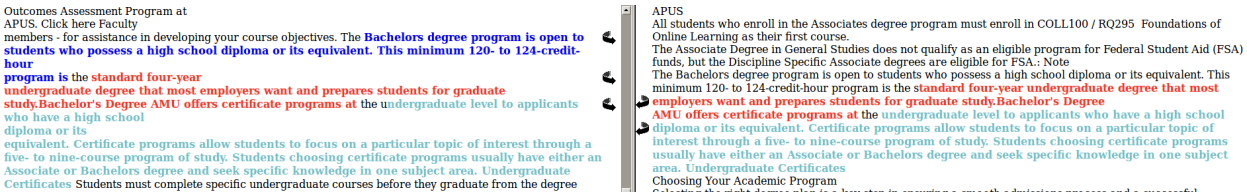


Figura 7.10: *Alinhamentos de comprimento maximal encontrados pelo JPlag ao detectar um caso de detecção de plágio. O alinhamento de maior comprimento, com 60 tokens, está destacado na cor celeste.*

Para esse mesmo caso de detecção de plágio foi rodado o BLASTD. A nossa ferramenta detectou 13 alinhamentos mostrados na figura 7.11. O comprimento maximal, mostrado na figura 7.12, contém 107 tokens. Este alinhamento possui o alinhamento de 60 tokens detectado pelo JPlag mais os outros dois destacados nas cores vermelha e azul [4]. O JPlag, de acordo com o algoritmo GST, somente relata a ocorrência de um segmento. Este comportamento explica porque os alinhamentos nas cores vermelho e azul não fazem parte do alinhamento na cor celeste.

Summary

Id	Snippet	Coordinates	Accumulated Score	# Matches	# Similar	# Divergence	# Gaps
1	the bachelor s degree ... subject area undergraduate certificates	(538,197) => (644,303)	214	107	0	0	0
2	the bachelor s degree ... prepares students for graduate	(538,0) => (579,41)	84	42	0	0	0
3	degree from an institution ... a recognized accrediting body	(1412,595) => (1421,604)	20	10	0	0	0
4	students who possess a ... diploma or its equivalent	(546,125) => (555,134)	17	9	0	1	0
5	10110 battleview parkway suite ... 114 manassas va 20109	(1096,751) => (1103,758)	16	8	0	0	0
6	to applicants who have ... diploma or its equivalent	(590,7) => (600,17)	16	9	0	2	0
7	transfer credit evaluation tce ... tce application and fee	(1475,514) => (1481,520)	14	7	0	0	0
8	a high school diploma ... diploma or its equivalent	(594,128) => (600,134)	14	7	0	0	0
9	1 877 777 9081 ... email us at	(309,360) => (315,367)	11	7	0	0	1
10	for full instructions on ... full instructions on tce	(1482,678) => (1486,682)	10	5	0	0	0
11	you must submit the ... must submit the following	(1398,500) => (1402,504)	10	5	0	0	0
12	students to focus ... students to focus on	(741,263) => (744,266)	8	4	0	0	0
13	earned a bachelor's ... earned a bachelor's degree	(1409,60) => (1412,63)	8	4	0	0	0
			442	224	0	3	1

Figura 7.11: *Resumo dos alinhamentos encontrados pelo BLASTD para um caso de detecção de plágio.*

Por tanto, o BLASTD consegue relatar mais casos de plágio que o algoritmo GST, mas é necessário um experimento maior com todos os dados do PAN 2013 para obter as métricas da competição.

Por outro lado, fizemos também um experimento com o JPlag com dois documentos históricos. Especificamente comparamos os textos de Wadding e Angelo Clareno. A

[4] O número de tokens relatado pelo JPlag não corresponde com o número de tokens nos alinhamentos detectados.

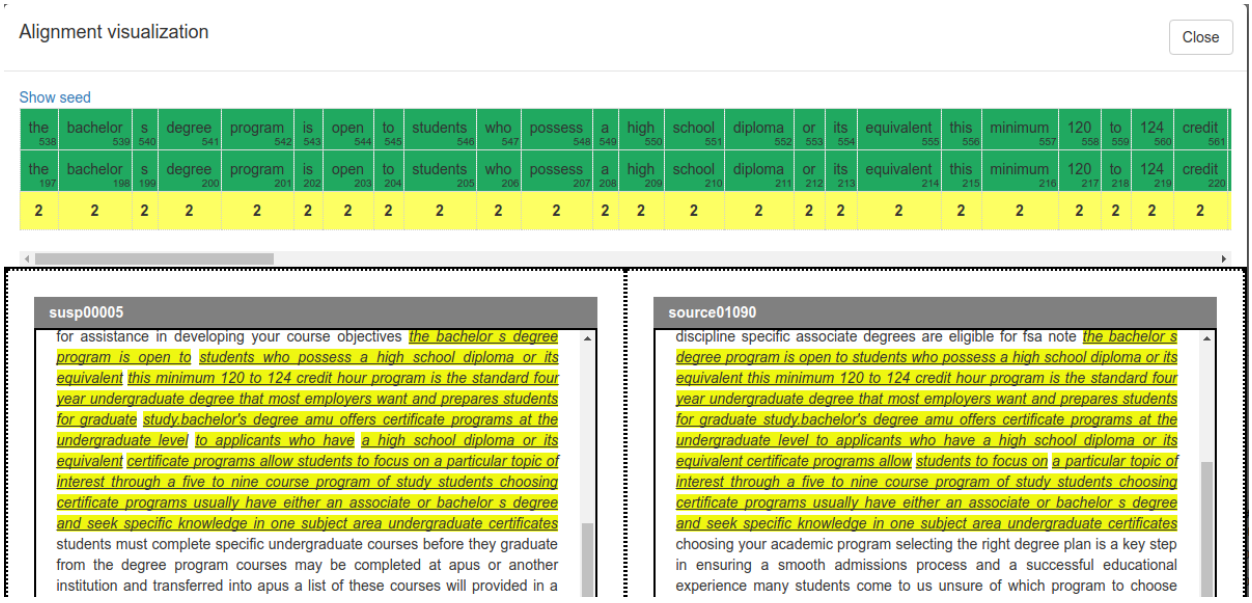


Figura 7.12: *Alinhamentos encontrados pelo BLASTD ao detectar um caso de detecção de plágio. O alinhamento de maior comprimento, com 107 tokens, está destacado na cor amarela.*

figura 7.13 mostra o resumo dos alinhamentos encontrados pelo JPlag para estes documentos, enquanto que a figura 7.14 mostra alguns alinhamentos locais que correspondem ao começo de ambos documentos.

	wadding0.txt (16.285715%)	angelo0.txt (4.470588%)	Tokens
	wadding0.txt(1-1)	angelo0.txt(1-1)	8
	wadding0.txt(1-1)	angelo0.txt(1-1)	9
	wadding0.txt(1-1)	angelo0.txt(1-1)	5
	wadding0.txt(3-3)	angelo0.txt(8-8)	20
	wadding0.txt(3-3)	angelo0.txt(8-8)	10
	wadding0.txt(3-3)	angelo0.txt(8-8)	5

Figura 7.13: *Resumo dos alinhamentos encontrados pelo JPlag na comparação de textos históricos.*

Por outro lado, como já vimos anteriormente na figura 7.2, o BLASTD consegue relatar 8 alinhamentos. Na figura 7.15 mostramos um alinhamento local encontrado pela nossa ferramenta. Este alinhamento possui espaços inseridos além de colunas de palavras com similaridade e disparidade (a taxa de erro é de 30%). Este alinhamento contém aquele destacado na cor azul pelo JPlag na figura 7.14. Vemos que o BLASTD consegue relatar padrões mais complexos que o JPlag para o caso de documentos históricos.

De certa forma, já esperavamos que esses experimentos realizados com o JPlag levassem a resultados inferiores àqueles apresentados pelo BLASTD porque, usando a busca de segmentos maximais comuns de comprimento mínimo, a ferramenta não deve encontrar alinhamentos de documentos onde existem ocorrências de substituições, inserções ou remoções de palavras.

Ainda que não tenhamos realizado nenhum experimento com a ferramenta MOSS pelo motivo já exposto, suspeitamos que a escolha da menor chave em uma janela possa ter um impacto negativo na busca de similaridade entre documentos com muitos erros como aqueles obtidos através de OCR.

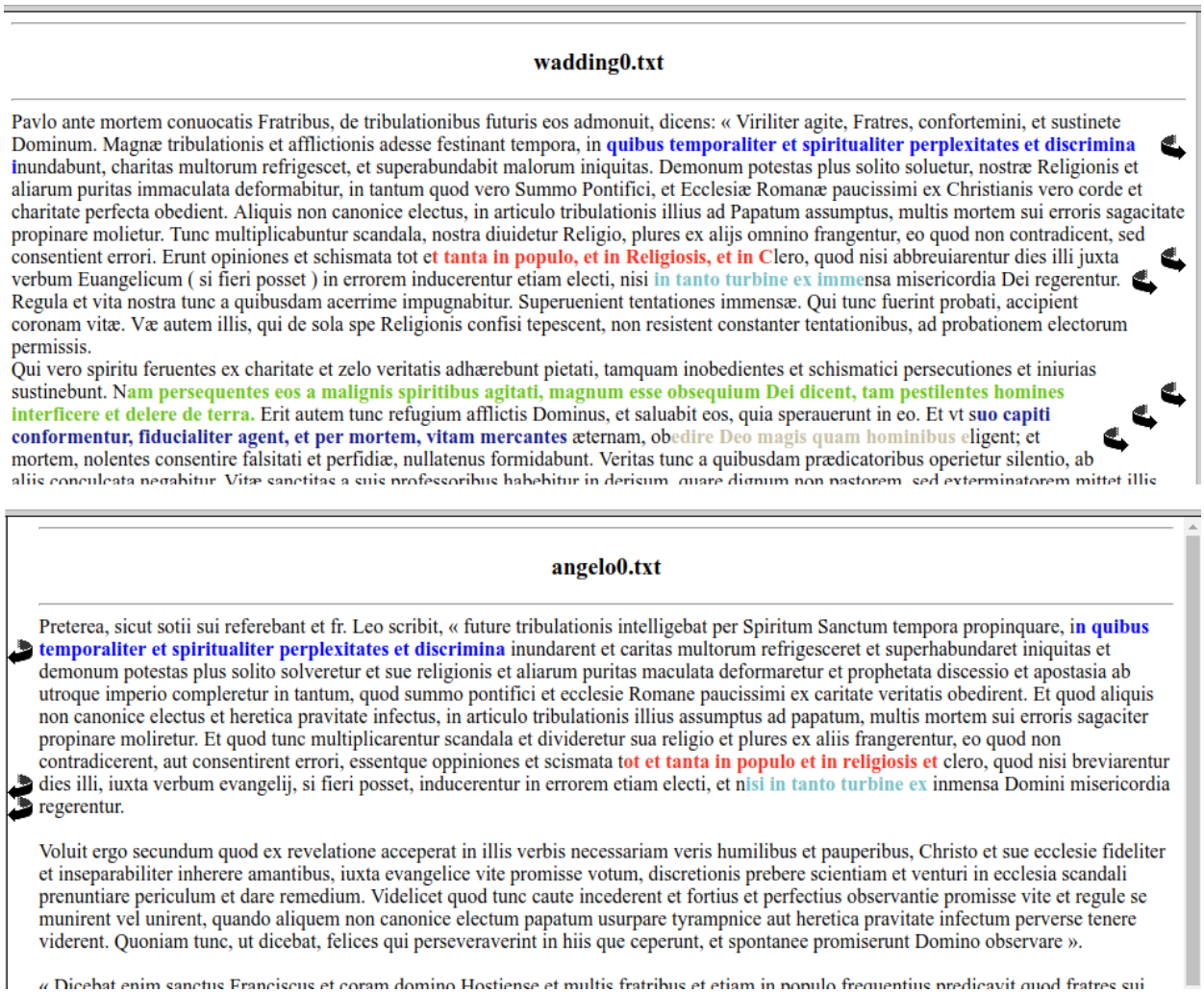


Figura 7.14: *Alinhamentos de comprimento maximal encontrados pelo JPlag na comparação de textos históricos. Foram relatados 6 alinhamentos sendo que o maior deles possui 20 tokens.*

Figura 7.15: *Alinhamento encontrado pelo BLASTD na comparação de textos históricos. Em comparação com o JPlag, a nossa ferramenta consegue obter alinhamentos mais significativos.*

7.3 Processamento de Imagens

Foram também realizados testes baseados nos algoritmos de processamento de imagens mostrados no capítulo 3. Foram realizados testes em uma das páginas do conjunto de imagens que pertence à obra *Beati Patris Francisci Assiciatis Opuscula* de Luke Wadding escrita em 1623. A página possui 43 imagens que foram classificadas como caracteres grudados.

Especificamente, fizemos testes de ajuste paramétrico utilizando várias fórmulas para calcular a *pontuação de combinação* para as pontuações individuais x e y de duas regiões de uma imagem que possui caracteres com ligações. O algoritmo de programação dinâmica que utiliza esta pontuação é mostrado no capítulo 3. Foram testadas as seguintes fórmulas para computar a pontuação de combinação:

A a multiplicação $\beta x * y$

B a média geométrica $\beta\sqrt{x * y}$

C a média harmônica $\beta\dfrac{2xy}{x+y}$

Para cada fórmula, foi realizada a otimização paramétrica utilizando valores de β que aumentassem a acurácia da segmentação da imagem. Junto com essas versões do algoritmo de programação dinâmica, também foi testada a segmentação do Tesseract (versão 3.05.01).

O método utilizado para verificar a segmentação foi visual. No caso do algoritmo de programação dinâmica, verificamos a segmentação em cada imagem que possui caracteres grudados. Somente os casos onde a segmentação é correta são contabilizados de forma positiva e no fim é computada a sua proporção com respeito ao total de imagens que possuem caracteres grudados.

A figura 7.16 mostra um estudo paramétrico para a fórmula A, multiplicativa. Os valores de β foram $1.5, 2, 3, 4, 5, 6.1, 7.2$ e o melhor resultado, 77%, foi alcançado para $\beta = 2$.

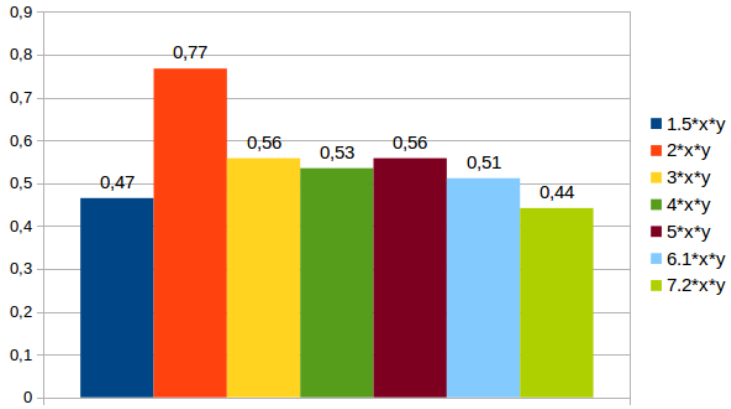


Figura 7.16: *Acurácia da fórmula A no algoritmo de segmentação de caracteres grudados*

Na figura 7.17, mostramos o resultado para a fórmula B, que utiliza a média geométrica. Os valores de β para esta execução foram $0.25, 0.35, 0.5, 0.7, 1.0, 2.0$, e o melhor resultado foi alcançado para $\beta = 0.7$, com acurácia de 86%.

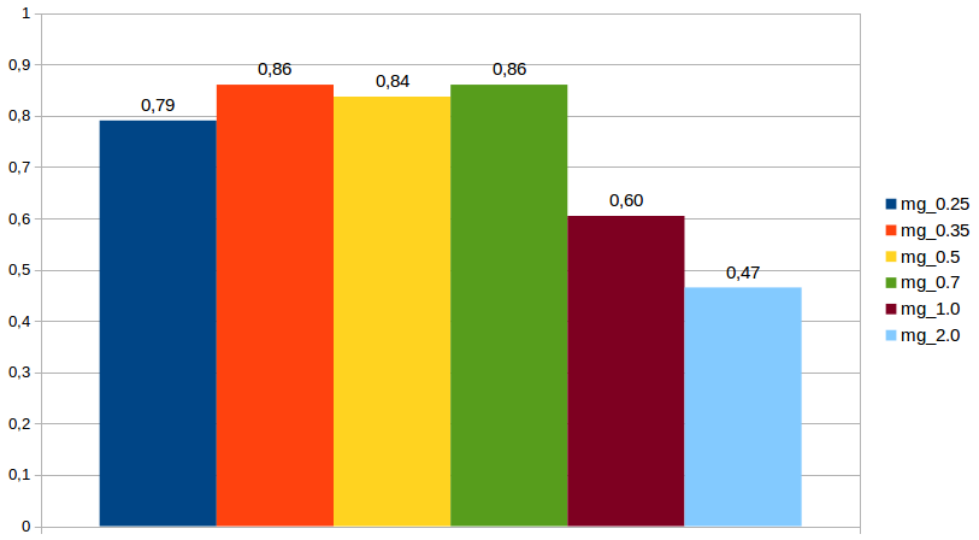


Figura 7.17: *Acurácia da fórmula B no algoritmo de segmentação de caracteres grudados*

Finalmente, na figura 7.18 mostramos os resultados para a média harmônica, com β assumindo valores de $0.25, 0.35, 0.5, 0.7, 1.0, 2.0$. Novamente, o desempenho do algoritmo é alto chegando a 86% para β igual a $0.35, 0.5, 0.7$. Este resultado se nos mostrou mais robusto em comparação com a média geométrica, onde a acurácia também atingiu 86%, razão pela qual escolhemos esta fórmula como a melhor para a pontuação de combinação.

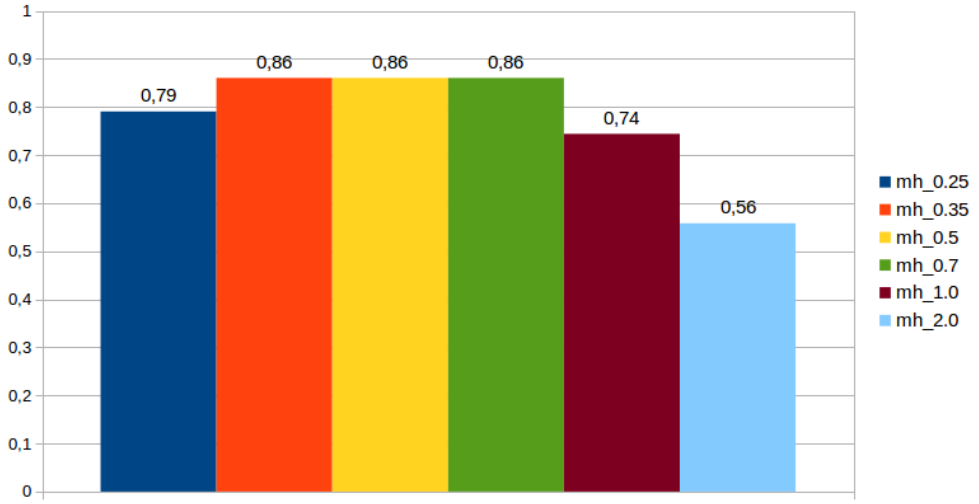


Figura 7.18: *Acurácia da fórmula C no algoritmo de segmentação de caracteres grudados*

Na figura 7.19, comparamos os melhores resultados de cada fórmula como o Tesseract, que utiliza no algoritmo de segmentação a pontuação do classificador para as regiões eventualmente resultantes do processo de segmentação para tomar uma decisão.

Por outro lado, no caso do Tesseract, foram também verificadas as segmentações dessas mesmas imagens. Neste caso o método também foi visual mas tivemos como referência o arquivo de saída HOCR gerado pelo Tesseract. Caso o texto gerado seja correto então o resultado é contabilizado de forma positiva. Assim como com os algoritmos de programação dinâmica, também é calculada a proporção em relação ao total de imagens com caracteres grudados.

A fórmula C, que usa a media harmônica, acerta nos mesmos casos que o Tesseract.

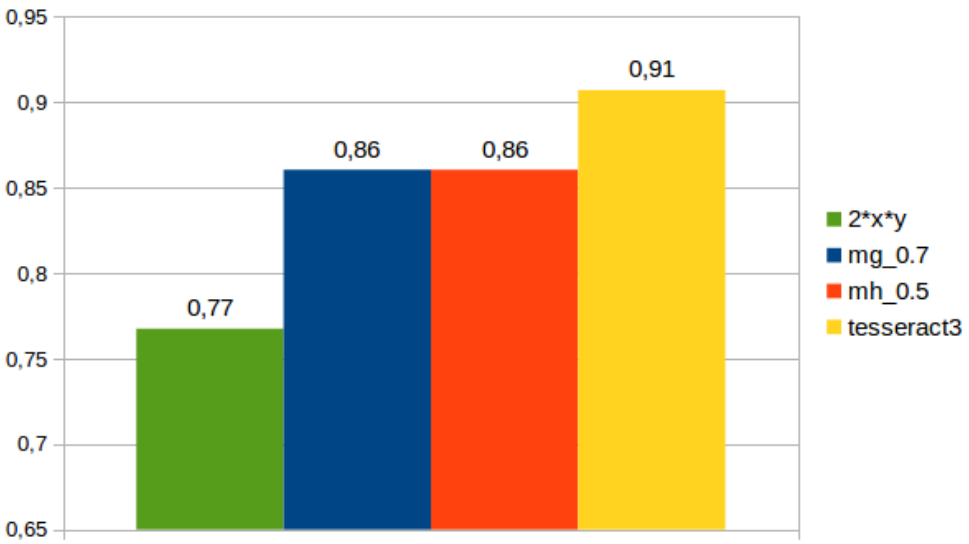


Figura 7.19: *São exibidas as melhores pontuações de acurácia alcançadas pela segmentação de caracteres não supervisionada com as fórmulas A,B e C. Mostramos também o resultado para o Tesseract, que ao contrário dos outros casos, usa a pontuação de classificação supervisionada das imagens segmentadas para decidir a melhor segmentação.*

Com respeito aos casos de falha, essa fórmula erra em 6 casos de segmentação enquanto que o Tesseract erra em 4.

O Tesseract utiliza uma técnica de busca em profundidade (algoritmo A*) para encontrar a melhor segmentação. Para complementar a estratégia gulosa, o Tesseract se apoia no classificador.

O nosso algoritmo obtém uma acurácia próxima ao do Tesseract pois ele faz uma escolha inteligente no espaço de segmentações dos caracteres mesmo com um modelo não-supervisionado que possui menos informação que o classificador do Tesseract (o modelo possui duas informações, as medidas de localização e dispersão dos caracteres da página). Os dois caracteres que não foram identificados pelo nosso algoritmo são casos que podem ser resolvidos com um aprimoramento da geração de caminhos de segmentação.

Vale a pena ressaltar que nosso modelo é não-supervisionado, não usando informação da classificação das imagens depois da segmentação. Ademais, existe espaço para melhorias pois não foram exploradas outras informações, como por exemplo a quantidade de pixels que são apagados da imagem ao segmentar duas regiões. Também, seria interessante rodar o algoritmo em imagens mais difíceis, com mais casos de caracteres grudados

Tentamos também comparar o nosso algoritmo competindo em um evento organizado pelo Kaggle. A competição, embora realizada em 2016, infelizmente foi organizada somente para alguns convidados e o seu conjunto de dados não se encontra disponível.

7.4 Busca Aproximada

Fizemos também experimentos sobre os algoritmos de busca aproximada apresentados no capítulo 4. Nesse capítulo foi apresentado um filtro que recupera as palavras de um dicionário de termos para um termo de entrada e uma taxa de erro. Esse filtro computa, primeiro, a distância por sacos de símbolos. Os termos que estão dentro da taxa de erro são verificados utilizando o Teste de Ukkonen que é uma solução baseada em programação dinâmica. Este filtro é utilizado na busca de bibliotecas e no alinhamento local

de documentos de texto para encontrar sementes de palavras aproximadas.

Nos interessa saber o impacto da computação do saco de símbolos no alinhamento local de documentos de texto. Este filtro diminui o número de chamadas ao Teste de Ukkonen. O nosso experimento mede o tempo gasto no alinhamento local das obras de Angelo Clareno e Frei Conrado descritos no começo deste capítulo. O comprimento em palavras de ambos os documentos é 1265 e 7122 respectivamente. Os testes foram executados com os seguintes parâmetros de alinhamento: $k = 3$, $t_{igual} = 2$, $t_{similar} = 1$, $t_{dspar} = -1$, $t_{\bar{e}} = -3$, queda máxima $X = 15$, pontuação mínima de um PAP 5 e taxa de erro assumiu os valores $\beta = 20, 35, 50\%$. Para cada valor de β foram executados 10 execuções do alinhamento local e são apresentadas as médias.

Os resultados são mostrados na figura 7.20. A execução do alinhamento local sem saco de símbolos fica 19%, 47% e 25% mais lenta que a versão com saco de símbolos para β igual a 20%, 35% e 50%, respectivamente. Também verificamos que, em média, a execução do filtro de busca aproximada consome entre 40% e 50% do tempo total de execução do alinhamento local.

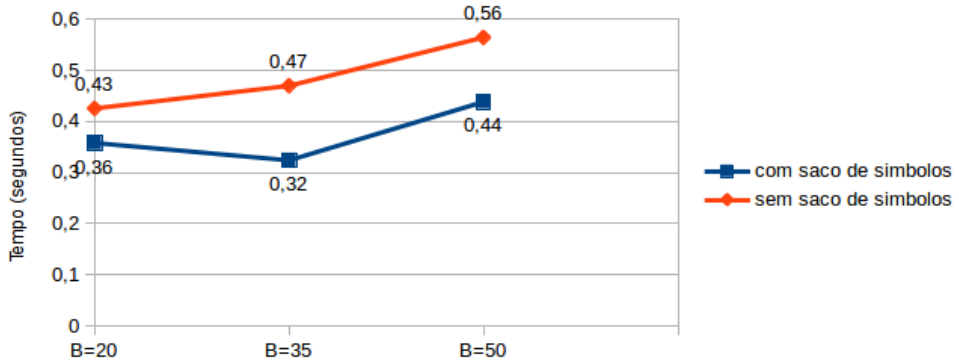


Figura 7.20: *Resultados da medição de tempo da Busca Aproximada no alinhamento das obras de Angelo Clareno e Frei Conrado.*

7.5 Conclusões

Neste capítulo foram apresentados alguns experimentos realizados em nosso sistema. O primeiro experimento mostra a aplicação da técnica de alinhamento múltiplo apresentado no capítulo 5 para tentar descobrir a gênese da profecia de São Francisco de Assis. Este estudo sugere que a profecia contida no texto *Verba Conradi* serviu como base para a profecia contida no texto *Expositio regulae Fratrum Minorum* de Angelo Clareno que o escreveu em 1323. Por outro lado, também temos evidência de que a cópia da profecia contida no texto *De conformitate vitae beati Francisci ad vitam Domini Iesu* escrito por Bartolomeu de Pisa, e aprovado pelo Capítulo Geral da ordem franciscana em 1396, é gerada a partir dos textos de Angelo e Conrado. Como vimos, o texto da profecia compilada por Wadding é quase toda ela coberta por alinhamentos com os textos disponíveis de Angelo Clareno, Bartolomeu de Pisa e Frei Conrado sendo que os primeiros dois são descendentes do último. Se Armstrong afirma que *Verba Conradi* são palavras de Frei Leão [AHS99], e este as atribui a São Francisco de Assis conforme o texto afirma, é natural se supor que o fundador da ordem franciscana tenha mesmo comunicado a profecia narrada a nós por Frei Leão e Frei Conrado, ainda que edições mais recentes de *Escritos de São Francisco* tenham preferido expurgar a profecia que Luke Wadding inseriu em 1623 na primeira edição da obra.

No segundo experimento, foi explorado o alinhamento local de textos para tentar encontrar regiões de plágio entre dois documentos. Foi escolhido o conjunto de dados de teste da competição PAN 2013 que possui ao todo 5.000 documentos de texto apresentando casos de plágio entre pares de documentos. O nosso algoritmo BLASTD obteve pontuações altas mesmo sem nenhum recurso linguístico que poderia melhorar a sua performance.

No terceiro experimento, foi estuda a acurácia do algoritmo de segmentação de caracteres grudados proposto no capítulo 3. Foram estudadas algumas variantes da fórmula que combina regiões da imagem para tentar decidir que região contém um caractere minúsculo. A nossa solução obteve uma acurácia de 86% embora possua um modelo não-supervisionado que dá suporte ao método de combinação de regiões. Este resultado pode ser melhorado a partir de um estudo mais detalhado do comportamento dos algoritmos nos casos onde ele erra. Em particular, julgamos adequado examinar os critérios de geração e seleção dos caminhos de segmentação neste conjunto de exemplos.

No último experimento, pudemos verificar a eficácia do filtro de saco de símbolos ao diminuir consideravelmente as chamadas realizadas para o Teste de Ukkonen, a ponto de que um alinhamento local sem o filtro chega a executar 47% mais lento quando adotamos $\beta = 35\%$.

Capítulo 8

Conclusões e Trabalhos futuros

O nosso trabalho se focou no estudo de algoritmos de Processamento de Imagens, Busca Aproximada e Biología Molecular Computacional em documentos históricos em formatos de imagens e texto. Ainda, esses algoritmos foram integrados em um sistema web que permite aos usuários realizar OCR, busca e alinhamento em documentos históricos. Os documentos históricos que contêm texto apresentam uma taxa de erro alta dado que possuem uma tipografía desconhecida aos motores OCR.

Para resolver o problema específico da baixa qualidade desses sistemas, foi proposto no capítulo 3 um fluxo de processamento para imagens de documentos antigos. Este fluxo contempla imagens com alto ruído de background (por exemplo manchas) chegando até a segmentação de caracteres grudados. Esse último fenômeno é um problema bastante difícil na área de Processamento de Imagens e, até onde temos visto, a modelagem e soluções propostas baseiam-se fortemente em estratégias gulosas e ainda são insatisfatórias, requerendo heurísticas que fazem uma filtragem a posteriori dos erros mais frequentes. Este é também o caso do Tesseract. A nossa contribuição nessa área passa pela proposição de um algoritmo de programação dinâmica que roda em tempo polinômial no número de caminhos de segmentação da imagem. O algoritmo utiliza um modelo não-supervisionado obtido mediante técnicas de Estatística Robusta. Nos experimentos sobre uma página de um documento de 1623, a acurácia desse algoritmo foi de 86% ficando muito próximo de algoritmos de segmentação como o do sistema OCR Tesseract que obteve 91% com os mesmos dados e lançando mão de um modelo supervisionado.

Depois de obtidos os documentos de texto para um conjunto de imagens, o nosso sistema oferece ao usuário a capacidade de realizar busca aproximada. No capítulo 4 propomos algumas técnicas que restringem fortemente o espaço de busca aproximada através da utilização de índices de k-gramas e da computação da distância de sacos de símbolos para produzir termos aproximados presentes nos documentos de texto. Esses termos próximos servem para estender a consulta feita pelo usuário. Nos experimentos feitos para determinar a gênese de documentos antigos (capítulo 7) esta ferramenta se mostrou útil ao verificar a ocorrência de trechos de texto em documentos dos séculos 14 e 17.

Ademais o sistema oferece a capacidade de realizar alinhamentos locais e alinhamento múltiplo destes documentos. O principal algoritmo de alinhamento local, o BLASTD, é descrito no capítulo 5 e utiliza heurísticas da área de Bioinformática para determinar sementes entre dois documentos de texto. As sementes são estendidas com base em um sistema de pontuações definido pelo usuário e cujo resultado final são regiões de texto comuns em ambos os documentos. Para poder tratar documentos históricos com uma taxa

de erro considerável, BLASTD utiliza as técnicas de busca aproximada do capítulo 4. Em nossos experimentos com esse tipo de documentos, BLASTD se mostrou bastante eficaz ao localizar regiões comuns mesmo entre documentos diversos com diferença de 3 séculos. Esta análise foi aplicada de forma a mostrar o alto grau de semelhança entre um texto colocado pelo historiador franciscano Luke Wadding entre as obras de São Francisco de Assis e textos datados do início do século XX. Fizemos também uma comparação do BLASTD com outra ferramenta que visa também encontrar alinhamentos locais utilizando a técnica de Greedy String Tiling (GST) e concluimos que o BLASTD é superior pois consegue encontrar padrões mais complexos entre documentos históricos, relatando mais alinhamentos e alinhamentos mais longos.

Por outro lado, o BLASTD foi testado também no dominío da detecção de plágio. Neste domínio ele alcançou uma performance de detecção alta (pontuação $F_1 = 0.811$) em categorias onde a informação linguística é importante (a outra solução comparada obteve pontução $F_1 = 0.819$). Este resultado é animador pois existe ainda a possibilidade do BLASTD ser estendido com informação linguística. O algoritmo GST foi testado neste domínio para um par de documentos com o objetivo de compará-lo com o BLASTD. O GST se mostrou inferior principalmente por sua limitação em tratar as repetições presentes no mesmo documento.

Em todas as tarefas descritas sobre a utilização do alinhamento de dois ou mais documentos, os diferentes módulos de visualização mostraram-se bastante úteis para cada tipo de alinhamento. Houve uma parcela significativa de esforço nesta área e o sistema ainda pode ser melhorado para incluir outros tipos de visualização, como por exemplo, uma integração mais adequada da visualização dos diversos alinhamentos locais envolvendo dois documentos através de grafos bipartidos.

Finalmente, foi proposto um método que treina sistemas OCR utilizando alinhamentos de pares de documentos de texto. Este método, descrito no capítulo 6, visa corrigir a saída do OCR utilizando alinhamento caractere a caractere. As colunas do alinhamento que apresentam disparidade são utilizadas para gerar novos casos de treinamento para o OCR. Este método precisa ainda ser melhorado para obter uma arquitetura efetiva por conta das dificuldades em treinar o Tesseract.

Com respeito ao estudo filogenético de documentos históricos, a catalogação e compilação de fragmentos e documentos históricos tem propiciado muitas reinterpretações e elucidações possam ser feitas a respeito de documentos antigos importantes. Por exemplo, o trabalho realizado por Hermann Diels (1848-1922) na sua obra "Fragmenter der Vorsokratiker" e publicado em 1906 constitui um trabalho enorme que demorou 20 anos para ser finalizado e reúne todas as frases conhecidas dos filósofos pre-socráticos. Este trabalho é importante porque os textos que temos sobre esses filósofos são somente fragmentos que por vezes parecem ser até contraditórios. O trabalho de Diels (depois retomado por Walther Kranz em 1934) deixa mais perguntas em aberto [Phi56] e o seu estudo pode levar a reescrever a história dos filósofos daquela época. O trabalho de Diels e Kranz serviu de fundamento para o filósofo italiano Giovanni Reale na sua obra "Para uma nova interpretação de Platão". Nesta obra, Reale realiza uma rigorosa análise textual dos "Dialogos" de Platão levando em conta as "doutrinas não-escritas" para que "possa emergir em toda sua grandeza a primeira e a mais audaz construção metafísica da filosofia occidental" [Rea97]. Em sua obra, o autor observa o quanto interpretações antigas e equivocadas de Platão foram determinantes no pensamento de Nietzsche antes que o autor alemão pudesse contar com as elucidações providas pelo trabalho de Diels. Tivemos a possibilidade de expor o nosso trabalho ao exegeta bíblico Innocenzo Gargano, que costuma fazer reinterpretações envolvendo diferentes documentos históricos e se mostrou

bastante interessado a uma eventual utilização e colaboração de nossa ferramenta.

Ao todo, o nosso sistema demonstra que uma tal ferramenta é prática e de grande auxílio no problema enfrentado pelos historiadores e linguistas ao fazer busca e comparação em documentos históricos. Vale a pena ressaltar duas características que o fazem útil a esses profisionais: a velocidade de processamento, na busca e alinhamento, mesmo para documentos grandes e o fato de ser um sistema web que facilita a colaboração de vários usuários.

Até onde seja de nosso conhecimento, este é o único sistema que integra reconhecimento digital de textos em imagens de documentos antigos com busca e indexação aproximada e alinhamento múltiplo dos documentos, muito importante nos estudos filogenéticos de documentos históricos e a melhoria da performance do próprio classificador OCR.

Apêndice A

Guia de utilização do sistema

A interface de usuário do nosso sistema foi construída usando principalmente Javascript, CSS e HTML. São feitas chamadas JSON ao sistema para executar a funcionalidade oferecida ao usuário. A funcionalidade agrupa-se nos seguintes módulos:

- Processamento de imagens
- Documentos de texto
- Bibliotecas
- Busca
- Parâmetros de alinhamento.
- Alinhamento

A tela principal do sistema, mostrada na figura A.1, possui um menu que permite ao usuário acessar cada módulo.

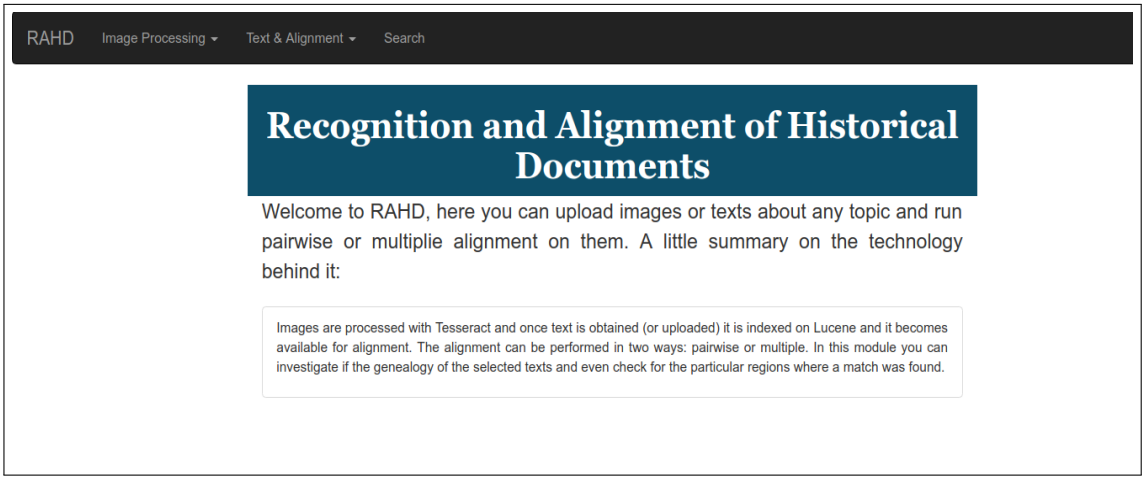


Figura A.1: *Menu principal do sistema*

Ao clicar em alguma opção o sistema mostra uma tela diferente para cada um. Nas seguintes seções fazemos uma breve introdução ao funcionamento de cada tela.

A.1 Processamento de imagens

No módulo de Processamento de Imagens o usuário pode adicionar, atualizar ou remover conjuntos de imagens. As imagens são previamente armazenadas dentro de uma pasta específica no sistema de arquivos do computador que hospeda o sistema . O sistema permite ao usuário navegar dentro dessa pasta para selecionar aquela que contém as imagens de interesse. Ao adicionar (ou atualizar) um conjunto de imagens, o usuário pode configurar a língua em que as imagens vão ser reconhecidas pelo sistema OCR (figura A.2) assim como os parâmetros da etapa de segmentação descrita no capítulo 3 (figuras A.3, A.4, A.5).

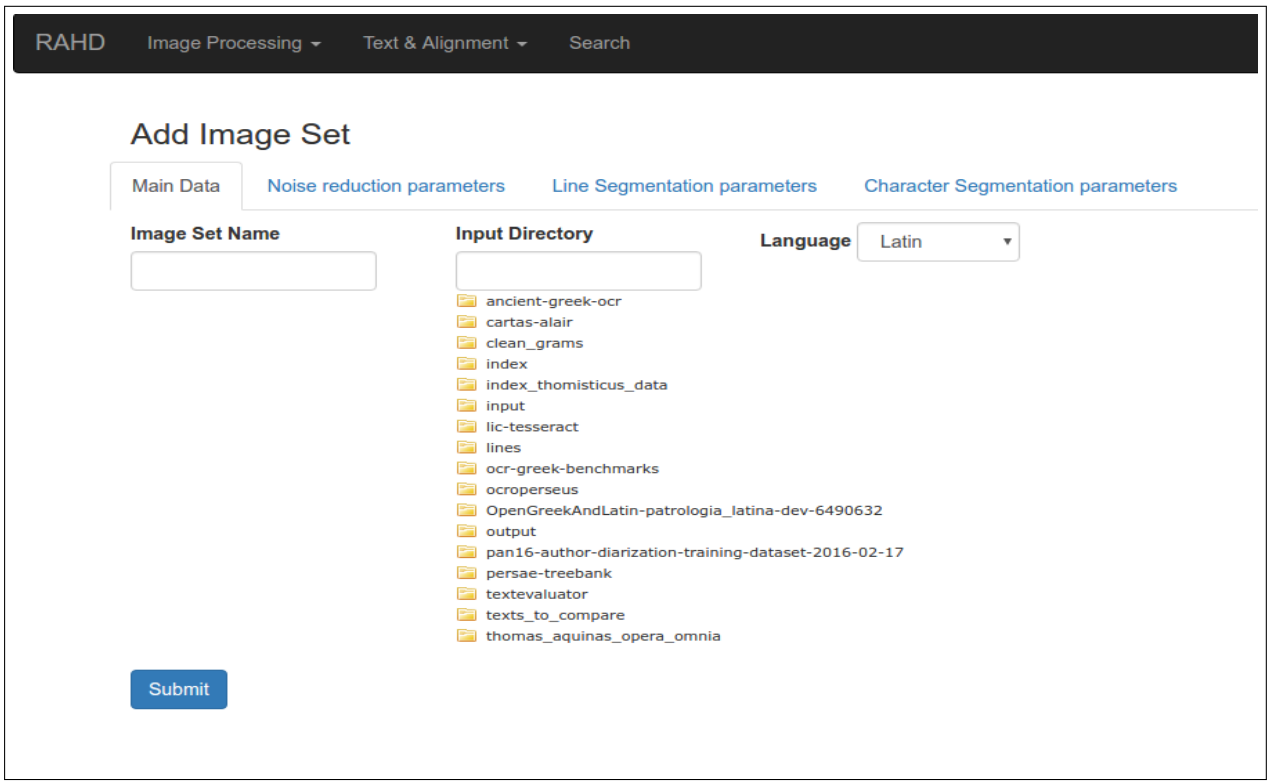


Figura A.2: *Adição (ou atualização) de conjuntos de imagens.*

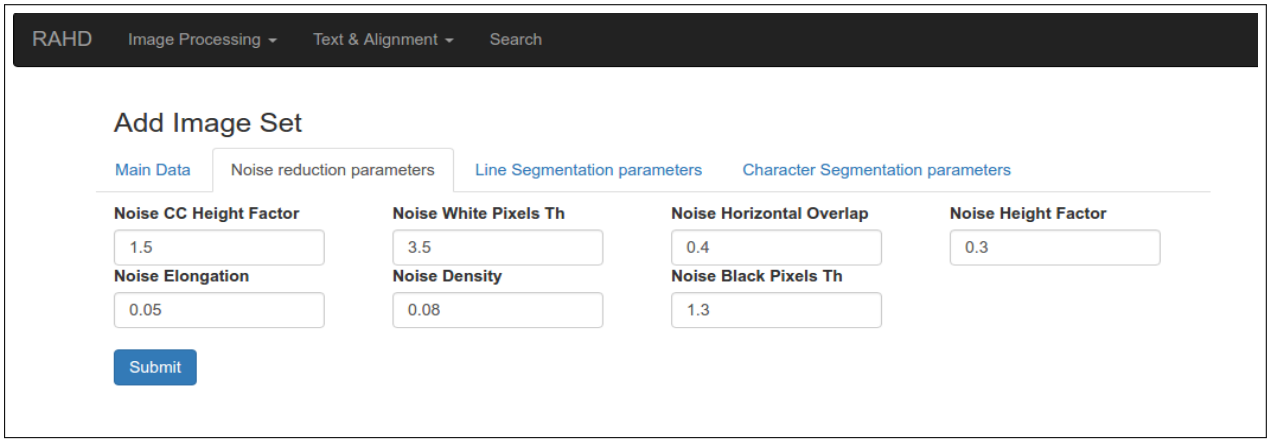


Figura A.3: *Configurando parâmetros de redução de ruído para um conjunto de imagens*

O sistema também lista os conjuntos de imagens disponíveis (figura A.6). Nesta listagem são oferecidas as opções de atualização e remoção para cada conjunto.

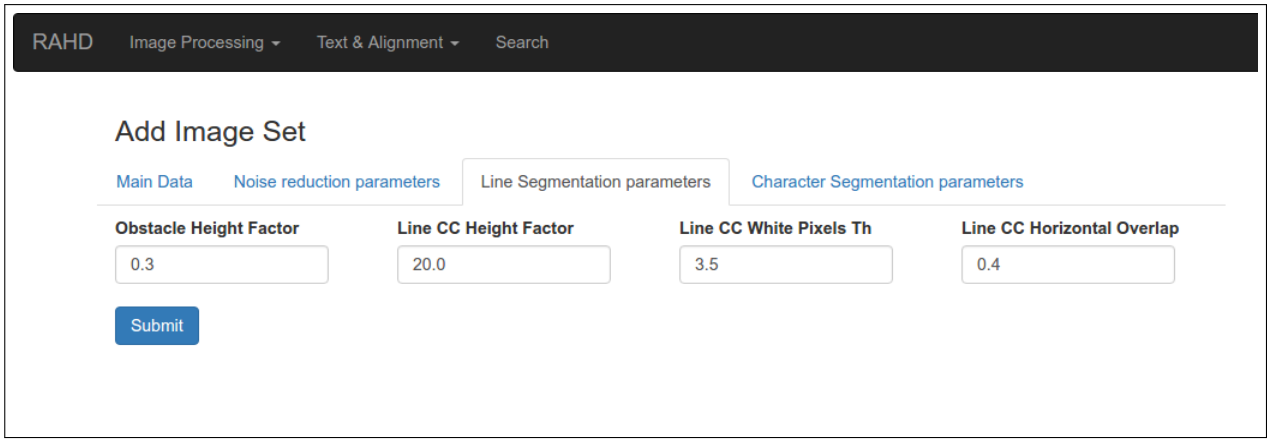


Figura A.4: *Configurando parâmetros de segmentação de linhas para um conjunto de imagens*

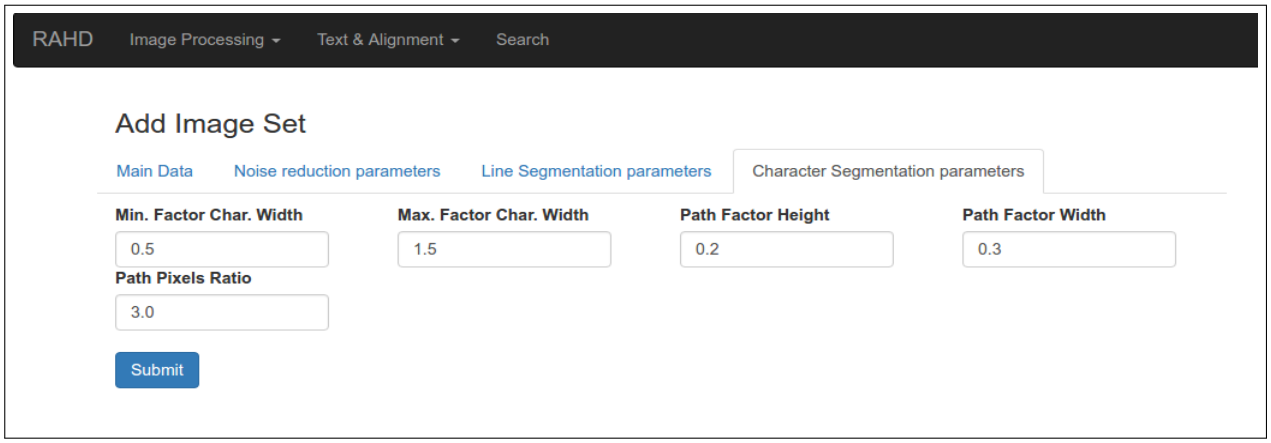


Figura A.5: *Configurando parâmetros de segmentação de caracteres para um conjunto de imagens*

A.2 Documentos de texto

O módulo de documentos de texto pode ser acessado a partir do menu que fica no topo clicando na opção "Documents" como mostra a figura A.7. Ao clicar nela, o sistema mostra a listagem dos documentos cadastrados no sistema (figura A.8). Nessa tela o usuário pode adicionar, remover ou atualizar documentos de texto. Também é possível fazer filtragem dos documentos por língua ou titulo do documento.

Para adicionar um documento de texto é necessário clicar no botão "Add new document" que fica no canto superior direito. Ao clicar nesse botão, o sistema mostra a caixa de diálogo da figura A.9). Nessa caixa de diálogo podemos inserir o nome do documento, a língua em que ele está escrito e o arquivo de texto. Essa mesma caixa de diálogo é reutilizada pela operação de atualização de documentos.

A.3 Bibliotecas

Neste módulo o usuário pode agrupar conjuntos de documentos em bibliotecas. O módulo pode ser acessado através da opção "Libraries" do menu principal (figura A.10). Ao clicar nessa opção o sistema mostra a listagem de bibliotecas (figura A.11).

Novas bibliotecas podem ser criadas clicando no botão "Add new library". Ao clicar nesse botão, o sistema mostra uma caixa de diálogo com controles organizados para fa-

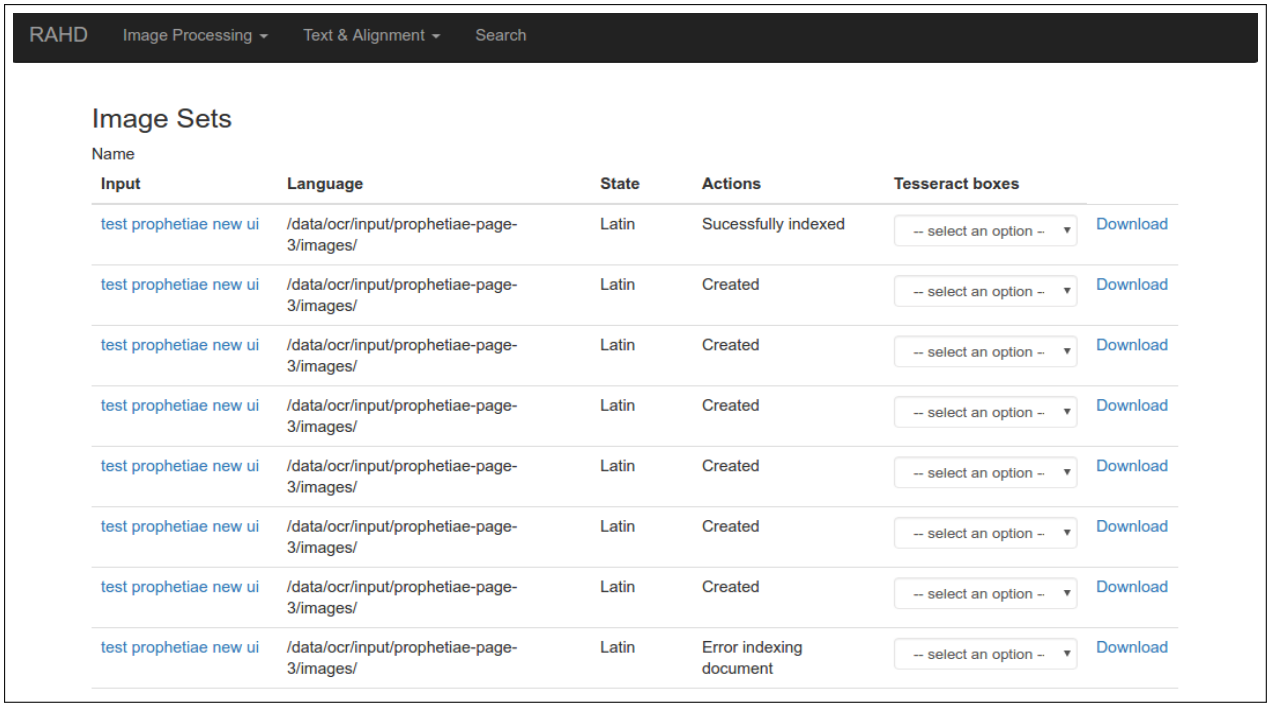


Figura A.6: *Listagem de um conjunto de imagens cadastradas no sistema*

cilitar a busca de documentos que fazem parte da nova biblioteca. A figura A.12 mostra essa caixa de diálogo enquanto qua a figura A.13 mostra uma biblioteca já configura. Essa mesma caixa de diálogo serve para atualizar bibliotecas.

Finalmente, depois de criar ou atualizar uma biblioteca, o sistema cria o índice do Lucene assim como os índices de busca aproximada para os documentos da biblioteca. O estado de indexação de uma biblioteca pode ter três valores: "Not Indexed", "Index Ready" e "Index Failed". Quando uma nova biblioteca é criada o sistema mostra ela como "Not Indexed" e um processo é disparado de forma assíncrona para criar os índices. Se a indexação ocorre de forma correta, o estado de indexação vai para "Index Ready", senão assume o valor "Index Failed". Somente as bibliotecas com estado "Index Ready" são disponibilizadas para tarefas de alinhamento e busca.

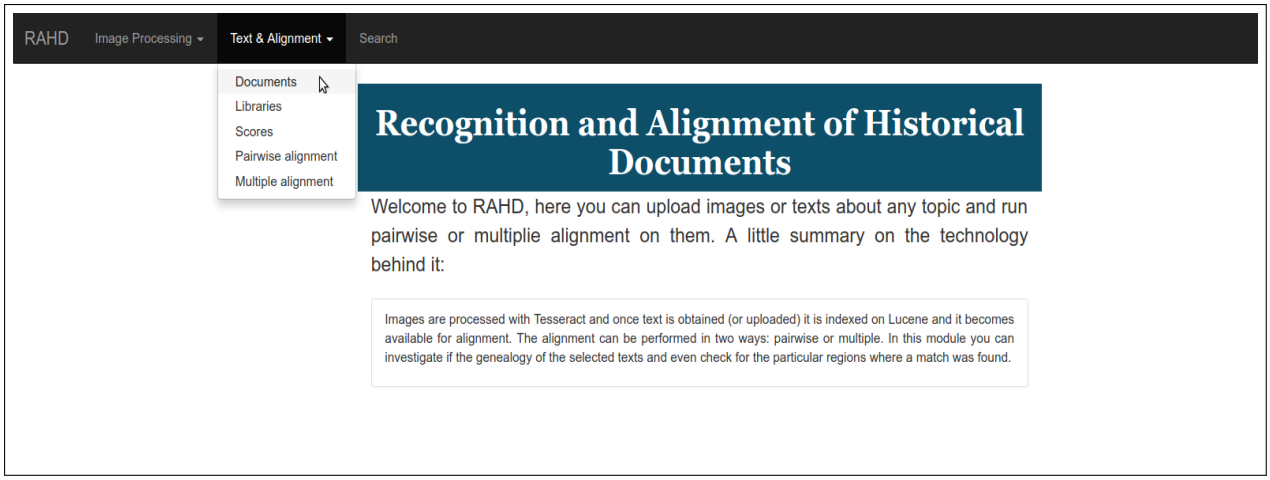


Figura A.7: *Menu de processamento de textos*

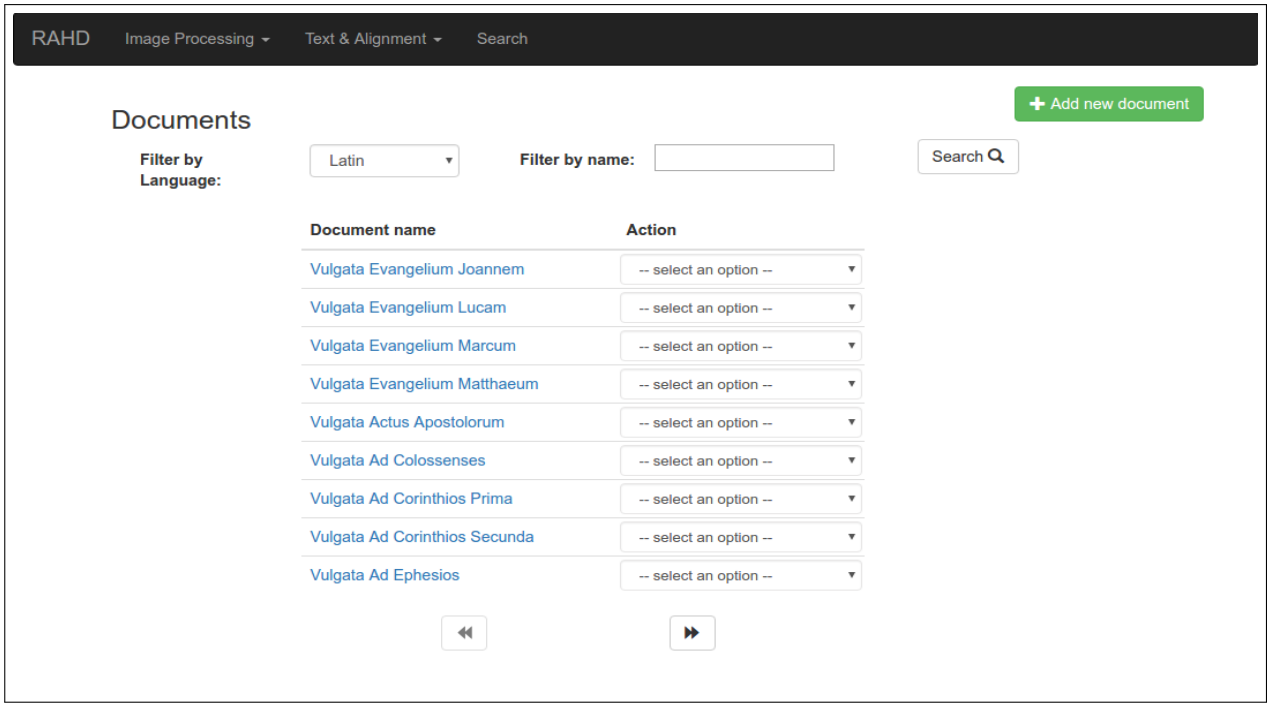


Figura A.8: *Listagem de documentos de texto*

A.4 Busca

No módulo de indexação e busca o usuário pode realizar buscas aproximadas sobre bibliotecas cadastradas no sistema. Para acessar esse módulo basta clicar na opção "Search" do menu principal (figura A.14). A tela inicial de busca (figura A.15), contém dois quatro elementos: uma caixa de texto onde o usuário pode inserir a consulta de texto, uma caixa de seleção onde é selecionada a biblioteca sobre a qual sera feita a busca, a taxa de erro admitida para procurar termos na biblioteca e finalmente o botão de submissão da busca.

Ao submeter uma consulta, o sistema gera várias consultas aproximadas utilizando as técnicas do capítulo 4. Essas novas consultas são submetidas ao índice Lucene da biblioteca e os documentos recuperados são mostrados na tela (figura A.16). Nessa tela é possível selecionar documentos para posterior execução de algoritmos de alinhamento de pares ou múltiplo. O usuário pode ainda especificar conjuntos de parâmetros cadastrados no sistema ou criar novos conjuntos A.17. Quando o usuário clica no botão "Run Alignment" o sistema mostra uma caixa de diálogo pedindo ao usuário indicar o documento de referência (A.18) e uma nova aba é aberta para mostrar o resultado do alinhamento. Dependendo do número de documentos selecionado o sistema executa o alinhamento de pares ou alinhamento múltiplo.

A.5 Parâmetros de Alinhamento

Neste módulo podem ser configurados os conjuntos de parâmetros para os algoritmos de alinhamento de pares e alinhamento múltiplo de documentos. A tela pode ser acessada clicando na opção "Scores" do menu principal (figura A.19.

Nessa tela o usuário são listados todos os conjuntos de parâmetros cadastrados no sistema(figura A.20). Em cada linha é mostrado um conjunto assim como as suas informações (pontuação de similaridade, pontuação de disparidade, etc.). Ao clicar no botão

Figura A.9: *Adicionando documento de texto*

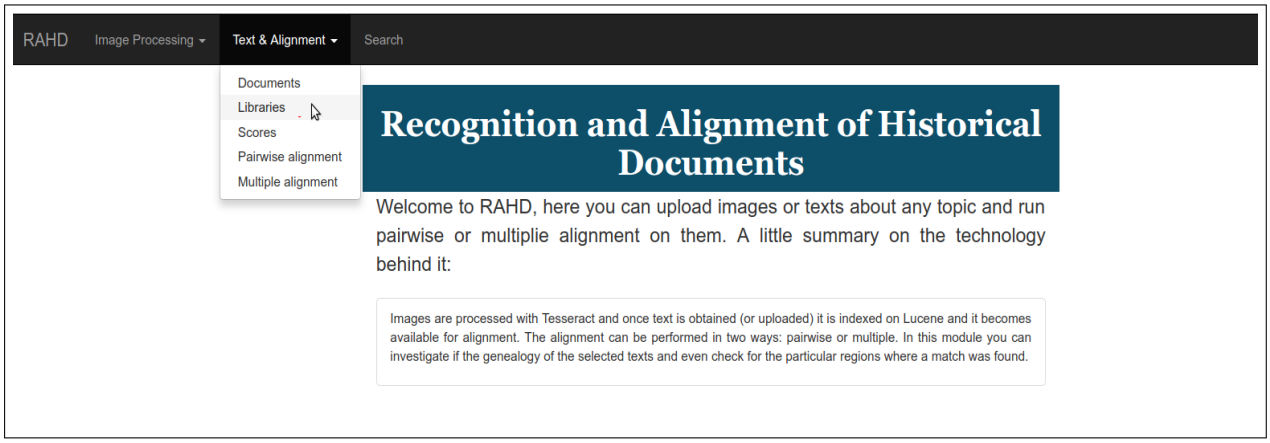


Figura A.10: *Menu de Bibliotecas*

"Add new scores", o sistema mostra uma caixa de diálogo que permite criar novos conjuntos de parâmetros. Essa caixa de dialogo, mostrada na figura A.21, possui controles que permitem inserir o nome do conjunto de parâmetros, o tamanho do k-grama, o tipo de sistema de pontuação de caracteres e as pontuações entre palavras. O tipo de sistema de pontuação de caracteres pode ser de dois tipos "Basic Score" e "Soft Score".

O primeiro tipo impede aos algoritmos de alinhamento de utilizar busca aproximada pois não admite a configuração de taxa de erro, enquanto que o segundo permite esse tipo de comportamento. Por outro lado, ao salvar as informações na caixa de diálogo, diversas validações são realizadas para assegurar a consistência dos parâmetros. A figura A.22 mostra a caixa de diálogo de um conjunto paramétrico.

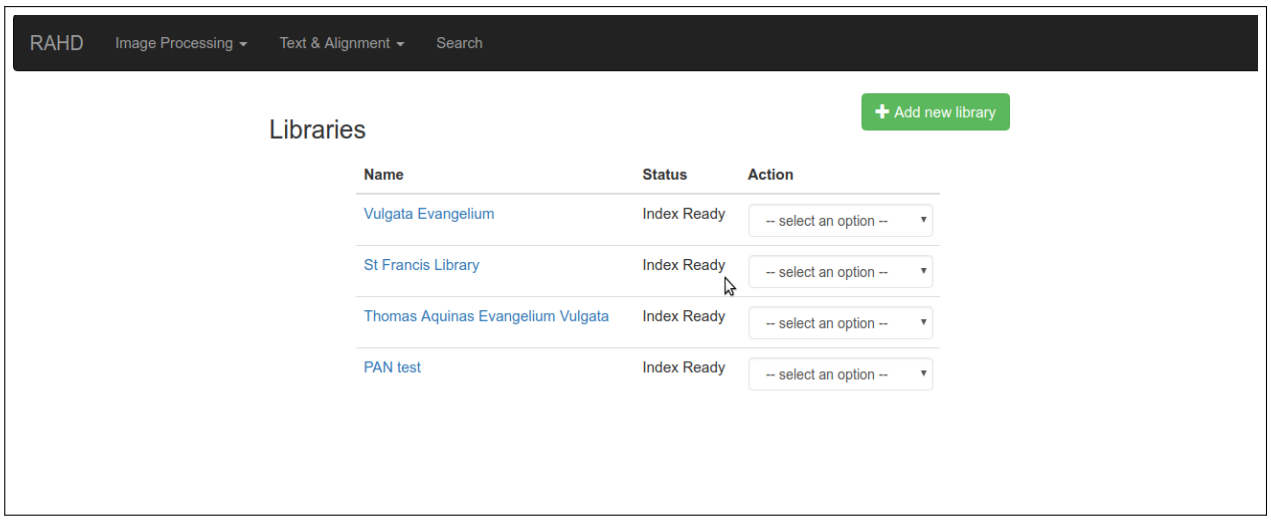


Figura A.11: *Listagem de bibliotecas*

A.6 Alinhamento de documentos de texto

Neste módulo podem ser rodados alinhamentos de pares e alinhamento múltiplo de documentos de texto. Ambos os módulos utilizam os conjuntos de parâmetros cadastrados no sistema. O módulo de alinhamento múltiplo depende do módulo de alinhamento de pares pois o usuário pode visualizar o alinhamento local para cada par de documentos no alinhamento múltiplo.

A.6.1 Alinhamento de pares de documentos

O módulo de alinhamento de pares de documentos pode ser acessado clicando na opção "Pairwise alignment" (figura A.23). A figura A.24 mostra a tela de configuração para um alinhamento local. Na caixa "Score Group" pode ser selecionado o conjunto de parâmetros. É possível também criar um conjunto de parâmetros sem necessidade de ir no módulo de Parâmetros de Alinhamento (figura A.25). Uma vez configurado o conjunto paramétrico, o usuário precisa selecionar uma biblioteca. Ao selecionar uma biblioteca o sistema carrega as duas caixas de seleção "Select document", onde o documento da esquerda é a referência e o documento da esquerda é o secundário. Finalmente, ao clicar no botão "Run" o sistema envia a requisição de alinhamento para o servidor. O servidor executa a requisição de maneira assíncrona e ao mesmo tempo mantém um canal aberto com a interface de usuário para informá-lo sobre o estado do alinhamento.

Uma vez obtido o alinhamento local, o sistema mostra uma tabela com um resumo das regiões em comum entre ambos os documentos. Para cada região, são apresentadas estatísticas sobre o número de igualdades, similaridades, disparidades e espaços (figura A.26).

Ao clicar em uma das regiões é mostrada uma janela que coloca ambos documentos para comparação e destaca as regiões em comum. Ao clicar em uma região de um documento o sistema mostra a região correspondente no outro documento. Essa janela mostra também, para uma região específica, as colunas do alinhamento local. Foram escolhidas cores para representar o tipo de coluna: verde escuro para igualdade, verde claro para similaridade, roxo para disparidade e vermelho para espaços (figura A.27).

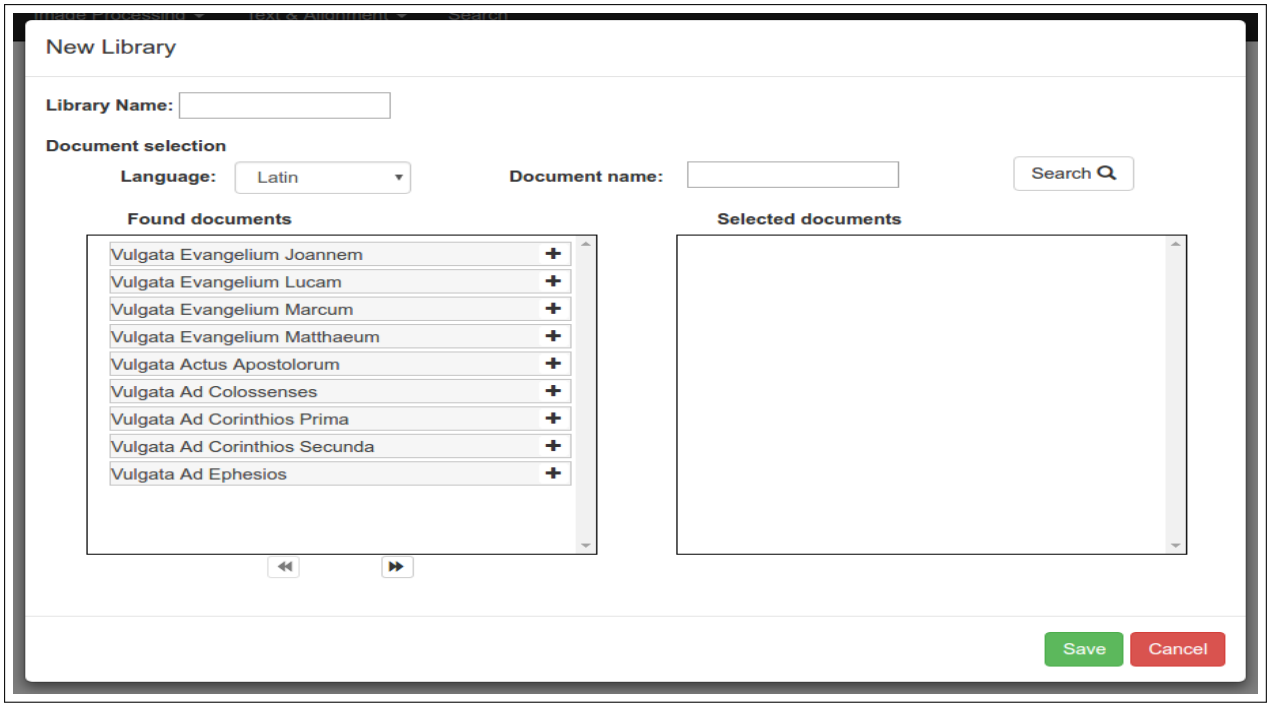


Figura A.12: *Adicionando nova biblioteca. A caixa de seleção "Language" e a caixa de texto "Document name" servem para filtrar os documentos de interesse. Ao filtrar usando estes controles, os resultados são mostrados na caixa "Found documents". Os documentos mostra nessa caixa podem ser selecionados clicando no botão "+". Ao serem selecionados eles são mostrados na caixa da direita "Selected documents". De forma análoga, documentos selecionados podem ser removidos clicando no botão "-" e mostrados de volta na caixa "Found documents".*

A.6.2 Alinhamento múltiplo de documentos

No caso do alinhamento múltiplo, ele pode ser acessado clicando na opção "Multiple alignment" (figura A.28). A configuração do alinhamento múltiplo é similar ao alinhamento de pares com a única diferença que, ao invés de selecionar um documento secundário, é necessário selecionar dois ou mais documentos secundários (figura A.29).

Da mesma forma que no alinhamento de pares de documentos, neste caso também é mostrado um resumo das regiões em comum entre o documento de referência e todos os documentos secundários. Para cada região, são mostradas as estatísticas sobre o número de colunas iguais, similares e díspares (figura A.30).

Ao clicar em uma região o sistema permite ao usuário selecionar entre dois modos de visualização: por colunas de alinhamento ou por regiões em comum. O primeiro modo de visualização mostra as regiões do documento de referência cobertas por regiões dos documentos secundários. Essa cobertura é mostrada coluna a coluna utilizando as mesmas cores que no alinhamento de pares mas com uma pequena variação de intensidade para distinguir os documentos envolvidos (figura A.31).

No segundo modo de visualização é uma extensão da comparação de documentos do alinhamento de pares. Neste caso, no entanto, são mostrados três documentos (figura A.32).

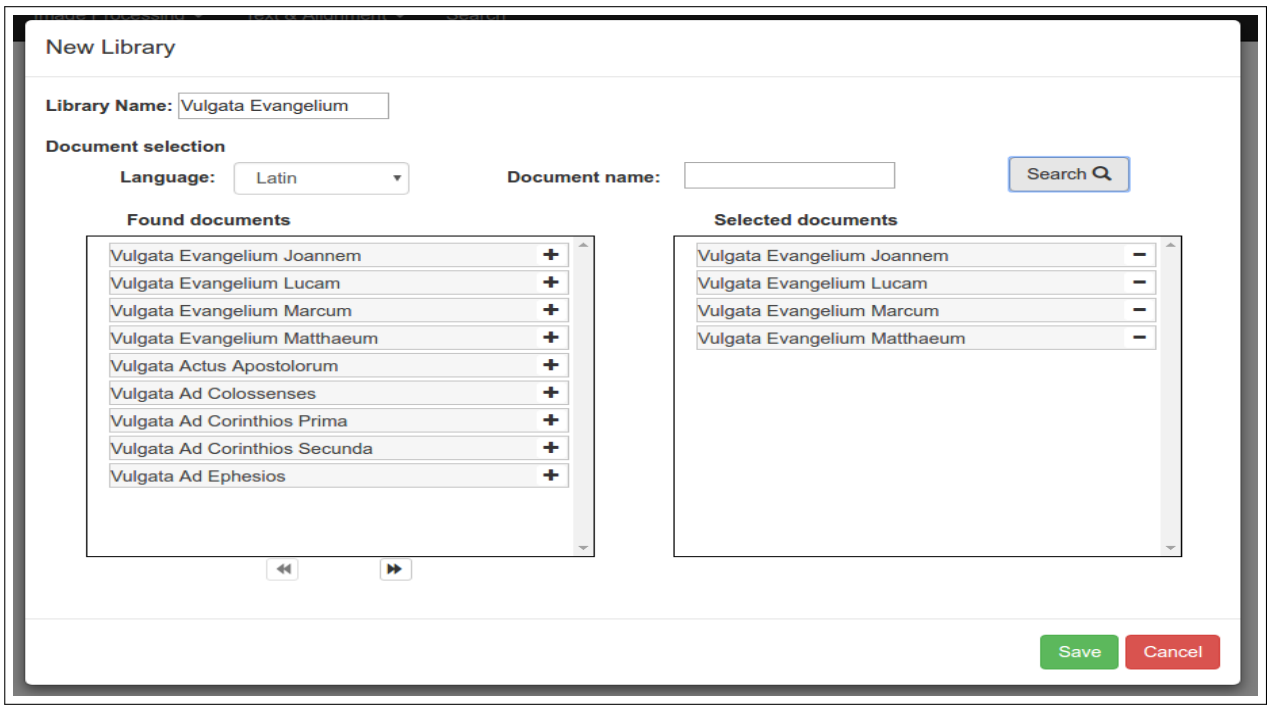


Figura A.13: *Uma biblioteca já configurada com documentos selecionados.*

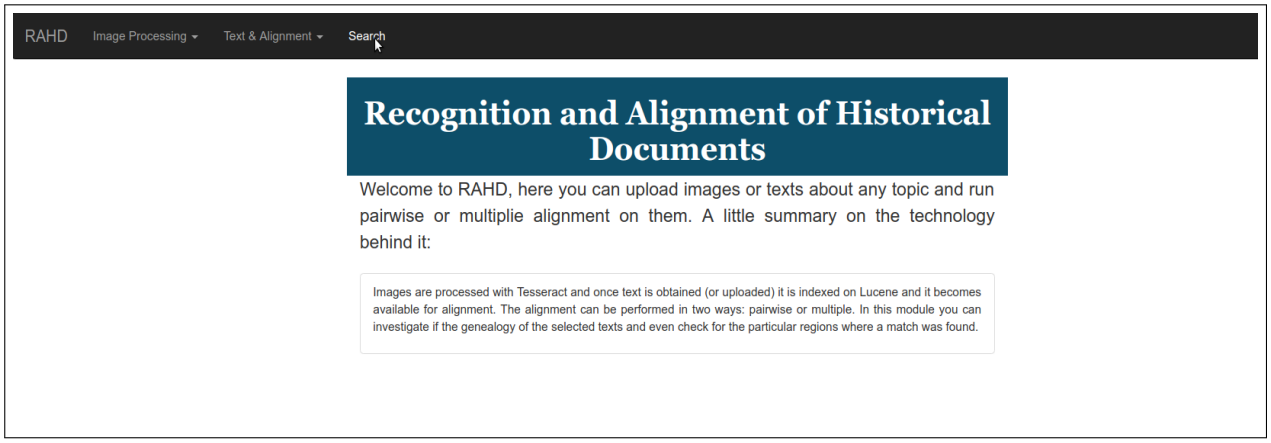


Figura A.14: *Acessando ao módulo de Busca*

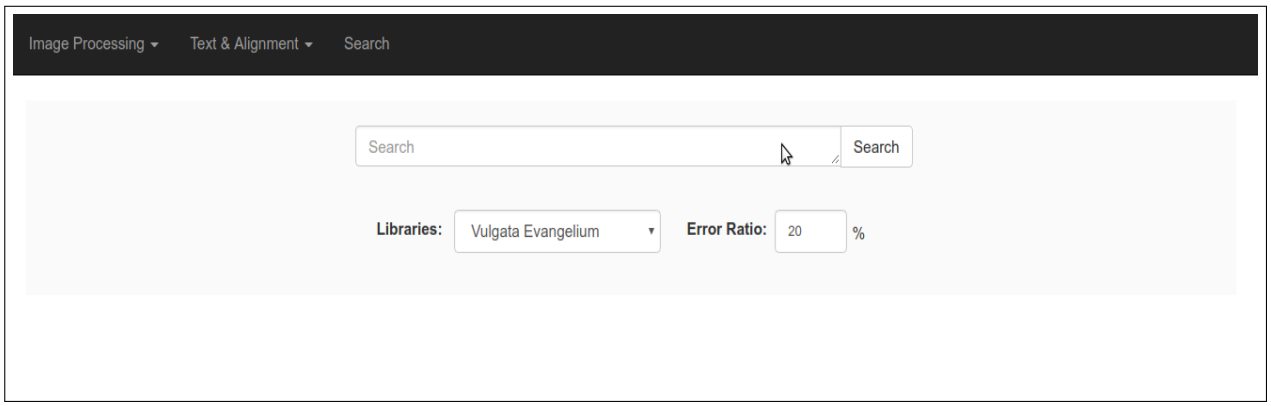


Figura A.15: *Página inicial de busca*

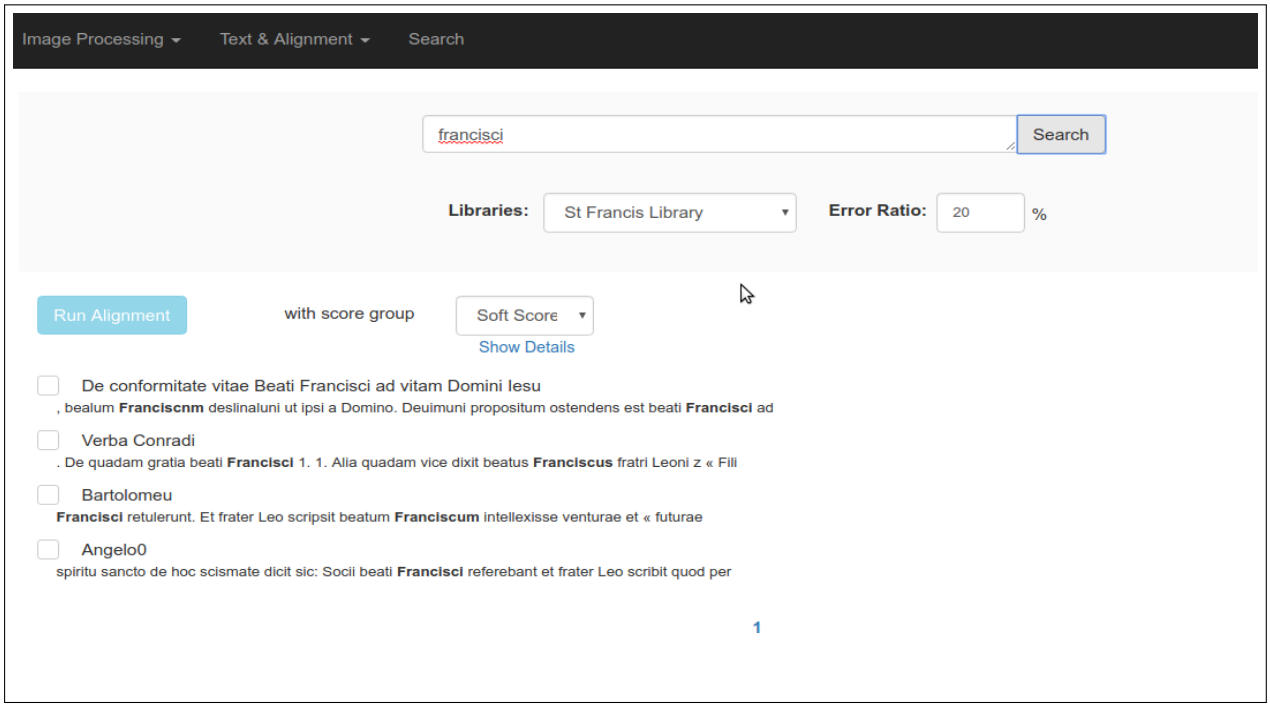


Figura A.16: *Página de busca com exemplo de consulta executada nos documentos do conjunto de dados descrito no capítulo 7. O sistema destaca em negrito as palavras encontradas.*

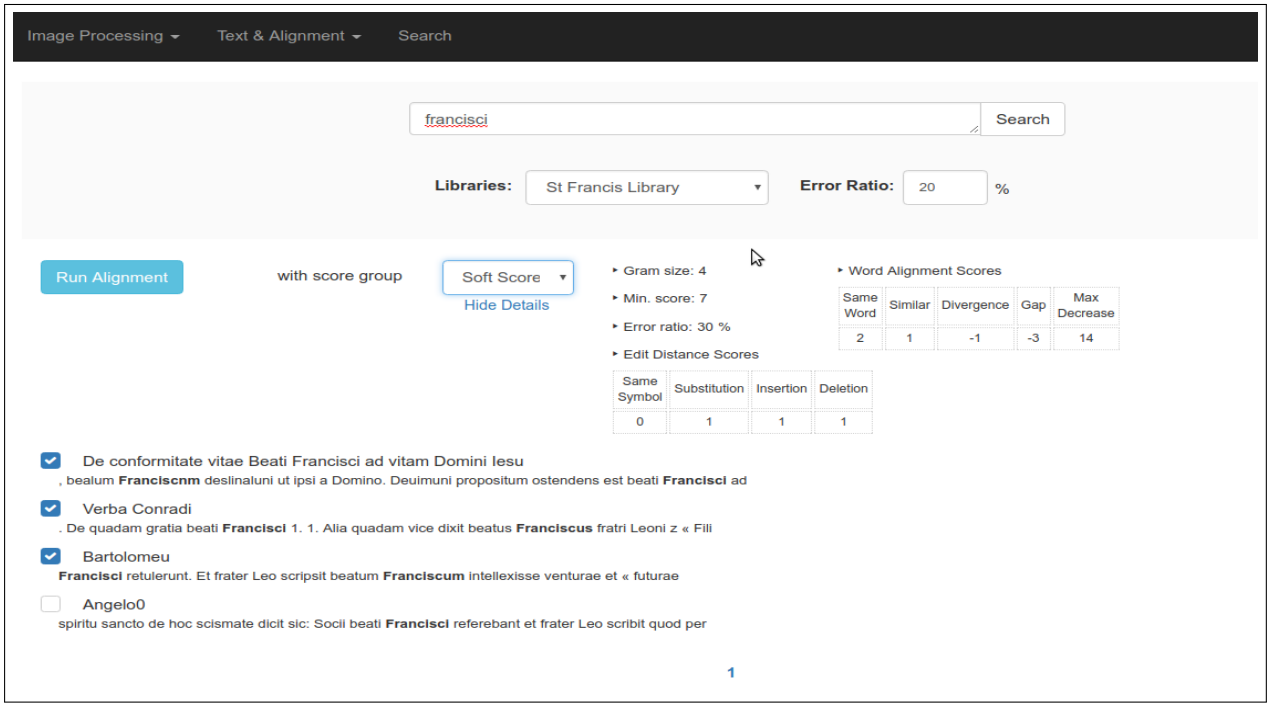


Figura A.17: *Selecionando documentos para realizar alinhamento múltiplo. A figura mostra também os detalhes do conjunto de parâmetros que será utilizado para essa execução.*

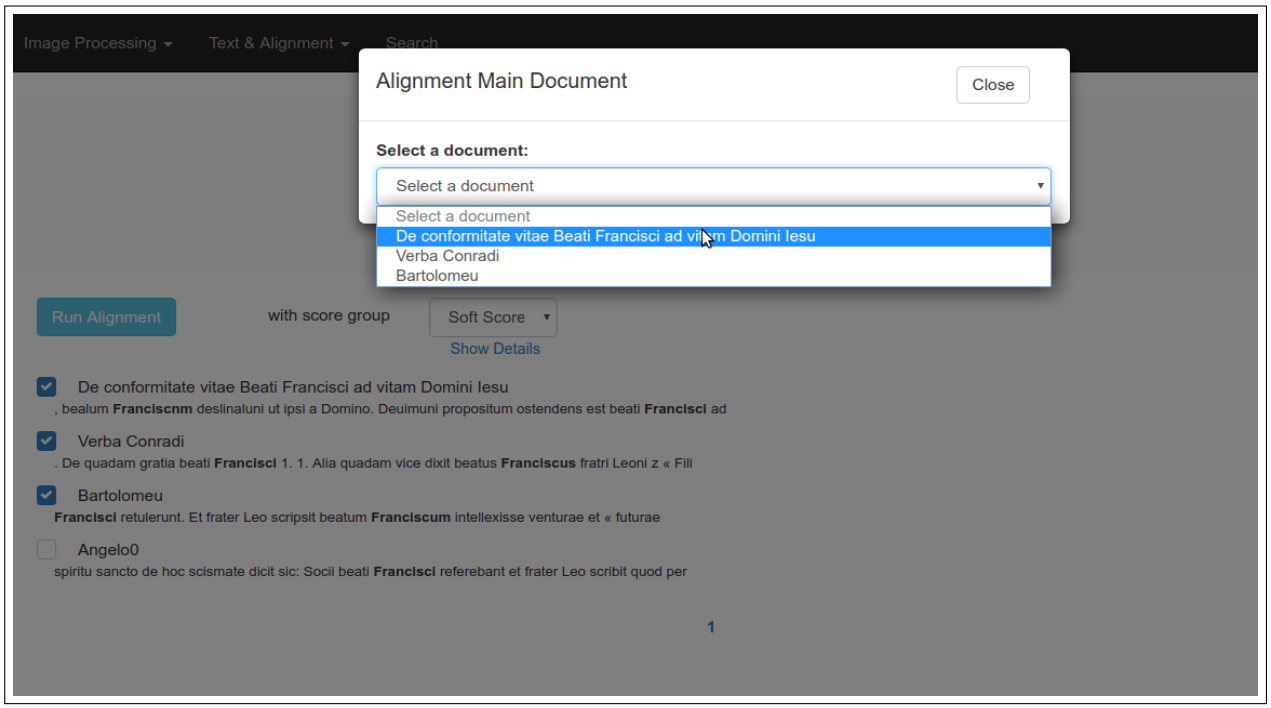


Figura A.18: *Ao clicar no botão "Run Alignment" o sistema mostra uma caixa de diálogo onde é selecionado o documento de referência do alinhamento. Dependendo do número de documentos selecionados, o sistema executa o alinhamento de pares ou alinhamento múltiplo.*

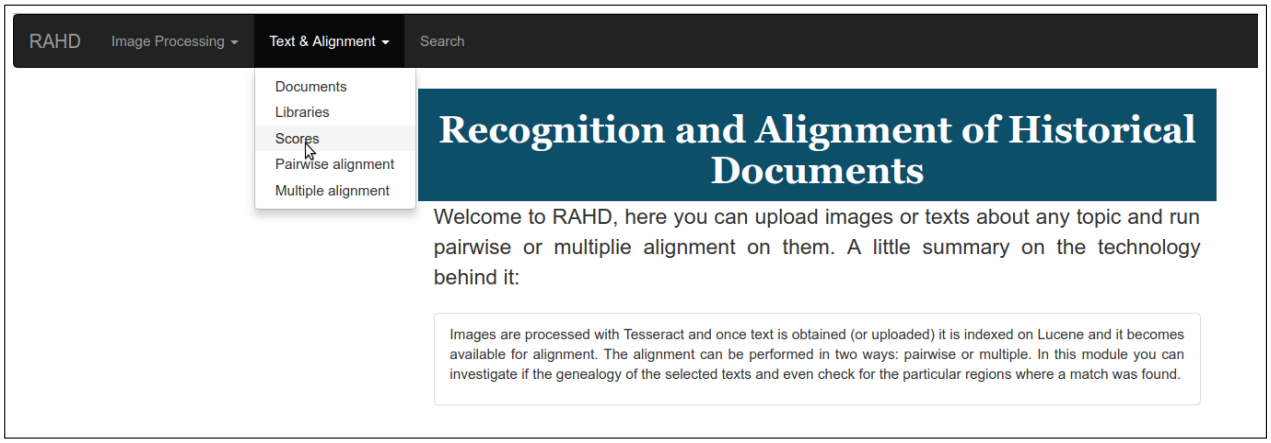


Figura A.19: *Acessando ao módulo de Parâmetros de Alinhamento*

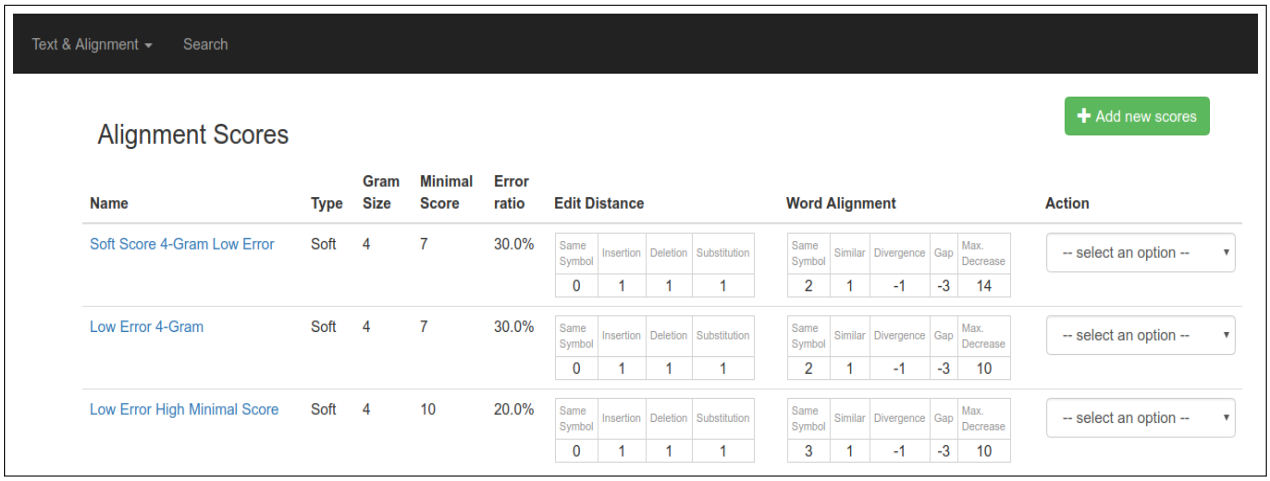


Figura A.20: *Listando parâmetros de alinhamento*

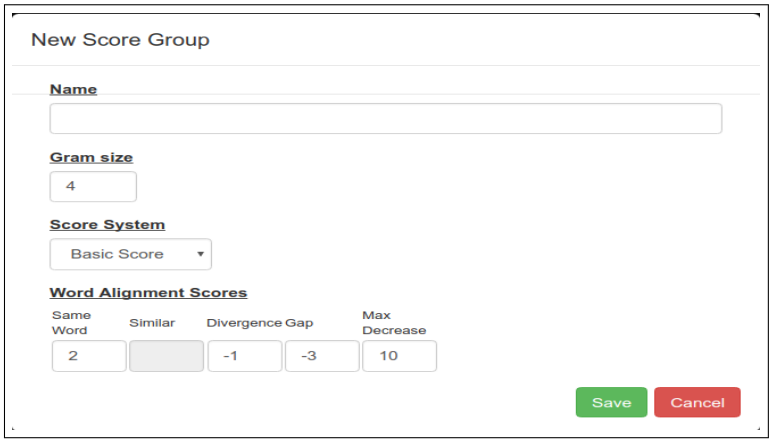


Figura A.21: *Adicionando parâmetros tipo "Basic"*

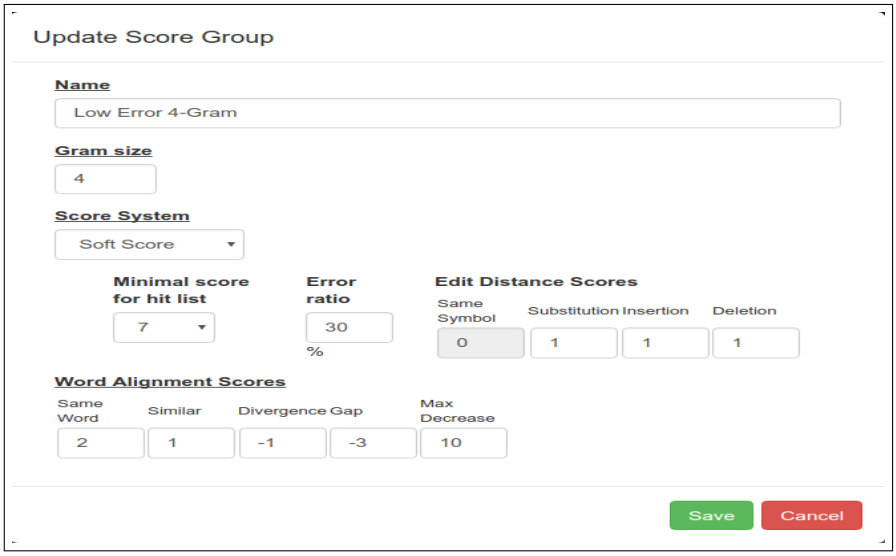


Figura A.22: *Modificando parâmetros tipo "Soft"*

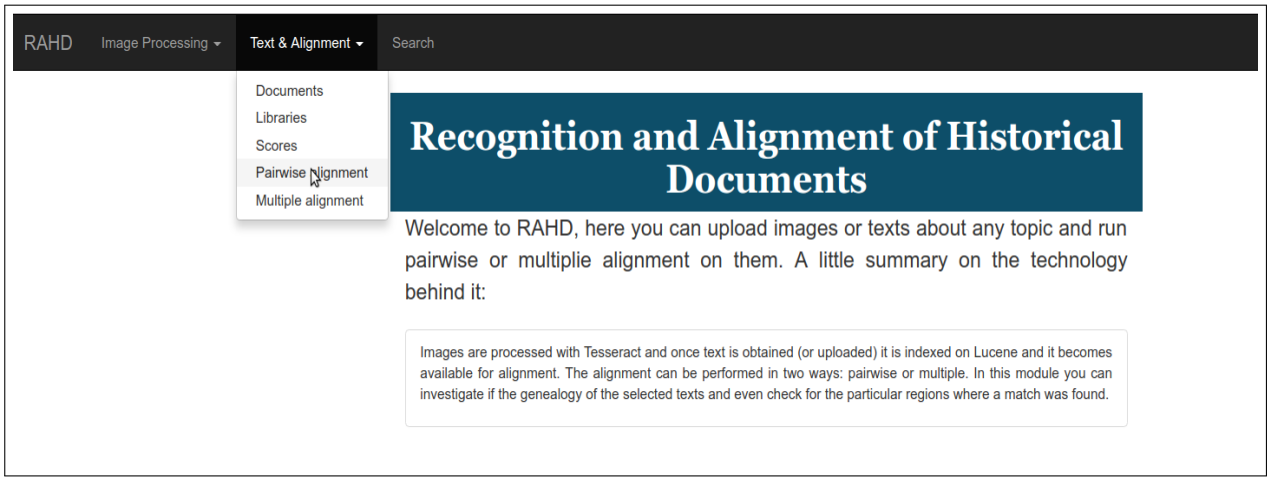


Figura A.23: *Acesso ao módulo de alinhamento de pares de documentos*

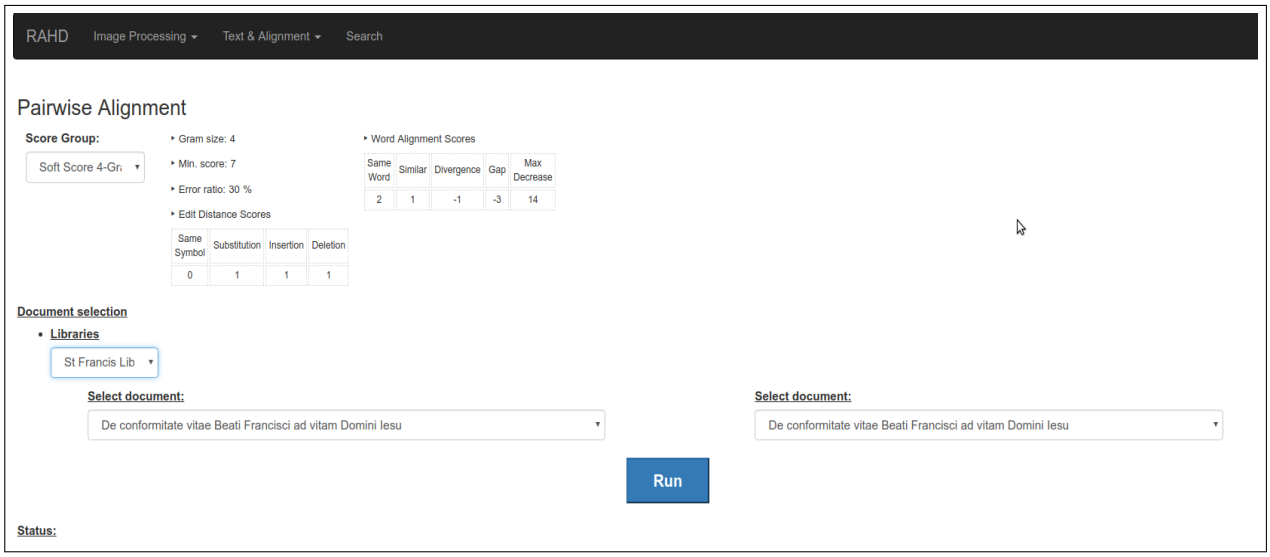


Figura A.24: *Configuração do alinhamento de pares de documentos*

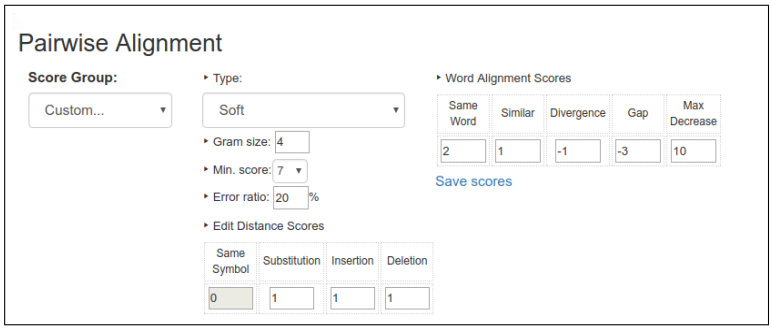


Figura A.25: *Conjunto de parâmetros personalizado para a execução do alinhamento de pares de documentos*

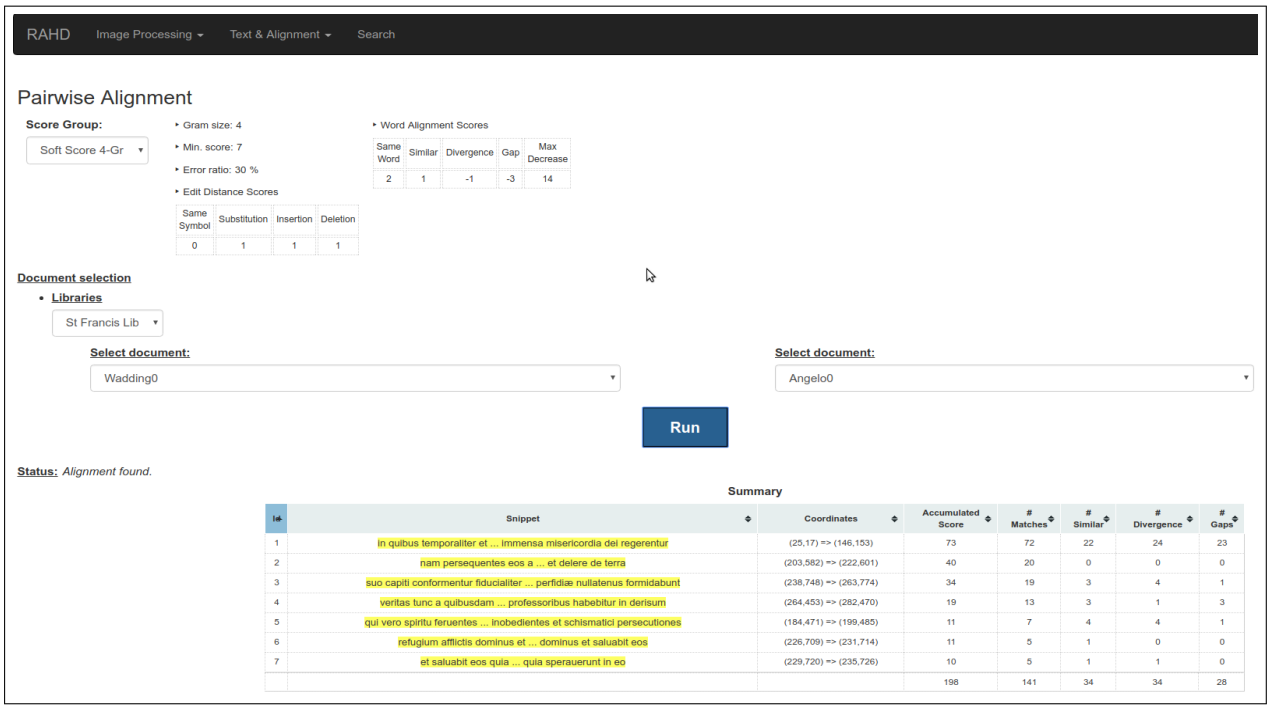


Figura A.26: *Resumo do alinhamento local de dois documentos. As linhas destacadas em amarelo correspondem com regiões em ambos documentos. À direita de cada região são mostradas as estatísticas.*

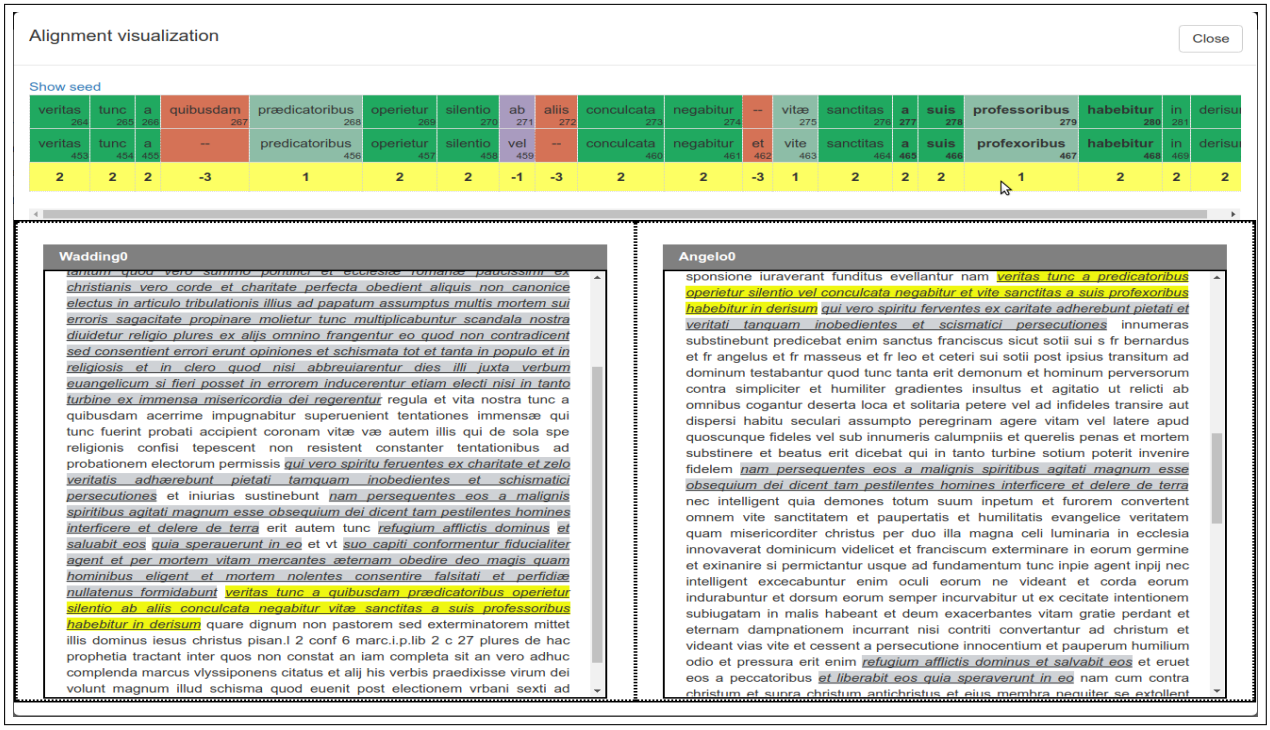


Figura A.27: *Visualização do alinhamento de um par de documentos. Os trechos de texto destacados em amarelo são regiões que ambos documentos tem em comum. Na parte superior é mostrada essa região coluna a coluna.*

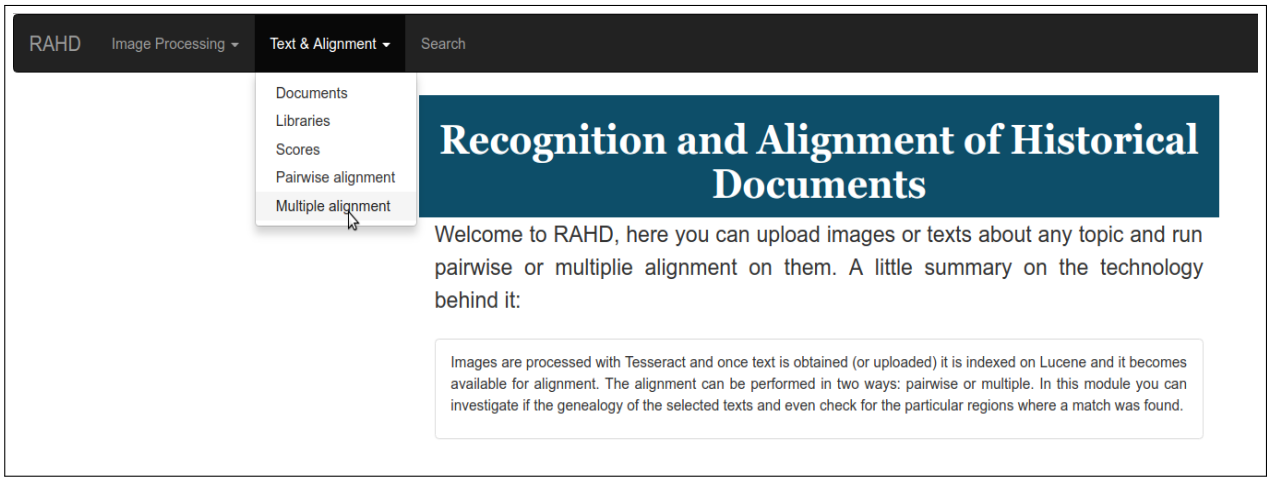


Figura A.28: *Acesso ao módulo de alinhamento múltiplo*

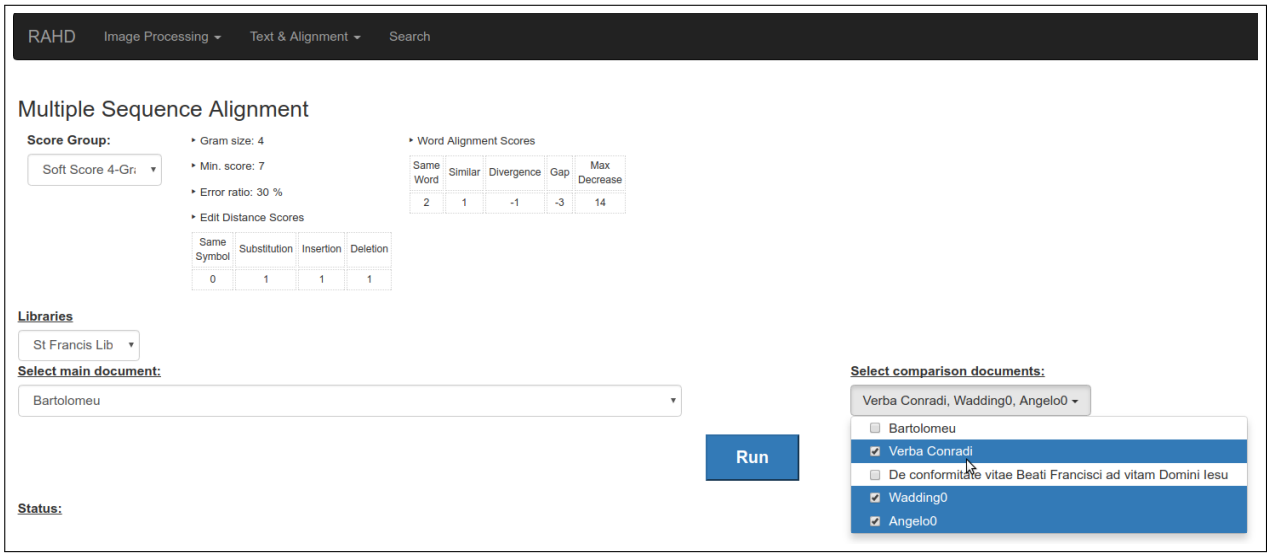


Figura A.29: *Configuração do alinhamento múltiplo de documentos*

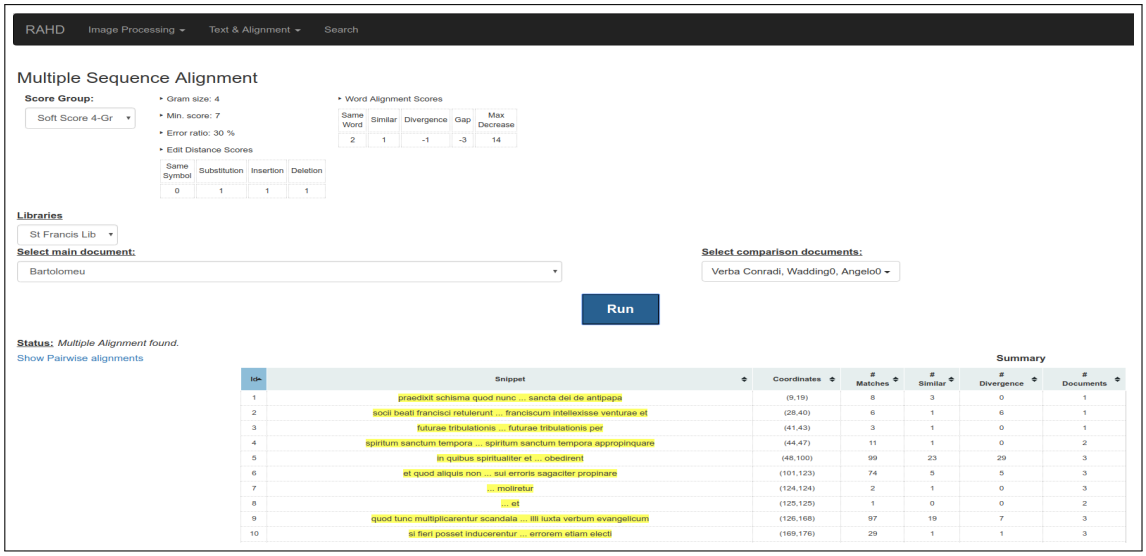


Figura A.30: *Saída do alinhamento múltiplo de documentos. O sistema mostra um resumo das regiões em comum assim como as suas estatísticas.*

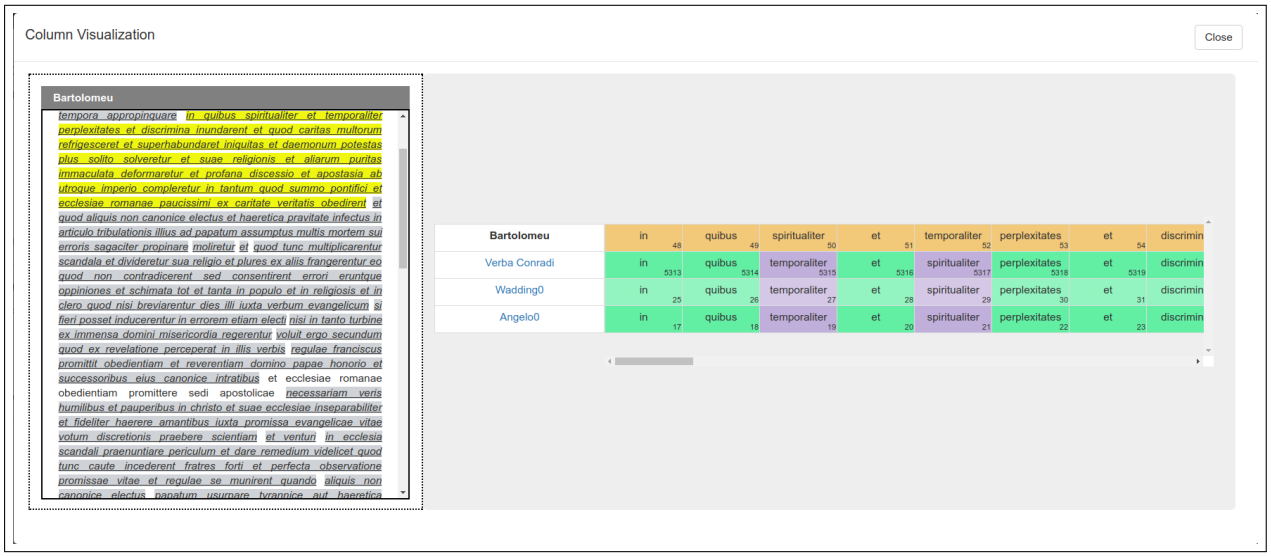


Figura A.31: *Visualização de alinhamento múltiplo por colunas. Ao selecionar uma região no documento de referência o sistema mostra as colunas na direita.*

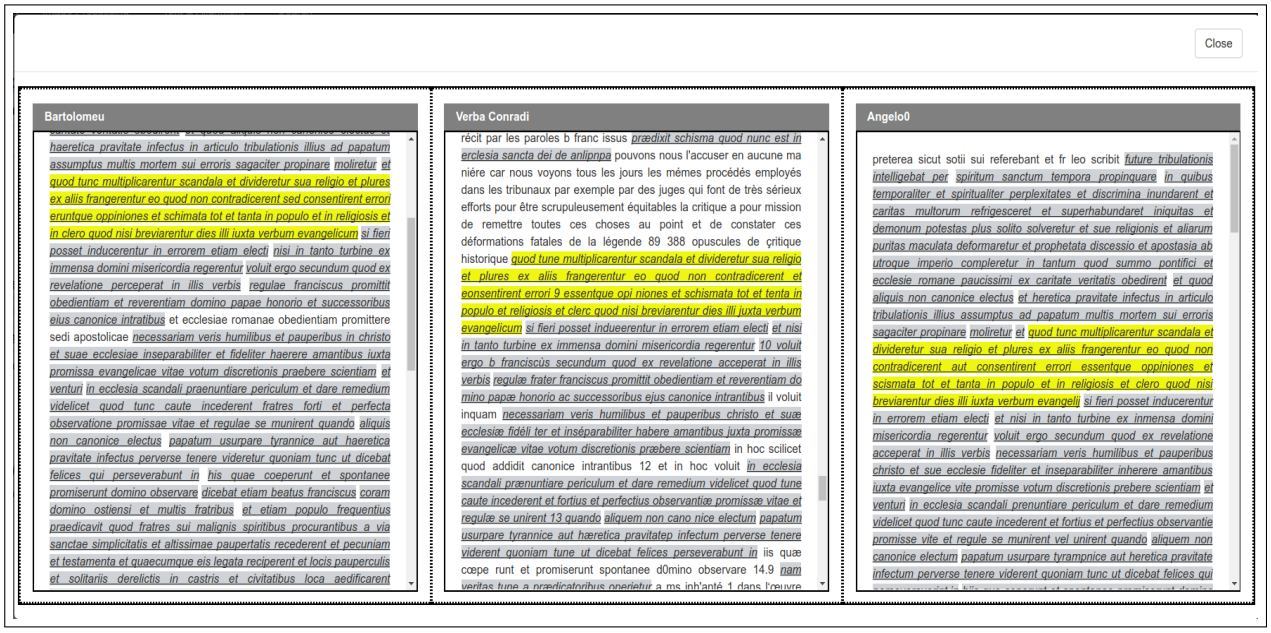


Figura A.32: *Visualização de alinhamento múltiplo por comparação de textos. Ao clicar em uma região em cinza o sistema ajusta os textos correspondentes para mostrar as regiões de texto correspondentes. Todas as regiões selecionadas recebem destaque em amarelo.*

Referências Bibliográficas

[AGM+90] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers e David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, oct 1990. 75

[Agu10] Veronica Aparecida Silveira Aguiar. *A construção da norma no movimento franciscano: Regulae e Testamentum nas práticas jurídicas mendicantes (1210-1323)*. Tese de Doutorado, Universidade de São Paulo, São Paulo, nov 2010. 113

[AHS99] Regis J. Armstrong, J. A. Wayne. Hellmann e William J. Short. *Francis of Assisi : Early documents Vol. 3*. New City Press, 1999. 113, 133

[AMS+97] S F Altschul, T L Madden, A A Schäffer, J Zhang, Z Zhang, W Miller e D J Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–402, sep 1997. 77, 78

[BAKD13] Ofer Biller, Abedelkadir Asi, Klara Kedem e Itshak Dinstein. WebGT: An Interactive Web-Based System for Historical Document Ground Truth Generation. Em *2013 12th International Conference on Document Analysis and Recognition*, páginas 305–308. IEEE, aug 2013. 1

[BBP11] Marcel Bollmann, Marcel Bollmann e Florian Petran. Rule-Based Normalization of Historical Texts. Em *Language Technologies for Digital Humanities and Cultural Heritage Workshop*, páginas 34–42, 2011. 4

[BCP02] Ilaria Bartolini, Paolo Ciaccia e Marco Patella. String Matching with Metric Trees Using an Approximate Distance. páginas 271–283. Springer, Berlin, Heidelberg, 2002. 61

[BL94] Vic. Barnett e Toby. Lewis. *Outliers in statistical data*. Wiley, 1994. 17

[BL03] Regina Barzilay e Lillian Lee. Learning to paraphrase. Em *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - NAACL '03*, volume 1, páginas 16–23, Morristown, NJ, USA, may 2003. Association for Computational Linguistics. 5

[Blu67] Harry Blum. A Transformation for Extracting New Descriptors of Shape. *Models for the Perception of Speech and Visual Form*, páginas 362 – 380, 1967. 13

[Bos09] Federico Boschetti. *A Corpus-based Approach to Philological Issues*. Tese de Doutorado, 2009. 4, 106

[Cat] Catholic Encyclopedia. Angelo Clareno. http://www.newadvent.org/cathen/01484b.htm. 112

[CD16] Pramit Chaudhuri e Joseph P. Dexter. Bioinformatics and Classical Literary Study. feb 2016. 5

[CKP+12] Neil Coffee, Jean-Pierre Koenig, Shakthi Poornima, Roelant Ossewaarde, Christopher Forstall e Sarah Jacobson. Intertextuality in the Digital Age. *Transactions of the American Philological Association*, 142(2):383–422, 2012. 1, 5

[Cla23] Angelo Clareno. Expositio regulae Fratrum Minorum. https://archive.org/details/expositioregulae00ange, 1323. 113

[Clea] Gregory Cleary. Blessed Conrad of Offida - Encyclopedia Volume - Catholic Encyclopedia - Catholic Online. http://www.catholic.org/encyclopedia/view.php?id=3270. 113

[Cleb] Gregory Cleary. Luke Wadding - Encyclopedia Volume - Catholic Encyclopedia - Catholic Online. http://www.catholic.org/encyclopedia/view.php?id=12167. 109

[Com89] Analytical Methods Committee. Robust statistics how not to reject outliers. Part 1. Basic concepts. *The Analyst*, 114(12):1693–1697, jan 1989. 17

[CW00] Yi-Kai Chen e Jhing-Fa Wang. Segmentation of single- or multiple-touching handwritten numeral string using background and foreground analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1304–1317, 2000. 38

[Dh214] Dh2014. DH 2014 : Conference Abstracts - Digital Humanities, 2014. 1

[Dh215] Dh2015. DH2015 Conference Abstracts - Digital Humanities Research Group, 2015. 1

[dL] A. Pereira do Lago. A batalha final entre a ciência e a anticiência. Trabalho em andamento. 6

[DL17] Mark Davis e Laurentiu Lancu. Unicode Text Segmentation. Relatório técnico, 2017. 56

[DN73] Margaret O. Dayhoff e National Biomedical Research Foundation (U.S.). *Atlas of protein sequence and structure. Supplement*. National Biomedical Research Foundation, 1973. 75

[FBN+14] Andreas Fischer, Horst Bunke, Nada Naji, Jacques Savoy, Micheal Baechler e Rolf Ingold. The HisDoc Project. Automatic Analysis, Recognition, and Retrieval of Handwritten Historical Documents for Digital Libraries. páginas 91–106, jan 2014. 1

[FFN+] Pamela Forner, Pamela Forner, Roberto Navigli, Dan Tufis (eds, Martin Potthast, Matthias Hagen, Tim Gollub, Johannes Kiesel, Paolo Rosso, Efstathios Stamatatos e Benno Stein. Overview of the 5th International Competition on Plagiarism Detection. 122

[FHSW04] Martin C Frith, Ulla Hansen, John L Spouge e Zhiping Weng. Finding functional sequence elements by multiple local alignment. *Nucleic acids research*, 32(1):189–200, jan 2004. 74

[Gar15] Angelika; Garz. HisDoc 2.0: Toward Computer-assisted Paleography, 2015. 1

[Gle15] University of Chicago Glenn Roe, Australian National University; Alfie Abdul-Rahman, University of Oxford; Min Chen, University of Oxford; Clovis Gladstone, University of Chicago; Robert Morrissey, University of Chicago; Mark Olsen. Visualizing Text Alignments: Image Processing Techniques for Locating 18th-Century Commonplaces | Glenn Roe and Clovis Gladstone - Academia.edu, 2015. 1, 5, 79

[Gli14] Demetrios Glinos. Discovering Similar Passages within Large Text Documents. Em *Information Access Evaluation. Multilinguality, Multimodality, and Interaction: 5th International Conference of the CLEF Initiative*, páginas 98–109. Springer, Cham, 2014. 5, 124

[GPP06] B Gatos, I Pratikakis e S J Perantonis. Adaptive degraded document image binarization. *Pattern Recognition*, 39(3):317–327, mar 2006. 26

[GRR+11] Annette Gotscharek, Ulrich Reffle, Christoph Ringlstetter, Klaus U. Schulz e Andreas Neumann. Towards information retrieval on historical document collections: the role of matching procedures and special lexica. *International Journal on Document Analysis and Recognition (IJDAR)*, 14(2):159–171, jun 2011. 4

[GRRS09] Annette Gotscharek, Ulrich Reffle, Christoph Ringlstetter e Klaus U. Schulz. On lexical resources for digitization of historical documents. Em *Proceedings of the 9th ACM symposium on Document engineering - DocEng '09*, página 193, New York, New York, USA, sep 2009. ACM Press. 1, 11

[GSLP14] Basilis Gatos, Nikolaos Stamatopoulos, Georgios Louloudis e Stavros Perantonis. H-DocPro. Em *Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage - DATeCH '14*, páginas 131–136, New York, New York, USA, 2014. ACM Press. 1

[Gus97] Dan Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997. 91

[GW08] Rafael C. Gonzales e Richard E. Woods. *Digital ImageProcessing*. 3ra. ediçã edição, 2008. 11, 12

[Hao15] Switzerland Hao Wei (hao.wei@unifr.ch), University of Fribourg, Switzerland and Kai Chen (kai.chen@unifr.ch), University of Fribourg, Switzerland

and Mathias Seuret (mathias.seuret@unifr.ch), University of Fribourg, Switzerland and Marcel Würsch (marcel.wuersch@unifr. DIVADIAWI - A Web-based Interface for Semi-automatic Labeling of Historical Document Images, 2015. 1

[Har62] Fr. Harold. *Fr. Lucae Waddingi, annalium minorum authoris, Vita*. Rome, epitome a edição, 1662. 109

[Har00] Johanna Sarah Hardin. *Multivariate Outlier Detection and Robust Clustering with Minimum Covariance Determinant Estimation and S-Estimation*. Tese de Doutorado, University of California - Davis, 2000. 22, 35

[HBK84] Douglas M. Hawkins, Dan Bradu e Gordon V. Kass. Location of Several Outliers in Multiple-Regression Data Using Elemental Sets. *Technometrics*, 26(3):197, aug 1984. 20

[HH92] S Henikoff e J G Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of the United States of America*, 89(22):10915–9, nov 1992. 75

[Hil69] C J Hilditch. Linear skeletons from square cupboards. páginas 403–420, jan 1969. 15, 36

[HM11] Iris Hendrickx e Rita Marquilhas. From old texts to modern spellings: an experiment in automatic normalisation. *JLCL*, 2011. 4

[HRA08] Mia Hubert, Peter J Rousseeuw e Stefan Van Aelst. High-Breakdown Robust Multivariate Methods. *Statistical Science*, 23(1):92–119, 2008. 21

[HRV12] Mia Hubert, Peter J. Rousseeuw e Tim Verdonck. A Deterministic Algorithm for Robust Location and Scatter. *Journal of Computational and Graphical Statistics*, 21(3):618–637, jul 2012. 21

[HS07] Andreas W. Hauser e Klaus U. Schulz. Unsupervised Learning of Edit Distance Weights for Retrieving Historical Spelling Variations. Em *Proceedings of the First Workshop on Finite-State Techniques and Approximate Search*, 2007. 4

[JTU96] Petteri Jokinen, Jorma Tarhio e Esko Ukkonen. A Comparison of Approximate String Matching Algorithms. *Software: Practice and Experience*, 26(12):1439–1458, dec 1996. 61

[Kay93] Lily E. Kay. *The Molecular Vision of Life: Caltech, the Rockefeller Foundation, and the Rise of the New Biology*. Monographs on the history and philosophy of biology. Oxford University Press, 1993. 6

[KFV12] Khurram Khurshid, Claudie Faure e Nicole Vincent. Word spotting in historical printed documents using shape and sequence comparisons. *Pattern Recognition*, 45(7):2598–2609, jul 2012. 1, 4, 11

[KGN+07] T. Konidaris, B. Gatos, K. Ntzios, I. Pratikakis, S. Theodoridis e S. J. Perantonis. Keyword-guided word spotting in historical printed documents using synthetic data and user feedback. *International Journal of Document Analysis and Recognition (IJDAR)*, 9(2-4):167–177, mar 2007. 27

[LC92] Hsi-Jian Lee e Bin Chen. Recognition of handwritten Chinese characters via short line segments. *Pattern Recognition*, 25(5):543–552, may 1992. 36

[LFS09] Timo Lassmann, Oliver Frings e Erik L L Sonnhammer. Kalign2: high-performance multiple alignment of protein and nucleotide sequences allowing external features. *Nucleic acids research*, 37(3):858–65, feb 2009. 74

[LI66] V. I. Levenshtein e V. I. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady, Vol. 10, p.707*, 10:707, 1966. 50

[Lis62] Frei Marcos De Lisboa. *Chronicas da ordem dos frades Menores*. Joannes Blauio de Colonza, 1557. Reimpressão feita em 2001 da edição de 1613, volume par edição, 1662. 111

[Lis14] Johan-Mattis List. *Sequence Comparison in Historical Linguistics*. Tese de Doutorado, 2014. 4

[Man64] Raoul Manselli. *Dizionario Biografico degli Italiani (DBI) VI*. 1964. 111

[Mar15] Jörg Ritter Marcus Pöckelmann, André Medek, Paul Molitor. CATview - Supporting The Investigation Of Text Genesis Of Large Manuscripts By An Overall Interactive Visualization Tool. *Digital Humanities*, 2015. 1

[Mas12] Alessandro Mastromatteo. *Similem illum fecit in gloria sanctorum : il profilo cristiforme di Francesco d'Assisi nel De conformitate di Bartolomeo da Pisa*. Antonianum, 2012. 112

[MMPR] Barbara Mcgillivray, Barbara Mcgillivray, Marco Passarotti e Paolo Ruffolo. The Index Thomisticus Treebank Project: Annotation, Parsing and Valency Lexicon. 81

[MMY06] Ricardo A. Maronna, R. Douglas Martin e Víctor J. Yohai. *Robust Statistics*. Wiley Series in Probability and Statistics. John Wiley & Sons, Ltd, Chichester, UK, mar 2006. 16

[MS99] Christopher D. Manning e Hinrich Schutze. Foundations of statistical natural language processing. jul 1999. 5, 79

[Nav97] Gonzalo Navarro. Multiple approximate string matching by counting. *Proceedings of the 4th South American Workshop on String Processing (WSP' 97)*, páginas 125–140, 1997. 61

[Nav01] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, mar 2001. 49, 50

[Nib85] Wayne Niblack. An introduction to digital image processing. oct 1985. 4

[Nim87] Duncan Nimmo. *Reform and division in the medieval Franciscan order*. Capuchin Historical Institute, Rome, 1987. 113

[NMG+10] Nikos Nikolaou, Michael Makridis, Basilis Gatos, Nikolaos Stamatopoulos e Nikos Papamarkos. Segmentation of historical machine-printed documents using Adaptive Run Length Smoothing and skeleton segmentation paths.

Image and Vision Computing, 28(4):590–604, apr 2010. 3, 11, 23, 24, 25, 36, 39, 46

[NW70] Saul B. Needleman e Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, mar 1970. 51, 70

[OHR] Mark Olsen, Russell Horton e Glenn Roe. Something Borrowed: Sequence Alignment and the Identification of Similar Passages in Large Text Collections. 5, 79

[Ots79] Nobuyuki Otsu. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979. 4

[Phi56] J. A. Philip. The Fragments of the Presocratic Philosophers. *Phoenix*, 10(3):116, 1956. 136

[Pis86] Bartolomeu De Pisa. *De conformitate vitae Beati Francisci ad vitam Domini Iesu*. Analecta Franciscana (reimpresso em Quaracchi em 1912. http://gallica.bnf.fr/ark:/12148/bpt6k1144828%7D, p.166-168), 1386. 111, 113

[Pis96] Bartolomeu De Pisa. De Vita et laudibus Beatae Mariae Virginis. https://books.google.com.br/books?id=LQRTAAAAcAAJ, 1596. 112

[PPCA13] Christos Papadopoulos, Stefan Pletschacher, Christian Clausner e Apostolos Antonacopoulos. The IMPACT dataset of historical document images. Em *Proceedings of the 2nd International Workshop on Historical Document Imaging and Processing - HIP '13*, página 123, New York, New York, USA, aug 2013. ACM Press. 1, 3

[PPLD12] Partha Pratim Roy, Umapada Pal, Josep Lladós e Mathieu Delalandre. Multi-oriented touching text character segmentation in graphical documents using dynamic programming. *Pattern Recognition*, 45(5):1972–1983, 2012. 4, 33, 39

[PS36] E. S. Pearson e C. Chandra Sekar. The Efficiency of Statistical Tools and A Criterion for the Rejection of Outlying Observations. *Biometrika*, 28(3/4):308, dec 1936. 20

[PSBCR10] Martin Potthast, Benno Stein, Alberto Barrón-Cedeño e Paolo Rosso. An evaluation framework for plagiarism detection. páginas 997–1005, aug 2010. 5

[PSdL+09] Pierre Peterlongo, Gustavo Akio Tominaga Sacomoto, Alair Pereira do Lago, Nadia Pisanti e Marie-France Sagot. Lossless filter for multiple repeats with bounded edit distance. *Algorithms for molecular biology : AMB*, 4(1):3, jan 2009. 74

[PV17] Gaspar Pizarro V. e Juan D. Velásquez. Docode 5: Building a real-world plagiarism detection system. *Engineering Applications of Artificial Intelligence*, 64:261–271, sep 2017. 5

[Rea97] Giovanni Reale. *Para uma nova interpretacão de Platão*. Edicões Loyola, 1997. 136

[RHM12] Martin Reynaert, Iris Hendrickx e Rita Marquilhas. Historical spelling normalization. A comparison of two statistical methods: TICCL and VARD2. Em *Second Workshop on Annotation of Corpora for Research in the Humanities*, 2012. 4

[RL87] Peter J. Rousseeuw e Annick M. Leroy. *Robust Regression and Outlier Detection*. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., Hoboken, NJ, USA, oct 1987. 20

[Rob] Udi Manber Robert Muth. Approximate Multiple String Search. 74

[Rou84] Peter J. Rousseeuw. Least Median of Squares Regression. *Journal of the American Statistical Association*, 79(388):871–880, dec 1984. 20

[RvD99] Peter J. Rousseeuw e Katrien van Driessen. A Fast Algorithm for the Minimum Covariance Determinant Estimator. *Technometrics*, 41(3):212, aug 1999. 21

[RVGF14] Jan Richarz, Szilard Vajda, Rene Grzeszick e Gernot A. Fink. Semi-supervised learning for character recognition in historical archive documents. *Pattern Recognition*, 47(3):1011–1020, mar 2014. 4

[RvZ91] Peter J. Rousseeuw e Bert C. van Zomeren. Robust Distances: Simulations and Cutoff Values. páginas 195–203. Springer, New York, NY, 1991. 22

[Sab03] Paul Sabatier. Opuscules de Critique Historique. https://hdl.handle.net/2027/msu.31293010410953. páginas(370-391), 1903. 106, 113, 114, 121

[San72] D Sankoff. Matching sequences under deletion-insertion constraints. *Proceedings of the National Academy of Sciences of the United States of America*, 69(1):4–6, jan 1972. 51

[Seb04] G. A. F. (George Arthur Frederick) Seber. *Multivariate observations*. Wiley-Interscience, 2004. 20

[Sho15] William J. Short. The "Liber conformitate"of Bartholomew of Pisa and its Sibylline and Prophetic Literature. *Carthaginensia*, 31:881–899, 2015. 112

[SM97] João Carlos Setubal e João Meidanis. *Introduction to Computational Molecular Biology*. PWS Pub., 1997. 69, 70, 71, 73

[Smi07] R. Smith. An Overview of the Tesseract OCR Engine. páginas 629–633, sep 2007. 99, 100, 101

[Smi11] Ray Smith. Limits on the Application of Frequency-Based Language Models to OCR. Em *2011 International Conference on Document Analysis and Recognition*, páginas 538–542. IEEE, sep 2011. 1, 11

[Son] Image Processing, Analysis and Machine Vision | Milan Sonka | Springer. 11, 12, 14, 15

[SPD10] Xiaolu Sun, Liangrui Peng e Xiaoqing Ding. Touching character segmentation method for Chinese historical documents. páginas 75340D–75340D–8, 2010. 33

[SPGS15] Miguel A Sanchez-Perez, Alexander Gelbukh e Grigori Sidorov. Dynamically Adjustable Approach through Obfuscation Type Recognition. Em *PAN at CLEF 2015*, 2015. 5

[SRPE06] Siwei Shen, Dragomir R. Radev, Agam Patel e Gunes Erkan. Adding syntax to dynamic programming for aligning comparable texts for the generation of paraphrases. jul 2006. 5

[SSW88] P.K Sahoo, S Soltani e A.K.C Wong. A survey of thresholding techniques. *Computer Vision, Graphics, and Image Processing*, 41(2):233–260, feb 1988. 13

[Sti77] Stephen M. Stigler. Do Robust Estimators Work with Real Data? *The Annals of Statistics*, 5(6):1055–1098, nov 1977. 18

[SW81] T.F. Smith e M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, mar 1981. 5, 72

[SW87] Smith e Raymond Wensley. The extraction and recognition of text from multimedia document images, 1987. 99

[SWA03] Saul Schleimer, Daniel S. Wilkerson e Alex Aiken. Winnowing: local algorithms for document fingerprinting. Em *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data - SIGMOD '03*, página 76, New York, New York, USA, 2003. ACM Press. 125

[The15] Alexei Lavrentiev Theodore Bluche, Dominique Stutzmann. From Text and Image to Historical Resource: Text-Image Alignment for Digital Humanists | Theodore Bluche, Dominique Stutzmann, and Alexei Lavrentiev - Academia.edu, 2015. 1

[TM] C. Tomasi e R. Manduchi. Bilateral filtering for gray and color images. Em *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, páginas 839–846. Narosa Publishing House. 26

[TMR13] Diego A. Rodríguez Torrejón, José Manuel e Martín Ramos. Text Alignment Module in CoReMo 2.1. Em *PAN at CLEF 2013*, 2013. 5

[Ukk85] Esko Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64(1-3):100–118, jan 1985. 52, 53

[Vin68] T. K. Vintsyuk. Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57, 1968. 51

[Wad23] Luke Wadding. Beati Patris Francisci Assisiatis Opera Omnia. http://books.google.com.br/books?id=oHByjBSC5BwC, 1623. 110, 111, 113

[Wis93] Michael J. Wise. String similarity via Greedy Strin g Tiling and running Karp-Rabin matching. Relatório técnico, University of Sydney, 1993. 125

[WSR+17] Maciej Wielgosz, Pawel Szczepka, Pawel Russek, Ernest Jamro e Kazimierz Wiatr. Evaluation and implementation of n-gram based algorithm for fast text comparison. *Computer and Informatics*, 36:887–907, 2017. 125

[WWC82] Friedrich M. Wahl, Kwan Y. Wong e Richard G. Casey. Block segmentation and text extraction in mixed text/image documents. *Computer Graphics and Image Processing*, 20(4):375–390, dec 1982. 23

[ZEP10] Konstantinos Zagoris, Kavallieratou Ergina e Nikos Papamarkos. A Document Image Retrieval System. *Engineering Applications of Artificial Intelligence*, 23(6):872–879, sep 2010. 3, 4