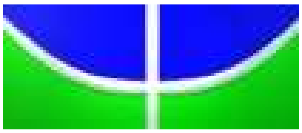


UNB – UNIVERSIDADE DE BRASÍLIA

FACE – Faculdade de Economia, Administração, Contabilidade e Ciência da Informação e Documentação

CID – Departamento de Ciência da Informação e Documentação

PPGCInf – Programa de Pós-Graduação em Ciência da Informação

AUTO TAVARES DA CÂMARA JÚNIOR

INDEXAÇÃO AUTOMÁTICA DE ACÓRDÃOS POR MEIO DE PROCESSAMENTO DE LINGUAGEM NATURAL

Brasília – DF

2007

AUTO TAVARES DA CÂMARA JÚNIOR

INDEXAÇÃO AUTOMÁTICA DE ACÓRDÃOS POR MEIO DE PROCESSAMENTO DE LINGUAGEM NATURAL

Dissertação apresentada à banca examinadora como requisito parcial à obtenção do Título de Mestre em Ciência da Informação pelo Programa de Pós-Graduação em Ciência da Informação do Departamento de Ciência da Informação e Documentação da Universidade de Brasília.

Orientadora: Profª. Drª. Marisa Bräscher Basílio Medeiros

Brasília – DF

2007

Folha de Aprovação

Título: **Indexação Automática de Acórdãos por Meio de Processamento de Linguagem Natural**

Autor: **Auto Tavares da Câmara Júnior**

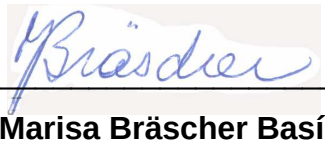
Área de Concentração: **Transferência da Informação**

Linha de Pesquisa: **Arquitetura da Informação**

Dissertação submetida à Comissão Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Ciência da Informação do Departamento de Ciência da Informação e Documentação da Universidade de Brasília como requisito parcial para obtenção do titulo de **Mestre em Ciência da Informação**.

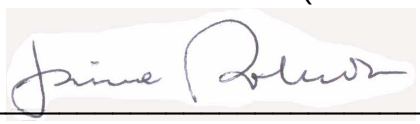
Dissertação aprovada em: 11 de junho de 2007

Aprovada por:



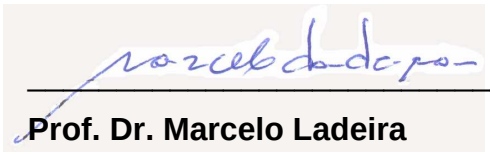
Profª. Drª. Marisa Bräscher Basílio Medeiros

Presidente – Orientadora (UnB/PPGCInf)



Prof. Dr. Jaime Robredo

Membro Interno – (UnB/PPGCInf)



Prof. Dr. Marcelo Ladeira

Membro Externo – (UnB/CIC)

Prof. Dr. Rogério Henrique de Araújo Júnior

Suplente – (UnB/PPGCInf)

Esta pesquisa é dedicada àqueles que acreditam e trabalham pela aproximação da Ciência da Informação à Ciência da Computação

Agradecimentos

À Profª. Drª. Marisa Bräscher, minha querida e paciente orientadora, pela orientação segura nos tortuosos e longos caminhos até a conclusão desta.

Aos professores cujas disciplinas tive o prazer e privilégio de cursar, Marisa Bräscher, Mamede Lima-Marques, Antônio Miranda, Suzana Mueller, Marcelo Ladeira. Vocês efetivamente abriram meus horizontes e fascinaram-me com novas e empolgantes discussões.

Ao meu irmão e irmãs que simplesmente me completam.

À Camila Ramos, meu amor, minha amiga, minhas razões, minha vida.

Ao Tribunal de Justiça do Distrito Federal e dos Territórios pelo fornecimento da base de dados de jurisprudência para realização da pesquisa.

Aos meus queridos amigos do TJDFT Sérgio, Marcelo, Vanessa, Gláucia, Fernando Dias, Fernando Dutra e ao Subsecretário de Desenvolvimento de Sistemas Ivonnilson Guimarães, os quais sobreviveram e apoiaram as minhas infinitas faltas até a conclusão desta.

À Maíra Murrieta que me trouxe a este Departamento originando todo o processo que culminou nesta pesquisa.

A todos aqueles que, direta ou indiretamente, contribuíram para a conclusão desta.

Resumo

A indexação é uma área pesquisada há muito tempo, ferramenta fundamental para a qualidade da recuperação da informação. Indexação automática é uma área mais recente cujos resultados melhoram com a evolução de técnicas de inteligência artificial. Com o recorrente crescimento de demanda judicial, os tribunais do País, em particular o Tribunal de Justiça do Distrito Federal e dos Territórios, têm de buscar mecanismos, em sua maioria apoiados em tecnologia da informação, para acelerar seus trâmites. A indexação automática oferece ganhos estratégicos aos tribunais, diminuindo seu custo com pessoal, melhorando a qualidade das pesquisas e acelerando seus processos organizacionais. Esta pesquisa insere-se nesta área e propõe um modelo de indexação automática de acórdãos que atenda a essas necessidades. A utilização de PLN sobrepõe-se aos métodos estritamente estatísticos com o objetivo de alcançar melhores resultados. Para isso, foram desenvolvidas ferramentas que constroem um *corpus* de língua portuguesa com jargão jurídico e indexam automaticamente uma base de jurisprudência de direito penal do TJDFT. Os resultados demonstram que a indexação automática utilizando PLN atinge níveis de revocação e precisão equivalentes à base indexada manualmente. Entre as contribuições dessa pesquisa encontram-se as ferramentas desenvolvidas, a metodologia de indexação automática proposta e o estudo sobre uma área muito pouco pesquisada em língua portuguesa.

Palavras-Chave: Indexação Automática; Indexação Manual; Recuperação da Informação; Processamento de Linguagem Natural; Jurisprudência.

Abstract

Indexing is an area that has been researched for a long time, crucial tool to information retrieval quality. Automatic indexing is a more recent area, in which results are improved with the evolution of artificial intelligence techniques. As judicial claims increase repeatedly, Country's courthouses, Tribunal de Justiça do Distrito Federal e dos Territórios in particular, have to search for mechanisms, in their majority based on information technology, to speed up their workflows. Automatic indexing offers strategic gains to courthouses, diminishing their personnel costs, improving research quality and accelerating their organizational processes. This research inserts itself in this area and proposes a sentence automatic indexing model which fulfills these needs. NLP utilization is chosen instead of strictly statistic methods with the purpose of achieving better results. In order to do so, tools have been developed which build a *corpus* of portuguese language containing common juridical vocabulary and automatically index a criminal law jurisprudence base from TJDFT. Results demonstrate that automatic indexing using NLP obtains equivalent revocation and precision levels than manual indexing. Between this research contributions are mentioned the developed tools, the automatic indexing methodology proposed and the study of a very little researched area in portuguese language.

Keywords: Automatic Indexing; Manual Indexing; Information Retrieval; Natural Language Processing; Jurisprudence.

Lista de Figuras

Lista de Quadros

Lista de Siglas

BOW – *Bag of Words* – Conjunto de Palavras

CPC – Código Processual Civil

IA – Inteligência Artificial

NILC – Núcleo Interinstitucional de Lingüística Computacional

NIST – *National Institute of Standards and Technology* – Instituto Nacional de Padrões e Tecnologia

PLN – Processamento de Linguagem Natural

STF – Supremo Tribunal Federal

STJ – Superior Tribunal de Justiça

TI – Tecnologia da Informação

TIC – Tecnologia da Informação e Comunicação

TJ – Tribunal de Justiça

TJDFT – Tribunal de Justiça do Distrito Federal e dos Territórios

TREC – Annual Text REtrieval Conference – Conferência anual americana de recuperação de texto do NIST

SUMÁRIO

INTRODUÇÃO

A construção de índices para recuperação de documentos de uma base de dados é uma área pesquisada há muito tempo. Diversos autores propõem metodologias e técnicas para indexação. Essa é uma ferramenta fundamental para a qualidade da recuperação de informação no aspecto que ela procura reduzir a quantidade de itens irrelevantes retornados em uma pesquisa, aumentando-se consequentemente os relevantes. A indexação automática, em particular, também tem sido pesquisada sob vários objetivos, entre eles a seleção de melhores descritores, a melhoria do desempenho de produção dos mesmos e, em última instância, a solução dos problemas encontrados na indexação manual, tais como a inserção da subjetividade do indexador, por exemplo. A evolução dos estudos e resultados oferecidos pela Inteligência Artificial tem fornecido suporte e aprimorado as técnicas de indexação automática. Resultados promissores estão sendo obtidos principalmente considerando que a quantidade de documentos eletrônicos produzidos aumenta substancialmente fornecendo, dessa forma, massa crítica e estímulo para pesquisas.

Estatísticas oficiais de crescimento da prestação jurisdicional no Brasil apresentam um quadro de ascendência constante. Entende-se por prestação jurisdicional o atendimento que o órgão de Justiça local, quer seja uma Comarca, Fórum, Tribunal de Justiça ou mesmo um Tribunal Superior, oferece à demanda da população por resoluções judiciais. No Distrito Federal, essa percepção não é diferente. Tanto em primeira quanto em segunda instância, o aumento do tamanho da população implica necessariamente no aumento da busca da Justiça.

Essa pesquisa tem por objetivo analisar essa situação. Com o crescimento da procura por Justiça, os tribunais precisam construir métodos de trabalho que aumentem sua eficiência, a fim de não verem seus recursos esgotados e sua burocracia estancar seus resultados. Uma metodologia de indexação automática de acórdãos surge como proposta para aceleração dos trâmites jurisprudenciais, fornecimento de informação precisa e completa para embasamento

de decisões futuras e, em última instância, liberação de recursos humanos envolvidos em tarefas mecânicas para fins mais intelectualmente desafiantes.

Os documentos que foram utilizados para o estudo são os que compõem a jurisprudência do Tribunal de Justiça do Distrito Federal e Territórios. O estudo utiliza os acórdãos de Direito Penal do Tribunal, formando assim uma base controlada de documentos. Esses documentos foram analisados e indexados automaticamente por meio de ferramentas construídas com funcionalidades de processamento de linguagem natural. A comparação dos resultados dessa indexação com a indexação manual realizada tradicionalmente foi o balizador para avaliação da metodologia e das ferramentas.

A dissertação resultado da pesquisa organiza-se por meio de capítulos introdutórios cujo objetivo consiste na apresentação do tema da pesquisa, além da definição de seus objetivos, justificativas e delimitação. Após essa ambientação, capítulos de referência teórica são descritos para apresentar os pressupostos teóricos utilizados para a construção dos resultados. Discutem-se a tramitação processual na Justiça Brasileira bem como a definição dos acórdãos. Além disso, estudos anteriores sobre indexação e indexação automática são recuperados para demonstrar o que tem sido pesquisado nessas áreas. Procura-se, por fim, fundamentar o processamento de linguagem natural para justificar a escolha dos componentes utilizados nas ferramentas.

A estruturação metodológica do estudo é, por conseguinte, explicitada. A definição do tipo da pesquisa, bem como a delimitação da amostra, são descritas cuidadosamente. A explicação dos instrumentos utilizados e dos procedimentos adotados é foco dos capítulos seguintes. Em seqüência seguem-se análise e interpretação dos resultados, os quais, por fim, são remetidos na conclusão do trabalho. Essa analisa as limitações da pesquisa e sugere trabalhos futuros, os quais podem basear-se nos resultados atingidos até então para alcance de objetivos maiores. Uma lista de referência é encadeada ao final. Como anexo, encontra-se todo o código fonte das ferramentas desenvolvidas.

Entre os produtos gerados por esta pesquisa encontram-se as ferramentas de indexação automática construídas, um corpus em língua portuguesa com jargão jurídico e uma metodologia definida de indexação automática de documentos. Objetiva-se, por fim, acrescentar resultados a uma área pouco pesquisada atualmente, qual seja o processamento de linguagem natural em língua portuguesa para linguagem jurídica.

1. FORMULAÇÃO DA SITUAÇÃO PROBLEMA

O Quadro 1 que se segue demonstra a evolução da quantidade de processos iniciados, julgados e arquivados na segunda instância do Tribunal de Justiça do Distrito Federal e Territórios entre os anos de 1998 e 2006, até o mês de Outubro.

Ano	Processos Autuados	Processos Julgados	Processos Baixados
1998	9.996	9.899	9.231
1999	12.363	10.498	9.250
2000	13.907	12.444	11.280
2001	18.644	16.128	12.470
2002	22.669	19.201	16.167
2003	26.155	17.778	20.845
2004	29.506	21.385	25.538
2005	31.830	29.575	33.875
2006*	34.924	24.158	31.030
Total	**199.994**	**161.066**	**169.686**

Quadro 1 – Evolução da Quantidade de Processos no Distrito Federal

Fonte: Tribunal de Justiça do Distrito Federal e Territórios

* Até o dia 31/10/2006

Percebe-se, portanto, que o crescimento da quantidade de processos que entra no Tribunal tem sido bastante expressivo. Observa-se, também, que o número de processos julgados apresenta crescimento ano a ano, porém não no mesmo ritmo daqueles que ingressam. Isso gera um déficit de processos dentro da Casa de pouco menos de 40.000 processos sem julgamento, e 30.000 processos em tramitação. Em curto prazo, esse quadro tende a se alastrar demandando do Judiciário a criação de novas metodologias que possam acelerar a movimentação processual, diminuindo a morosidade que a Justiça Brasileira apresenta à população.

Diversas pesquisas têm sido realizadas na área, dessa forma, tentando buscar novas formas de acelerar e aperfeiçoar os trâmites processuais nos tribunais do País, incluindo o TJDFT. A Tecnologia da Informação tem sido recorrentemente utilizada, assim como iniciativas de automação têm apresentado bons resultados. A maior causa da morosidade da Justiça, contudo, é a possibilidade praticamente infinita de interposição de recursos nas decisões de instâncias inferiores. Muitos especialistas do Direito concordam que uma revisão do Código de Processo Civil tem condições de melhorar expressivamente esse quadro.

Considerando, entretanto, que essa é uma proposta em médio e longo prazo, alguma solução urge ser apresentada para minimizar o problema em curto prazo. A criação da Súmula Vinculante, procedimento por meio do qual tribunais de instância superior fecham o entendimento de determinadas matérias, garante que decisões de Primeiro Grau não sejam conflitantes ao entendimento anterior. Assim o índice de recorribilidade diminui com grande efetividade. Para o Segundo Grau, a jurisprudência tem forte papel nesse sentido.

Entende-se por jurisprudência o conjunto das decisões exaradas em Segunda Instância pelos Tribunais de Justiça. Esse conjunto de acórdãos, denominação dos documentos que armazenam essas decisões, tem forte influência sobre os órgãos julgadores, pois representam entendimento anterior sobre matérias já julgadas. A jurisprudência de uma côrte representa a forma como essa interpreta a lei, e o conhecimento de tal informação é estratégico para partes e advogados que acessam a Justiça, uma vez que é possível saber, a priori, o resultado de um pleito baseado na inclinação do TJ sobre pleitos semelhantes anteriores.

A pesquisa de jurisprudência é, destarte, o serviço de maior procura pelos jurisdicionados, e tem potencial para ser um fator de redução da procrastinação na Justiça. Várias pesquisas demonstram que a percepção de qualidade dos tribunais está diretamente ligada à qualidade e facilidade das pesquisas jurisprudenciais. O Tribunal de Justiça do Distrito Federal e Territórios, conhecendo tal fato, tem grande preocupação com a manutenção de sua base jurisprudencial.

A Secretaria de Doutrina e Jurisprudência é o setor responsável pela base de acórdãos do TJDFT. Essa equipe possui analistas de Direito que são responsáveis por indexar esses documentos permitindo melhores índices nas pesquisas textuais. Há um processo organizacional bem definido desde a concepção do acórdão dentro do gabinete do magistrado, passando pelo julgamento da ação, à revisão das notas taquigráficas e à assinatura digital dos documentos. Por fim, os acórdãos chegam à Jurisprudência onde são armazenados, indexados e disponibilizados para consulta.

Ocorre, todavia, que o crescimento da base de dados demanda um esforço muito grande da equipe. A quantidade de processos que chegam à Jurisprudência cresceu, e continua crescendo, em taxa claramente maior do que a chegada de novos recursos humanos para a Casa. Além disso, os analistas de Direito possuem vasto conhecimento jurídico, porém não possuem experiência com indexação, até porque essa não é sua formação. Percebem-se, portanto, algumas idiossincrasias nos resultados de pesquisa.

Normalmente os tribunais designam seus melhores analistas para a indexação jurisprudencial, dada à importância desse serviço prestado à população. O Tribunal de Justiça do Distrito Federal e Territórios não foge à regra. Essa equipe, outrossim, não possui formação nem conhecimento técnico de indexação, o que dificulta o bom resultado do trabalho. A equipe de informática, a qual também não tem treinamento para essa tarefa, acaba por não oferecer o suporte de TI adequado para o objetivo. Por fim, uma arquitetura de informação não é adequadamente planejada e construída de forma que a pesquisa de jurisprudência e, por conseguinte, os jurisdicionados, acabam por não obter as melhores ou mais relevantes informações que existem na base.

Há, por conseguinte, que se buscar soluções que possam desafogar a equipe de indexação e melhorar a qualidade dos índices construídos, em curto prazo. Em longo prazo, uma solução que não fosse dependente do crescimento da demanda traria benefícios incontáveis, no sentido de que a prestação jurisdicional é uma curva sempre ascendente. A indexação automática surge, dessa forma, como uma possibilidade com perspectivas promissoras para solução dessas questões

analisadas anteriormente, quais sejam a falta de experiência dos analistas de Direito com indexação, o grande crescimento da base de dados e a importância que a base de jurisprudência tem para o TJDFT.

Essa pesquisa tem por objetivo inserir-se nessa área. A indexação automática de acórdãos judiciais tem potencialidade para atender demandas da Justiça Brasileira, entre elas a diminuição da procrastinação da tramitação processual, a melhoria dos resultados de pesquisas jurisprudenciais e o aumento da capacidade dos tribunais frente à crescente procura da população por atendimento judicial. Além disso, a proposta de utilização de processamento de linguagem natural para indexação automática vem de encontro às abordagens estritamente estatísticas com potencialidade para obtenção de bons resultados. Essas últimas abordagens têm historicamente demonstrado melhor sucesso na área, até porque são pesquisadas há mais tempo. Acredita-se, no entanto, que a inserção de análise lingüística, em ambientes controlados, permite inteligibilidade, possibilitando que não apenas o reconhecimento de padrões ou a freqüência estatística sejam suporte para as decisões computacionais de indexação.

Postas tais considerações, a pergunta que essa pesquisa procura responder remete à efetividade de um mecanismo de indexação automática. A mesma apresenta-se por: Um sistema de indexação automática de acórdãos, utilizando processamento de linguagem natural, pode oferecer a um motor de busca uma base indexada que permita o aumento da revocação e da precisão em pesquisas textuais ?

2. OBJETIVOS

Nesse capítulo pretendem-se explicitar os objetivos geral e específicos dessa pesquisa.

2.1. OBJETIVO GERAL

Propor um modelo de indexação automática de acórdãos, baseado em processamento de linguagem natural, avaliando seu desempenho quanto aos índices de precisão e revocação de pesquisa textuais em relação a esses mesmos índices na indexação manual.

2.2. OBJETIVOS ESPECÍFICOS

Construir um corpus de língua portuguesa, baseado no vocabulário jurídico. O sistema de processamento de linguagem natural utilizado para apoiar o processo de indexação automática demanda um corpus para fundamentar suas análises probabilísticas. Há diferentes corpus de português, o mais utilizado em pesquisas atualmente é o Ceten-Folha construído a partir de texto jornalístico da Folha de São Paulo. Esse corpus, todavia, não apresenta o vocabulário específico que compõe o jargão do Direito, o que pode prejudicar os resultados dessa pesquisa. Foi decidido, então, que a construção de um corpus jurídico traria benefícios incalculáveis à pesquisa.

Indexar uma base de dados jurisprudenciais automaticamente com base no tesauro de jurisprudência do Superior Tribunal de Justiça.

Comparar os índices gerados a partir da indexação manual com os obtidos na indexação automática.

Comparar os resultados de buscas textuais nas bases quanto à revocação.

Comparar os resultados de buscas textuais nas bases quanto à precisão.

3. JUSTIFICATIVAS

Essa pesquisa justifica-se na constatação de que processamento de linguagem natural, em língua portuguesa, é uma área de pesquisa efetivamente incipiente. A maior parte das publicações da área é em outros idiomas, e há poucos grupos de pesquisa no Brasil que estudam PLN. O NILC – Núcleo Interinstitucional de Lingüística Computacional, laboratório da Universidade Federal de São Carlos, é um centro de pesquisa na área de processamento de linguagem natural em português. Diversos projetos e várias publicações têm origem nesse núcleo e a produção de ferramentas é bastante profícua.

Uma outra iniciativa que cabe ser mencionada é a Linguateca. Essa biblioteca digital, que se encontra no endereço eletrônico http://www.linguateca.pt, é um centro de recursos para processamento computacional da língua portuguesa. Muitos projetos estão disponíveis para utilização, bem como fóruns de discussão para a comunidade de pesquisa da área. Dessa forma, distribui-se o conhecimento e fomenta-se a pesquisa de PLN para o português.

Existem algumas outras iniciativas isoladas para estudo de PLN em português, e avanços em métodos de Inteligência Artificial têm construído um ambiente favorável para evolução dos resultados. Essa pesquisa, portanto, é justificada pela lacuna que a área de processamento de linguagem natural em português no âmbito técnico-jurídico possui, procurando produzir estudos nessa área.

Iniciativas de treinamento de equipes de indexação têm apresentado bons resultados, porém um sistema de indexação automática que apresente ganhos efetivos é uma solução que abrange diferentes aspectos. O crescimento da base não é problema para um sistema computacional, bem como o custo de operação é indiscutivelmente menor do que a manutenção de uma equipe de recursos humanos. A existência de erros, bem como a inserção de carga de subjetividade do indexador são questões bem solucionadas pela indexação automática. Assim, justifica-se uma pesquisa como essa baseado no fato de que indexação automática,

de acordo com vários autores, como Rowley (1988) ou Robredo (2005), tem potencialidade para igualar ou exceder os resultados alcançados com a indexação manual, a um custo bem menor.

Ademais, uma outra justificativa para esse estudo consiste na discussão sobre os resultados alcançados por indexação automática puramente estatística em sobreposição aos mesmos com métodos lingüísticos. Os métodos estatísticos apresentam bons resultados, mas os críticos creditam isso ao fato de que essa pesquisa existe há muito tempo. A utilização de metodologia lingüística, outrossim, é uma pesquisa recente decerto promissora. Isso se dá porque acredita-se que a inserção de componentes semânticos para indexação computacional tem condições de resolver problemas que a análise estatística desconhece, e por isso ignora.

Concluindo, as contribuições dessa pesquisa relacionam-se à geração de resultados de processamento de linguagem natural em língua portuguesa no domínio legal e à construção de um mecanismo de indexação automática de acórdãos.

4. REFERENCIAL TEÓRICO

Nesse capítulo pretende-se discutir e analisar o embasamento teórico para essa pesquisa. Os autores citados têm pesquisas na área de Direito e Indexação Automática os quais foram importantes para fundamentação das propostas.

4.1. TRAMITAÇÃO PROCESSUAL

O Poder Judiciário Brasileiro é regido e organizado por meio do Código de Processo Civil e do Código de Processo Penal. A tramitação processual é um *workflow* bem definido que apresenta o rito que qualquer ação deve seguir. Ações particulares, tais como de área cível, criminal, financeira, fiscal, dentre outras, possuem, evidentemente, diferenças intrínsecas em sua tramitação. De uma forma geral, contudo, o tronco principal da movimentação está definido no CPC – Código Processual Civil e é respeitado por todos os ramos do Direito.

A Justiça Brasileira possui três instâncias ou graus, quais sejam o primeiro, segundo e terceiro (SANTOS, 2002). A primeira instância é representada pelos fóruns e varas espalhados por todo o território nacional. Ela é composta por juízes de Direito, os quais julgam os processos de sua competência monocraticamente, ou seja, individualmente a partir de seus pressupostos experimentais e notório conhecimento jurídico.

Já o segundo grau é construído pelos tribunais de justiça dos Estados da Federação. Cada Estado, mais o Distrito Federal, possui um Tribunal de Justiça o qual é populado por desembargadores. Esses, os quais são eleitos dentre juízes, Procuradores ou Advogados membros da OAB, julgam os processos em colegiados, denominados órgãos julgadores, por meio de debate.

A terceira instância, por fim, é formada por Ministros, os quais são empossados a partir de desembargadores ou procuradores e compõem o STJ – Superior Tribunal de Justiça ou STF – Supremo Tribunal Federal. Schlichting (2004)

apresenta a descrição do CPC e permite a construção do seguinte diagrama o qual representa a tramitação processual nos tribunais do Brasil:

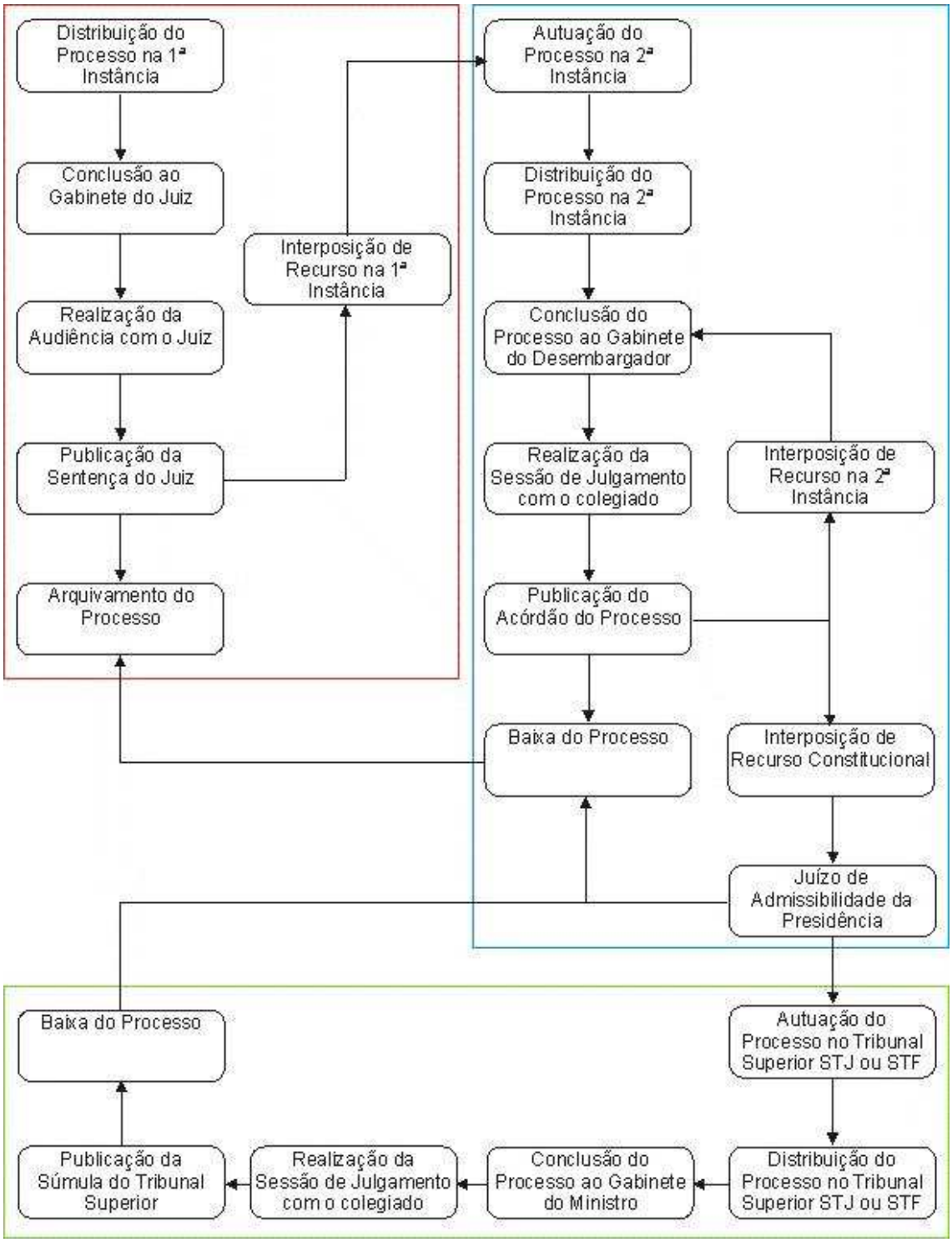


Figura 1 – *Workflow* da Tramitação Processual

A Figura 1 apresenta um limite vermelho para demonstrar a tramitação processual em primeira instância. Já o limite azul encapsula a tramitação do segundo grau. O limite verde, por conseguinte, representa a movimentação dos

processos em terceira instância, ou instância superior. Há uma linha geral entre as três possíveis áreas de andamento de processos, como apresenta Santos (2002), qual seja a autuação, distribuição, conclusão, decisão, publicação e baixa.

A autuação é o cadastramento do processo. Uma equipe de analistas de direito analisam a petição inicial dos advogados com o objetivo de classificar a causa para o tipo de Direito a qual a mesma se refere. Além disso, informações das partes, tais como outros processos antigos existentes ou em andamento são recuperadas e informadas. O processo é construído e fisicamente encapado com o objetivo de iniciar sua tramitação. O primeiro grau, em particular, não possui essa fase, a qual é aglutinada à segunda.

Já o segundo passo geral para movimentação processual é a distribuição. Nessa fase é sorteado aleatoriamente um magistrado o qual será eleito relator do processo. Isso significa que essa autoridade será responsável por analisar o processo e decidir sobre ele, em primeira instância, ou levá-lo a julgamento e expor seu voto nas instâncias superiores. A distribuição é das áreas mais críticas da justiça, pois a informação de qual magistrado será responsável pelo processo é estratégica para partes e advogados. O procedimento é, todavia, aleatório, não previsível e equânime, ou seja, procura manter as quantidades de processos entre os magistrados em níveis razoavelmente semelhantes.

O próximo passo, por conseguinte, é a conclusão. Essa representa simplesmente a recepção do processo pelo magistrado para o qual aquele foi distribuído. Isso significa que a autoridade, com o processo em mãos, fará sua análise com o objetivo de construir uma conclusão jurídica para ele. A importância dessa fase é demonstrada pelo fato de que a contagem de prazos para o processo inicia nela, e todas as estatísticas oficiais de produtividade da Justiça são originadas da conclusão processual.

O quarto passo, esse o foco deste trabalho, é a decisão do processo. Em primeira instância a decisão é um procedimento monocrático onde o Juiz decide o processo na audiência com as partes. Nos segundo e terceiro graus, entretanto, o colegiado de desembargadores ou ministros, respectivamente, decide o processo

por meio da discussão e do embate jurídico. O resultado do julgamento de primeiro grau é a Sentença. No segundo, o resultado é denominado acórdão. Na instância superior, por fim, esse compõe uma Súmula.

O quinto e penúltimo passo, qual seja a publicação, é a fase onde os prazos processuais para interposição de recursos iniciam sua contabilização. Com o pedido de recurso, a parte inconteste solicita novo julgamento de seu pleito na instância imediatamente superior a atual. A publicação se dá no órgão de imprensa oficial.

O último passo, por fim, denomina-se baixa. Os processos de instâncias superiores vão baixando às instâncias inferiores até atingirem o arquivamento. Tabelas de temporalidade são construídas para definição da quantidade de tempo de armazenamento dos Autos Processuais, e diversas iniciativas de digitalização e certificação digital de documentos estão em pesquisa e implantação para economia de espaço e outras vantagens que a TIC pode trazer para essa área.

Percebe-se, portanto, que a linha geral da tramitação processual na Justiça Brasileira é bastante simples e possui passos bem definidos e determinados. Como o foco dessa pesquisa concentra-se no julgamento processual, mais precisamente no resultado desse julgamento, a seção seguinte discute o documento gerado desse resultado, sua estrutura de formação e importância.

4.2. ACÓRDÃOS

Acórdãos são documentos que contêm o resultado de julgamentos em segunda instância da Justiça Brasileira. A macro-estrutura textual do acórdão é bem definida e rígida. O fato de que cada gabinete de desembargador ou juiz convocado produz um acórdão com formato próprio não permite que os documentos sejam estruturalmente diferentes. Apenas definições de layout, fonte, tamanhos entre outras superficialidades são diferenciadas. Não há legislação, contudo, para definir o formato ou conteúdo de um acórdão. Cada tribunal, exercendo a plenitude de sua independência funcional, na proposição de seu Regimento Interno, define a composição dos mesmos.

O TJDFT, no entanto, acaba de lançar um projeto de vanguarda para assinatura digital dos acórdãos logo ao final do julgamento das ações. O projeto Acórdão em Tempo Real tem em seus requisitos a padronização efetiva dos acórdãos gerados pelo Tribunal, de forma que o jurisdicionado terá sempre um documento completamente padrão e formatado para obter a confirmação da decisão de seus processos. A Figura 2 apresenta um exemplo de acórdão assinado digitalmente com suas partes evidentemente delimitadas.

Poder Judiciário da União
Tribunal de Justiça do Distrito Federal e dos Territórios

Órgão 4ª Turma Cível
Processo N. Agravo de Instrumento 20070020013293AGI
Agravante(s) GOLDEN CROSS ASSISTÊNCIA INTERNACIONAL DE SAÚDE LTDA
Agravado(s) FLÁVIA ROBERTA XAVIER DA SILVA
Relator Desembargador CRUZ MACEDO
Acórdão Nº 268.860

EMENTA

PROCESSUAL CIVIL. AGRAVO DE INSTRUMENTO. PLANO DE SAÚDE. NEGATIVA DE REALIZAÇÃO DE EXAME MÉDICO. ANTECIPAÇÃO DE TUTELA. REQUISITOS. DEFERIMENTO.
1. Presentes os pressupostos legais exigíveis (Art. 273, CPC), justifica-se o deferimento do pedido de antecipação de tutela para determinar à operadora de planos de saúde que promova a realização de exame médico prescrito à segurada (ultra-sonografia obstétrica), ante a possibilidade notória de graves danos à sua saúde ocasionados pela recusa, baseada em cláusula contratual aparentemente abusiva.
2. Recurso improvido.

ACÓRDÃO

Acordam os Senhores Desembargadores da 4ª Turma Cível do Tribunal de Justiça do Distrito Federal e Territórios, CRUZ MACEDO - Relator, ESTEVAM MAIA - Vogal, MARIA BEATRIZ PARRILHA - Vogal, sob a Presidência do Senhor Desembargador ESTEVAM MAIA em **NEGAR PROVIMENTO AO RECURSO, UNÂNIME,** de acordo com a ata do julgamento e notas taquigráficas.

Brasília (DF), 11 de abril de 2007

M319272
Brasília/DF
11 de abril de 2007 - 15:26:06
Desembargador CRUZ MACEDO
Relator

Código de Verificação: UYK2.2007.DT2P.GIU4.ZHQH.8X8B

AGRAVO DE INSTRUMENTO 2007 00 2 00 1329-3 AGI

RELATÓRIO

Cuida-se de agravo de instrumento interposto por GOLDEN CROSS ASSISTÊNCIA INTERNACIONAL DE SAÚDE LTDA em face da decisão (fls. 75/76) que deferiu à agravada pedido de antecipação de tutela para compelir a agravante a promover "os atos pertinentes à autorização de todos os exames que a autora necessite realizar por indicação médica", no prazo de 24 (vinte e quatro) horas, sob pena de multa diária por descumprimento.

Aduz a agravante, em síntese, que a decisão liminar extrapola os limites do contrato de prestação de serviços médico-hospitalares firmado entre as partes, em que expressamente exclui a realização do exame médico ansiado pela autora (ultra-sonografia obstétrica) durante o prazo inicial de carência. Diz ainda que a demandante não demonstrou que o procedimento enquadrava-se como de urgência ou de emergência, daí por que ausentes os requisitos autorizadores da tutela antecipada, e, por fim, argumenta que o Juízo *a quo* não observou a necessidade de prestação de caução idônea pela requerente.

Às fls. 142/143, indeferi o efeito suspensivo pleiteado.

Contra-razões às fls. 146/150, em que a parte agravada pugna pelo não conhecimento, pela falta de interesse de agir, e o improvimento do recurso, defendendo os termos da decisão agravada.

Preparo regular (fl. 139).

É o relatório.

VOTOS

O Senhor Desembargador CRUZ MACEDO - Relator

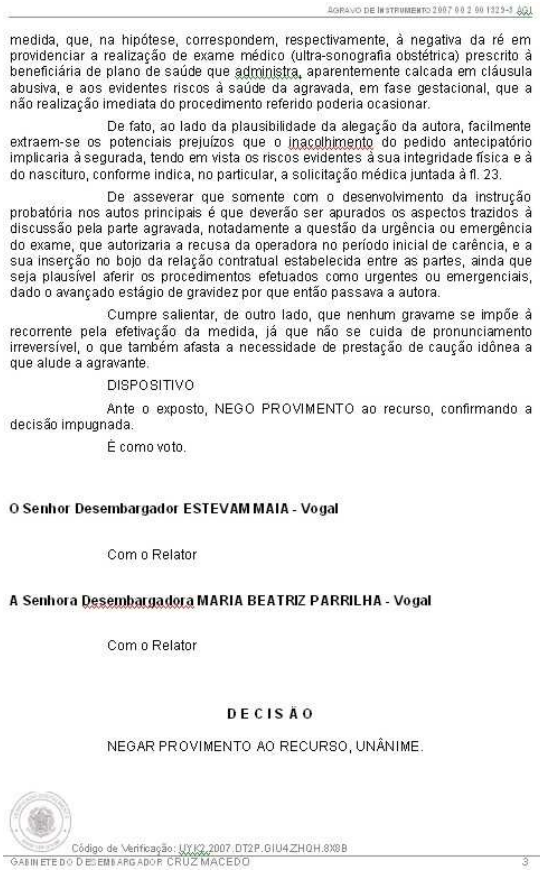
De início, afasto a alegação, produzida pela agravada em suas contra-razões, de que à recorrente faltaria interesse recursal, na medida em que o simples decurso do prazo de carência previsto contratualmente não elide o interesse posterior da parte agravante em ver-se ressarcida dos valores que despendeu para fazer face ao comando judicial antecipatório que lhe compeliu a promover a realização dos exames médicos prescritos para a postulante.

Conheço, pois, do recurso, porque presentes os pressupostos legais exigíveis.

A decisão agravada merece subsistir.

Em rigor, vislumbra-se a presença dos requisitos autorizadores da tutela antecipada, na esteira do disposto no Art. 273 do CPC, quais sejam, a verossimilhança das alegações da autora e o perigo da demora na efetivação da

Código de Verificação: UYK2.2007.DT2P.GIU4.ZHQH.8X8B
GABINETE DO DESEMBARGADOR CRUZ MACEDO 2



AGRAVO DE INSTRUMENTO 2007 00 2 00 1329-1 AGI

medida, que, na hipótese, correspondem, respectivamente, à negativa da ré em providenciar a realização de exame médico (ultra-sonografia obstétrica) prescrito à beneficiária de plano de saúde que administra, aparentemente calcada em cláusula abusiva, e aos evidentes riscos à saúde da agravada, em fase gestacional, que a não realização imediata do procedimento referido poderia ocasionar.

De fato, ao lado da plausibilidade da alegação da autora, facilmente extraem-se os potenciais prejuízos que o inacolhimento do pedido antecipatório implicaria à segurada, tendo em vista os riscos evidentes à sua integridade física e à do nascituro, conforme indica, no particular, a solicitação médica juntada à fl. 23.

De asseverar que somente com o desenvolvimento da instrução probatória nos autos principais é que deverão ser apurados os aspectos trazidos à discussão pela parte agravada, notadamente a questão da urgência ou emergência do exame, que autorizaria a recusa da operadora no período inicial de carência, e a sua inserção no bojo da relação contratual estabelecida entre as partes, ainda que seja plausível aferir os procedimentos efetuados como urgentes ou emergenciais, dado o avançado estágio de gravidez por que então passava a autora.

Cumpre salientar, de outro lado, que nenhum gravame se impõe à recorrente pela efetivação da medida, já que não se cuida de pronunciamento irreversível, o que também afasta a necessidade de prestação de caução idônea a que alude a agravante.

DISPOSITIVO

Ante o exposto, NEGO PROVIMENTO ao recurso, confirmando a decisão impugnada.

É como voto.

O Senhor Desembargador ESTEVAM MAIA - Vogal

Com o Relator

A Senhora Desembargadora MARIA BEATRIZ PARRILHA - Vogal

Com o Relator

D E C I S Ã O

NEGAR PROVIMENTO AO RECURSO, UNÂNIME.

Código de Verificação: UYIZ.2007.DT2P.GIU4.ZHQH.8X8B

GABINETE DO DESEMBARGADOR CRUZ MACEDO 3

Figura 2 – Exemplo de Acórdão

A Figura 2 demonstra a estrutura dos documentos, que se descreve por meio de suas partes. A primeira contém uma tabela com os dados do processo que originou a decisão do acórdão. Essa tabela possui informações como o colegiado julgador do processo, sua classe jurídica e os autores e réus do mesmo. Além disso, informação crucial dos recursos que foram julgados também encontram-se nessa área.

A próxima seção contém a ementa do acórdão. A ementa é um resumo do julgado, contendo o assunto do processo, palavras chave que o descrevam e os pressupostos utilizados para a decisão. Esse pequeno parágrafo é informação que é utilizada em citações jurisprudenciais na construção de petições ou votos.

Já a terceira parte descreve-se pelo acórdão propriamente dito. Esse é um parágrafo que contém o quorum do julgamento, descrito com seus respectivos papéis e a decisão final. A data do julgamento e a assinatura do relator do documento também fazem parte desse parágrafo.

A quarta seção, continuando, é denominada relatório. O relatório é onde o magistrado relator descreve os eventos e pressupostos do processo. São expostos o evento, a decisão do Primeiro Grau, o histórico da ação e eventuais interposições de autoridades externas, tais como o Ministério Público. O relatório é a forma por meio da qual os magistrados que não tiveram acesso ao processo tomam conhecimento da causa, no momento do julgamento. O mesmo é lido na sessão e o debate inicia-se após a conclusão de tal leitura.

Já a quinta parte é onde expõem-se os votos dos magistrados. O primeiro voto a ser descrito é o voto do relator do processo. Se a causa possuir revisor, desse é o próximo voto. Seguem os votos de cada um dos vogais do processo, ordenadamente até a final composição do quorum do julgamento. Em câmaras, esse número chega a oito. No Conselho Especial, doze. Para a Primeira Turma Cível, por exemplo, contam-se apenas quatro membros no julgamento de cada ação, sendo que um deles preside a sessão. Dessa forma, há no máximo dois vogais. Se houver debate, essa região permanece entrecortada por votos de vários magistrados, interpelando-se até o consenso atingir o seu resultado final.

A última seção, por fim, é uma breve repetição da decisão do acórdão, registrada ao final do documento com o objetivo de suceder-se à discussão do colegiado. Percebe-se, portanto, que a macro-estrutura do acórdão é fortemente bem definida e podem ser discernidas claramente quais áreas possuem a maior relevância para descrição do documento e do assunto do julgado. Esse tipo de informação é crucial para o sucesso da estratégia de indexação automática.

O próximo capítulo discute metodologias para avaliação da recuperação da informação. Isso se dá porque a indexação automática é uma área meio que tem por objetivo melhorar os resultados da área fim, qual seja a recuperação de informação. Sob esse aspecto, a pesquisa demanda o estudo de técnicas de medida de efetividade da recuperação para avaliação das ferramentas e propostas.

4.3. AVALIAÇÃO DE RECUPERAÇÃO DE INFORMAÇÃO

Sistemas de recuperação de informação possuem, de maneira geral, duas abordagens para serem avaliados. Uma primeira centra seu foco no usuário, onde são avaliadas as interfaces que os sistemas oferecem àqueles. Nesse tipo de medição, entrevistas são realizadas com o objetivo de averiguar a relação que os usuários desenvolvem com os sistemas, e a forma como conseguem extrair desses informação para atender a sua necessidade.

Uma segunda abordagem, essa mais quantitativa, procura avaliar sistemas de recuperação de informação com foco estrito na efetividade do sistema. Procuram-se, destarte, variáveis por meio das quais possam ser medidas a eficácia e eficiência das aplicações. A hipótese que sustenta tais medidas, na opinião de Aires (2005), é que quanto mais o sistema for efetivo em sua funcionalidade, ou seja, recuperar a maior quantidade possível de itens relevantes, com a menor quantidade possível de irrelevantes, melhor será atendida a necessidade do usuário.

Três conceitos fundamentais são utilizados para avaliação de sistemas de recuperação de informação. O primeiro deles é o conceito de relevância, e é provavelmente o mais difícil de ser avaliado. Um documento relevante é aquele que atende à necessidade do usuário, a qual foi submetida como uma pesquisa à base. O processo de traduzir uma necessidade em parâmetros de pesquisa é deveras subjetivo, contudo. Além disso, identificar quais documentos da base são relevantes para quais assuntos é uma classificação muito difícil de ser realizada, dificuldade essa que cresce exponencialmente com o tamanho do banco. O senso comum, entretanto, consegue reconhecer empiricamente um documento relevante e esse conceito é base para o cálculo dos outros.

Já o conceito de revocação refere-se à medida que avalia o quanto foi possível abranger os documentos relevantes da base de dados em uma pesquisa. Isso significa que, analisando-se o resultado de uma consulta, é possível calcular quantos documentos relevantes para o assunto utilizado como filtro foram efetivamente recuperados. Esse valor é analisado sobre todos os documentos que

há no banco de dados e tratam do mesmo assunto. A revocação como medida de avaliação é calculada por meio da seguinte fórmula:

$$\text{Revocação} = \frac{\text{Quantidade de Documentos Relevantes Recuperados}}{\text{Quantidade de Documentos Relevantes da Base}}$$

Essa definição procura demonstrar quão abrangente foi o resultado da pesquisa. O resultado do cálculo é entre zero e um, e quanto mais próximo de um for o valor, melhor é o índice de revocação. Percebe-se que uma pesquisa hipotética que retornasse todos os documentos da base teria o cálculo da revocação em um, uma vez que entre todos os documentos estão todos os relevantes, por definição. Esse índice representa, por fim, a cobertura do resultado.

O conceito de precisão, por fim, refere-se à medida que avalia o quanto o resultado de uma pesquisa foi limpo, ou seja, não sofreu a interferência de ruído. Documentos não relevantes para a pesquisa, mas que são retornados, poluem o resultado. A precisão é a medida que avalia o quanto o resultado da consulta foi efetivamente útil. A mesma é medida pelo cálculo abaixo:

$$\text{Precisão} = \frac{\text{Quantidade de Documentos Relevantes Recuperados}}{\text{Quantidade de Documentos Retornados pela Pesquisa}}$$

Essa avaliação procura demonstrar o quanto o resultado da pesquisa foi efetivo no aspecto de evitar que documentos irrelevantes poluam o retorno. Essa medida também varia no intervalo de zero a um, sendo um o indicador de melhor precisão possível. Uma pesquisa hipotética onde apenas um documento foi retornado, e esse único documento ser relevante, tem índice de precisão calculado por um. Seu índice de revocação, todavia, seria baixo no sentido de haverem outros documentos relevantes não recuperados. Na pesquisa hipotética anterior, onde todos os documentos foram retornados, o índice de revocação teria valor máximo, mas o de precisão teria valor bastante baixo, uma vez que a pesquisa retornou documentos relevantes e irrelevantes.

É notório, portanto, que os índices de precisão e revocação não melhoram necessariamente no mesmo sentido. O aumento da precisão de uma pesquisa pode diminuir a revocação, e o inverso também é verdadeiro. O ajuste de parâmetros e estratégias de pesquisa é uma sintonia fina para melhorar ambos os índices aumentando, consequentemente, a efetividade do sistema de busca.

Segue, na Figura 3, uma ilustração que resume os conceitos abordados anteriormente relativos à relevância, precisão e revocação. Uma base de documentos e os extratos utilizados para construção das medidas de avaliação para sistemas de recuperação de informação são delineados.

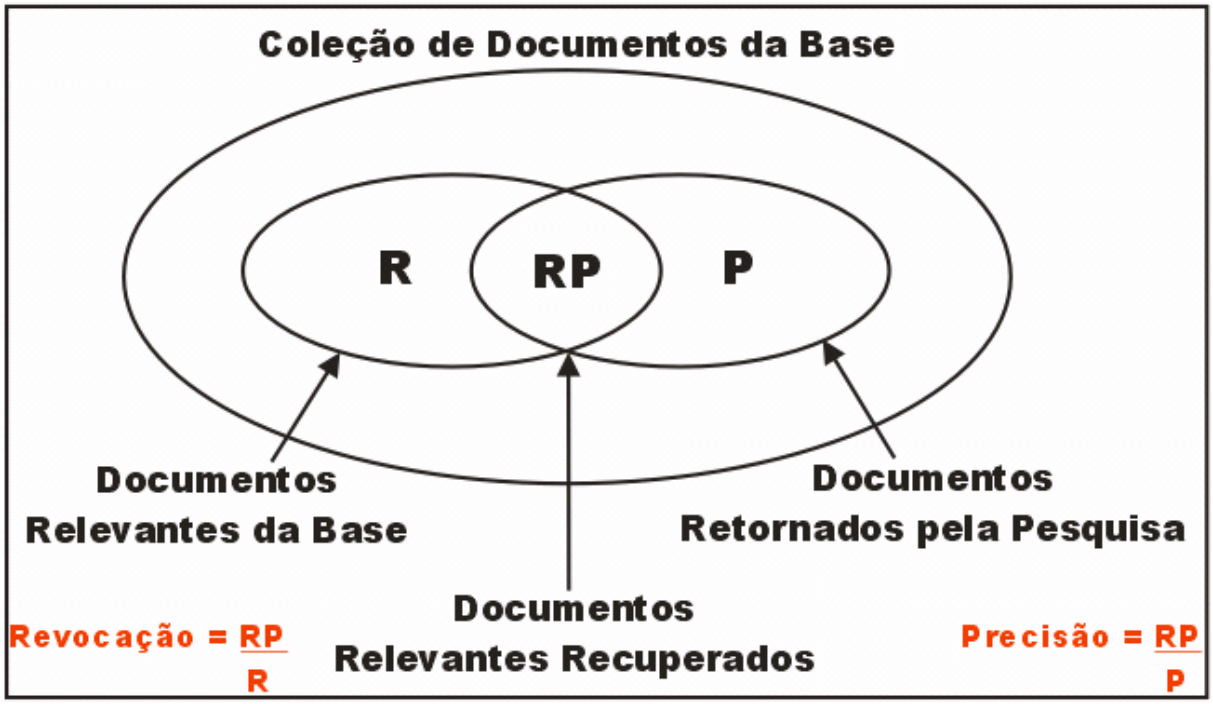


Figura 3 – Definições de Revocação e Precisão

O próximo tópico a ser discutido remete à indexação. Um pequeno embasamento teórico é colocado com o objetivo de fundamentar o efetivo objeto desse estudo, qual seja a indexação automática.

4.4. INDEXAÇÃO

Indexação é o processo por meio do qual definem-se termos que servem de índices para seleção de documentos (ROBREDO, 2005). Equivale-se à seleção de categorias de um esquema de representação de conhecimento, como descritores

de um tesauro. A indexação parte da premissa que a seleção do documento tem como ponto de partida o acesso à informação documentária.

Para Lancaster (2004), o objetivo central da feitura de indexação é a representação dos documentos existentes na base para fins de recuperação. Dois tipos de indexação concorrem para essa finalidade. O primeiro é a indexação por extração, qual seja, aquela indexação onde os termos utilizados para construção do índice são selecionados dentro do próprio texto. Já o segundo tipo é a indexação por atribuição. Essa, exaustivamente mais complexa de ser realizada por meios automáticos, baseia-se na definição de termos, os quais não necessariamente encontram-se no texto, para descrevê-lo e indexá-lo.

A definição de indexação como a construção de pontos de entrada para um mecanismo de recuperação de documentos é sugerida por Rowley (1988). A autora defende que a indexação deve considerar fortemente o público potencial da base de dados, uma vez que a seleção dos descritores deve ser compreensível para a maioria dos usuários. Sob esse aspecto, a indexação necessariamente deve passar por três passos: familiarização, análise e conversão de conceitos em termos para o índice. O objetivo da primeira fase é precisamente procurar inserir-se no universo do usuário potencial, com o intuito de reconhecer os documentos da base como ele o faz. A partir daí, a análise deve ser realizada para encontrar aquilo que é mais relevante em cada entrada do banco. A última fase, por fim, define a seleção dos descritores propriamente ditos e sua atribuição.

A finalidade última da indexação é precisamente a recuperação de informação que satisfaça as necessidades de potenciais usuários. Os usuários mais beneficiados pela indexação são aqueles que almejam dados sobre determinado assunto, mas não conhecem quais documentos da base de dados versam sobre ele. Os usuários que apenas desejam reconhecer documentos por meio de suas características próprias, tais como autor, título, data da publicação, edição, etc., não vêem na indexação ferramenta que traga grande diferencial (ROBREDO, 2005).

A Figura 4 contém a função da elaboração de índices e é apresentada abaixo com o objetivo de demonstrar a importância da indexação para a recuperação de informação útil e aplicável.

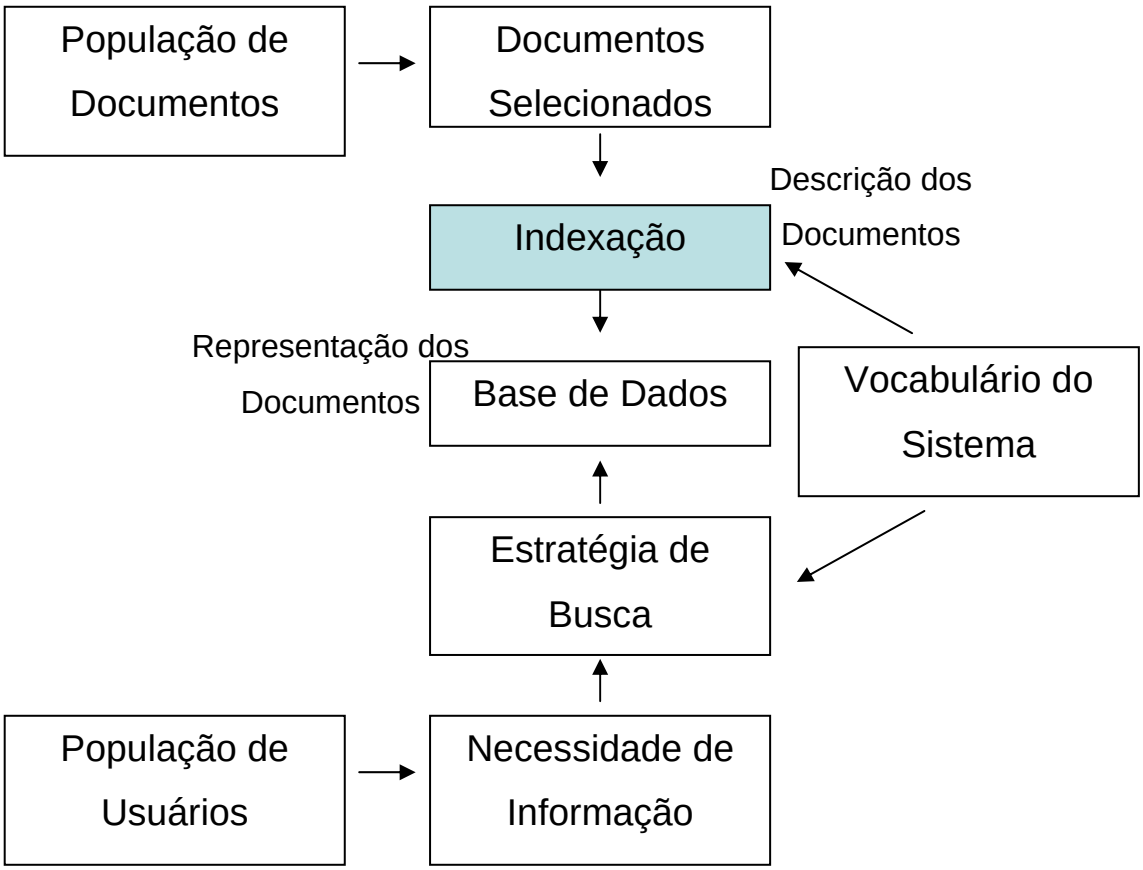


Figura 4 – Quadro Amplo da Recuperação da Informação
(LANCASTER, 2004, p. 2)

Percebe-se, portanto, que a indexação possibilita a criação de uma interface entre a população de usuários, com suas respectivas necessidades de informação, utilizando uma determinada estratégia de busca, e a base de representação dos documentos. O vocabulário próprio do sistema é o elo que une os sistemas de recuperação de informação e os bancos de dados de indexação.

Abaixo segue um fluxograma que detalha um processo de indexação que utiliza um tesauro para validação dos descritores.

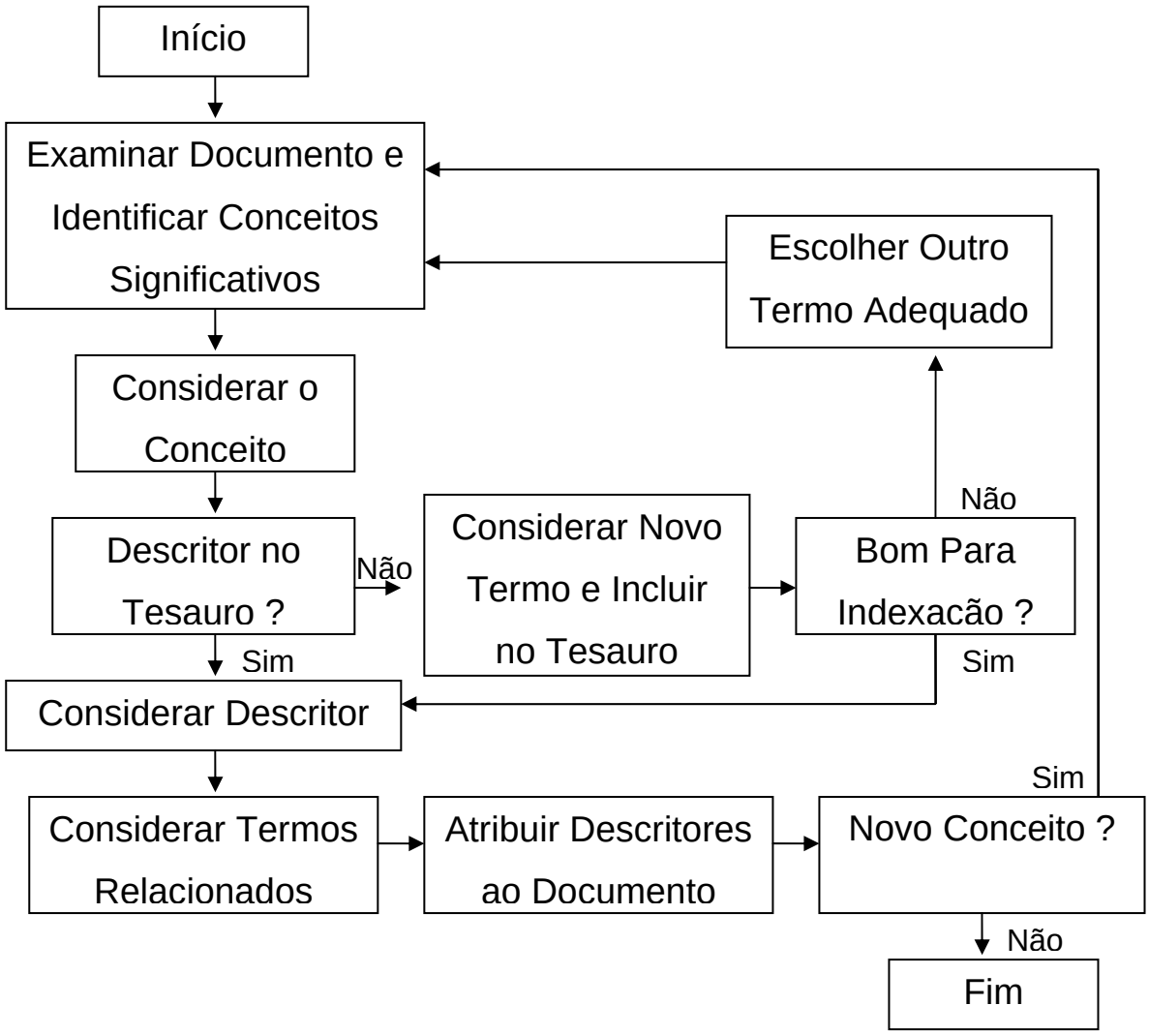


Figura 5 – Fluxograma Simplificado do Processo de Indexação Utilizando um Tesauro (ROBREDO, 2005, p. 167)

A Figura 5 demonstra que o apoio de um esquema de representação do conhecimento favorece grandemente o processo de indexação. Quanto mais relações forem representadas nesse esquema, melhor será o resultado da análise. Além disso, observa-se que o processo de indexação exige conhecimentos adequados do indexador, tanto de como realizar a tarefa tecnicamente, como da área que se está trabalhando. Ademais, esse é um processo que exige muito tempo, e, conseqüentemente, é bastante custoso. Todo o tempo gasto com indexação, todavia, é recuperado no momento em que se demandam pesquisas da base de dados. (ROWLEY, 1988)

A indexação manual possui grande carga da subjetividade do indexador, uma vez que cada pessoa constrói uma relação individualizada com o texto a ser indexado. Esse fator é demonstrado como influente nos índices de revocação e precisão de pesquisas, na opinião de Bastos (1984).

Na próxima seção discutem-se os conceitos, metodologias, propostas para indexação automática e métodos de avaliação de sistemas de indexação.

4.5. INDEXAÇÃO AUTOMÁTICA

A indexação automática de documentos, por outro lado, é aquela realizada diretamente por sistema de computador que analisa o texto, reconhece e constrói índices para recuperação do mesmo em pesquisas (BASTOS, 1984). A indexação automática pode partir estritamente de análises estatísticas das ocorrências das unidades léxicas ou processar a linguagem natural a fim de utilizar componentes morfológicos, sintáticos, semânticos ou pragmáticos para enriquecer a análise.

Para a construção do referencial teórico relativo à indexação automática, vários autores foram consultados, entre eles Pardo (2002), Braga Júnior (2001), Kuramoto (1999), Moens e de Busser (2002), Moens, Uyttendaele e Dumortier (1999), Abrahão (1997), Aires (2000). Esses autores, contudo, optaram por metodologias de indexação, sumarização ou recuperação diferentes desta pesquisa, mas foram importantes para fundamentação teórica.

O estudo de indexação automática remete a mais de 50 anos de pesquisa. Durante todo esse tempo, entretanto, grande parte dos estudos focou estritamente em texto escrito. Pesquisas em imagens, som, vídeo, e outras mídias são escassas, e mais recentes (ANDERSON; PÉREZ-CARBALLO, 2001). Para indexação automática de textos, dois modelos teóricos foram construídos durante esse período: o modelo espaço vetorial e o modelo probabilístico. A maioria dos resultados pragmáticos obtidos, no entanto, utiliza idéias dos dois modelos, de forma que os pressupostos de ambos entrelaçam-se mutuamente.

O processo de indexação automática tradicionalmente começa com unidades léxicas individuais de acordo com Lancaster (2004). A tokenização, ou seja, a separação do texto em palavras é definida pelo reconhecimento de texto entre marcas de pontuação. Tokenização é um neologismo do termo inglês *token* que representa, nesse sentido, uma cadeia de caracteres separada por determinadas marcas. Definir tais marcas é uma tarefa não trivial, porém, e decisões diferentes geram resultados idiossincráticos. Caracteres como hífen, parêntese, apóstrofo, ponto final, entre outros, demandam análises cuidadosas e individuais, pois seleções ambíguas podem advir da escolha dos mesmos como separadores, além, evidentemente, do caractere de espaço em branco. Um exemplo bastante simples disso é apresentado quando se utiliza o caracter hífen para separar uma oração explicativa, ou então quando esse separa uma unidade lexical composta. No primeiro caso, ele deve ser considerado como uma marca de tokenização. Já no segundo caso, deve ser ignorado, no sentido de fazer parte da palavra, e a unidade lexical deve ser selecionada por completo.

Sistemas de indexação automática muito comuns em editores de texto indexam documentos para pesquisa em toda e qualquer ocorrência de unidade lexical individual. Esse tipo de pesquisa tem alto índice de revocação, entretanto índice de precisão baixo. Além disso, palavras insignificantes como artigos ou conjunções são indexados com a mesma relevância de estruturas mais importantes.

A construção de dicionário de palavras não significativas surge como solução para essa situação. Anderson e Pérez Carballo (2001) defendem que uma lista que contenha unidades léxicas proibidas para indexação resolve muitos problemas, pois a diminuição do banco de dados dela decorrente facilita enormemente a consistência da aplicação e a velocidade das pesquisas. Alguns casos, no entanto, demandam análise cuidadosa, pois artigos e preposições, por exemplo, seriam bons candidatos a membros da lista negativa. Porém, em textos de biologia a expressão “Vitamina A” é relevante, e o “A” tokenizado seria ignorado, o que certamente geraria grande perda para o índice.

Historicamente percebeu-se que palavras isoladas não tinham capacidade suficiente para determinar o tema ou objetivo de um texto. Várias

metodologias para contagem de unidades léxicas surgiram, então, esperando que a freqüência das palavras pudesse explicitar o que é efetivamente importante no texto (MOENS, 2000). Tal idéia evoluiu pela percepção de que palavras com grande ocorrência em um documento não necessariamente oferecem vantagem para pesquisa desse material. Isso é explicado por meio de um simples exemplo, no qual uma base de dados jurídica possui na sua grande maioria, senão em sua totalidade, a unidade lexical “direito” em cada um de seus textos. Indexar tal termo, portanto, não traz diferencial estratégico algum para a pesquisa.

Dessa forma, a utilização de complexas estruturas matemáticas e cálculos estatísticos foram introduzidos para definição de pesos às palavras do texto. Assim, é possível criar um mecanismo mensurável para escolha dos descritores a partir das palavras que melhor se aderem ao objetivo de distinguir um documento. Esquemas de atribuição de pesos representam grande parte dos artigos publicados no TREC – *Text Retrieval Conference* – Conferência de recuperação de texto, disponível por meio do sítio de internet http://trec.nist.gov. Além disso, a análise da freqüência de uma palavra ou expressão em um texto, cruzada com a freqüência dessa mesma palavra na totalidade dos textos da base oferece uma curva que pode representar a efetiva utilidade dela para diferenciação do documento (LANCASTER, 2004). Essa informação também é utilizada em cálculos estatísticos para escolha de descritores.

Ainda procurando melhorar os resultados na avaliação dos pesos, a redução à raiz, ou lematização, é uma técnica bastante utilizada (ANDERSON; PÉREZ-CARBALLO, 2001). Diversas derivações de palavras ocorrem em qualquer língua, o que permite que várias unidades léxicas possam ser reduzidas ao seu radical, permitindo que apenas um deles seja eleito representante de uma classe. Isso diminui enormemente o banco de dados de indexação procurando melhorar os índices de revocação sem prejudicar em demasia os de precisão. Realizar essa tarefa automaticamente, por outro lado, é muito difícil. A análise morfológica que tem de ser realizada apresenta erros quando feita estritamente pelo computador. Um exemplo trivial apresenta-se na tentativa de excluir os plurais dos índices. Ao encontrar uma palavra terminada em “s”, destarte, basta remover o “s” para chegar ao radical singular. As unidades léxicas que têm plural diferente de apenas inserir o “s” ao final quebram essa regra, e as exceções, principalmente em língua

portuguesa, por vezes apresentam mais situações do que a própria regra. Krovetz (1993) acredita que a redução ao radical não oferece ganhos estratégicos para o cálculo de pesos para indexação automática, precisamente por causa das imensuráveis dificuldades para realizar esse processo computacionalmente.

A evolução natural desses métodos de indexação automática surge na utilização de frases além do reconhecimento de palavras apenas. Em alguns casos, palavras isoladas não representam efetivamente o significado de uma expressão, e isso gera idiossincrasias nos resultados de pesquisas (MOENS, 2000). O método mais simples de realizar tal tarefa é identificar adjacências com uma contagem da distância. Nessa oração, com uma distância de duas palavras, o resultado seria o seguinte:

nessa oração
oração com
com uma
uma distância
distância de
de duas
duas palavras
palavras o
o resultado
resultado seria
seria o
o seguinte

Quadro 2 – Indexação Automática por Meio de Identificação Contígua de Palavras da Última Oração do Parágrafo Anterior

Caso a distância selecionada fosse três palavras, então o mesmo período seria indexado de acordo com o seguinte Quadro 3:

nessa oração com
oração com uma

com uma distância
uma distância de
distância de duas
de duas palavras
duas palavras o
palavras o resultado
o resultado seria
resultado seria o
seria o seguinte

Quadro 3 – Indexação Automática por Meio de Identificação Contígua de Palavras do Mesmo Período com Distância de Três Unidades Léxicas

Percebe-se que, sucessivamente, é possível arranjar as palavras combinatoriamente de forma a produzir expressões. Essas deverão ser avaliadas em algum mecanismo de representação do conhecimento que permita garantir a semântica do resultado. Esse tipo de abordagem é bastante trivial, porém muito cara em medida computacional uma vez que, se os documentos forem grandes, e a distância tiver uma variação considerável, o espaço de armazenamento e processamento cresce exponencialmente.

Uma vertente mais sofisticada consiste na realização de *parser* do texto com o objetivo de reconhecer estruturas morfossintáticas (HLAVA, 2002). *Parser* é um termo que procura definir a etiquetagem do texto. O objetivo da ferramenta de *parser* é tokenizar o texto e etiquetar cada unidade lexical, ou conjunto delas, com a classificação desejada. Um *parser* morfológico faz etiquetação das palavras. Um *parser* sintático etiqueta orações ou sintagmas. Esse tipo de tarefa impede que combinações sem valor estrutural, tal como “duas palavras o” no Quadro 3, sejam recuperadas pelo sistema. Assim, diminui-se o espaço amostral de expressões candidatas a índices e, consequentemente, a complexidade algorítmica. Anderson e Pérez-Carballo (2001) defendem que a associação dessas metodologias para construção de expressões aos mecanismos de contagem de ocorrências, tanto dentro do texto como dentro do universo de documentos, oferece bons resultados para a seleção de índices. Dessa forma, a combinação de diferentes ferramentas e

abordagens pode e deve ser utilizada para aprimorar o processamento computacional.

Além dessas técnicas, a clusterização também apresenta-se como possibilidade para indexação automática. Entende-se por clusterização, um neologismo para o termo inglês *cluster*, o agrupamento de itens em classes de afinidade. Com isso, é possível agrupar textos, registros ou quaisquer entidades formando *clusters*, ou grupos, para objetivos de reconhecimento ou extração. Essa consiste em que um sistema automaticamente crie classes de documentos os quais possuam parâmetros de indexação com algum grau de congruência (MOENS, 2000). Classicamente a classificação implica em julgamento humano, porém com a definição parametrizável de alguns atributos reconhecidos por programas, é possível a construção de *clusters* de documentos. Um exemplo bastante trivial apresenta-se na utilização da informação de autor, termos de indexação, citação, referência ou até mesmo acoplamento bibliográfico para construção de grupos. Anderson e Pérez-Carballo (2001) levantam a tese que documentos similares que estejam num mesmo *cluster* tendem a ser relevantes para uma mesma pesquisa. Dessa forma, o custo computacional da construção dos grupos é pago pelo aumento da eficiência das pesquisas, pois a quantidade de acessos necessários para chegar a um documento diminui com o agrupamento dos mesmos via estruturas de dados.

A clusterização é insumo para diversas outras técnicas de indexação automática. Uma delas é a indexação semântica latente, a qual tem grande utilidade na procura por relacionamentos entre literatura dispersa (LANCASTER, 2004). A mesma consiste em que algoritmos computacionais procuram idéias subjacentes do texto por meio do vocabulário utilizado, mesmo que as palavras sejam diferentes. Isso significa que, por meio de análises de quantidades de co-ocorrências, um programa pode perceber que as unidades léxicas “furto”, “roubo”, “subtração”, “apropriação indébita” ou outras, por exemplo, tratem do mesmo assunto. Isso garante melhores índices de revocação em pesquisas, pois a ferramenta de busca pode fazer uso desse conhecimento (ANDERSON; PÉREZ-CARBALLO, 2001). Assim, um mesmo *cluster* será construído com os documentos que possuam essas palavras, e os mesmos serão recuperados para perguntas que utilizem qualquer uma delas.

Por fim, a avaliação de relevância é uma forma importante de se obter *feedback* do usuário final para a indexação automática (MOENS, 2000). O resultado do processo de indexação automática normalmente segue sem intervenção humana. A possibilidade de obter opinião de usuários finais, contudo, pode ajudar os sistemas de recuperação a melhorar seu desempenho em pesquisas posteriores. Atualmente as pesquisas procuram melhorar a indexação automática por meio de identificar padrões de comportamento humano em pesquisas e tomada de decisão, utilizando esses resultados para influenciar cruzamentos posteriores, na percepção de Anderson e Pérez-Carballo (2001). Assim, a avaliação de relevância tem um papel importante para que os sistemas possam melhor atender às necessidades de grupos de usuários.

Algumas medidas utilizadas para avaliação de sistemas de indexação automática são descritas por Hlava (2002). Uma delas refere-se à análise dos índices selecionados automaticamente. Esses se dividem em três classes: aqueles que coincidem com os escolhidos por um indexador humano; aqueles que o indexador teria escolhido e o processo automático ignorou; e por fim aqueles que o computador selecionou, mas um indexador humano preferiria ter ignorado. O estudo estatístico desses valores avalia a qualidade do sistema. Uma taxa de acerto de 85% é o requerido para um sistema minimamente aceito, na opinião da autora. Isso significa que a quantidade de índices que o processo automatizado ignorou, somado aos índices que foram erroneamente inseridos, deve ficar em no máximo 15% do total, considerando, evidentemente, que a indexação humana é balizada como 100% efetiva.

Bastos (1984) utilizou em sua pesquisa uma fórmula capaz de avaliar a aderência dos índices selecionados manualmente aos automaticamente. A declaração cria o conceito de coeficiente de avaliação que é calculado da seguinte forma:

$$q = \frac{c}{a + m - c}$$

Onde :

q : coeficiente de avaliação, valor comparativo entre duas indexações
c : quantidade de termos selecionados em comum pelos dois processos
a : quantidade de termos selecionados automaticamente
m: quantidade de termos selecionados manualmente

Esse coeficiente varia de valores entre 0 e 1, onde 0 é o pior caso, onde nenhum índice efetivamente coincidiu e 1 o melhor caso, onde todos os índices atribuídos automaticamente foram idênticos aos índices manuais. Multiplicando o resultado por 100 tem-se a porcentagem de aderência entre a indexação manual e a automática. Essa útil avaliação foi utilizada no contexto dessa pesquisa.

O último assunto tratado no referencial teórico é descrito na próxima seção. O Processamento de Linguagem Natural é uma área da Inteligência Artificial com grande utilidade para representação do conhecimento e raciocínio automatizado. Conquanto hajam abordagens aceitas em IA de que se pode ter comportamento inteligente sem conhecimento, Brachman e Levesque (2004) acreditam que o comportamento considerado inteligente parte sempre do mesmo princípio: conhecimento. Esse pode ser computacionalmente adquirido por meio de informações parametrizadas armazenadas. O PLN permite a parametrização de informação contida em textos de linguagem humana e a melhoria dos resultados desse tipo de pesquisa influencia diretamente os resultados de pesquisas de IA.

4.6. PROCESSAMENTO DE LINGUAGEM NATURAL

O PLN é uma área de pesquisa que procura construir mecanismos de tratamento e entendimento de documentos em linguagem natural. Essa linguagem, por possuir características informais de comunicação, tais como dependência de contextualização e morfologia e sintaxe não rigidamente definidas, entre outras, torna-a muito difícil de ser tratada computacionalmente. Diversas pesquisas na área têm surgido, principalmente para língua inglesa, porém poucas em língua portuguesa.

Os componentes utilizados em sistemas de processamento de linguagem natural são aqueles que executam tarefas de reconhecimento do texto de acordo com o nível de conhecimento lingüístico exigido ao tratamento. Cada um deles atua em um nível de profundidade e possui um grau de dificuldade de implementação crescente (MEDEIROS, 1999).

4.6.1. COMPONENTE MORFOLÓGICO

O componente morfológico é aquele que se preocupa com a forma como as unidades léxicas são apresentadas. Há a utilização de um dicionário, a fim de permitir a identificação de palavras válidas na linguagem utilizada. Uma maneira otimizável para isso é que o dicionário guarde o vocabulário em sua forma canônica, mais básica, e que o sistema seja capaz de derivar as formas mais complexas de cada entrada.

As classes morfológicas da língua portuguesa selecionadas para essa pesquisa encontram-se no Quadro 4.

ADJ – Adjetivo
ADV – Advérbio
ART – Artigo
CON – Conjunção
INT – Interjeição
NUM – Numeral
PON – Pontuação
PRE – Preposição
PRO – Pronome
SUB – Substantivo
V – Verbo

Quadro 4 – Classes Morfológicas da Língua Portuguesa

Observa-se que foi criada uma classe para pontuação o que, via de regra, não é uma classe morfológica. Isso é justificado pelo fato de que como o processo de tokenização é bastante complexo, optou-se por deixar essa classe para etiquetar

caracteres especiais que eventualmente passem pelo crivo do tokenizador. Dessa forma, é possível passar um segundo filtro ao resultado da primeira ferramenta melhorando os resultados finais.

A análise morfológica é a mais simples de ser realizada computacionalmente. Aplicações estritamente probabilísticas têm condições de realizar a tarefa com grande grau de acerto, sem demandar estruturas semânticas ou reconhecibilidade de contexto, utilizando autômatos finitos. Este componente é o mais utilizado no contexto desse estudo.

4.6.2. COMPONENTE SINTÁTICO

O componente sintático é aquele que se responsabiliza pela organização do conjunto das palavras, ou seja, as orações. A sintaxe é o conjunto das regras por meio das quais é possível reconhecer a estrutura da frase e as funções de cada um de seus constituintes. Esse procedimento é realizado por meio de um *parser* e o resultado armazenado em uma árvore.

As regras para a definição da sintaxe encontram-se em uma gramática formal a qual é utilizada pelo sistema. Essa gramática é utilizada para desconstruir a oração e montar uma árvore rotulada, onde cada estrutura será reconhecida pela sua funcionalidade e rotulada para derivação, via algoritmo automático.

Para esse trabalho, o componente sintático apresenta-se na construção dos sintagmas. As ferramentas desenvolvidas são capazes de recuperar formatos de sintagmas nominais ou verbais previamente estabelecidos, os quais são eleitos candidatos a descritores para os documentos.

4.6.3. COMPONENTE SEMÂNTICO

O componente semântico procura analisar as frases sintaticamente corretas para avaliar se as mesmas são compreensíveis. O objetivo é formalizar a interpretação do texto. Não é possível dar significado ao conteúdo, mas é possível analisar as relações válidas entre as palavras, a partir de seus conceitos.

Esse componente demanda um forte esquema de representação do conhecimento para ser efetivado. Uma ontologia que determine traços semânticos das palavras é demandada para reconhecimento da validade das relações. Tal análise não é realizada nesta pesquisa.

4.6.4. COMPONENTE PRAGMÁTICO

O componente pragmático, por fim, procura incluir o contexto a análise lingüística, a fim de permitir a geração de um significado. Esse utiliza uma base de dados construída em um esquema de representação de conhecimento para representar o contexto externo do texto e permitir a utilização desse conhecimento para inferências automatizadas. Esse difícil componente também não é utilizado para cumprimento dos objetivos desta pesquisa.

5. METODOLOGIA

Nesse capítulo é apresentada e discutida a metodologia utilizada para essa pesquisa. A descrição dos instrumentos utilizados bem como o detalhamento dos procedimentos são realizados com o objetivo de tornar claros os caminhos e decisões tomados durante a execução do projeto.

5.1. TIPO DE PESQUISA

Esta pesquisa tem um caráter fortemente quantitativo, como descrito por Gil (2002). Ela caracteriza-se pela contagem dos índices e avaliações dos valores de precisão e revocação por meio de técnicas de medida definidas na literatura. Além disso, segundo Marconi e Lakatos (2004) também se trata de uma pesquisa exploratória, dado que PLN em língua portuguesa na área jurídica é pouco estudado. O desenvolvimento das ferramentas e a aplicação para essa área reforçam a classificação exploratória.

5.2. AMOSTRA

Acórdãos de direito penal da base de jurisprudência do Tribunal de Justiça do Distrito Federal e Territórios do período compreendido entre 13/11/1997 e 22/03/2007, a qual possui documentos no padrão texto simples sem formatação. Tais documentos são filtrados da Primeira Turma Criminal, Segunda Turma Criminal, Câmara Criminal, Conselho Especial e Conselho da Magistratura do TJDFT. Para a Primeira Turma Criminal há uma base de 1.037 documentos. Na Segunda Turma, 1.211 acórdãos. Para a Câmara Criminal e os Conselhos Especial e da Magistratura somam-se 1.092 documentos. No somatório total de todos os colegiados selecionados, contabilizam-se 3.340 acórdãos de Direito Penal. Esses colegiados são compostos por quatro desembargadores em cada turma, oito desembargadores na câmara, doze magistrados no Conselho Especial e três autoridades no Conselho da Magistratura. Com um rodízio médio de sete juízes convocados em cada turma, somando quatorze juízes na câmara, representa-se uma amostra da formação de qualquer outro órgão julgador do Tribunal com um universo de trinta a trinta e quatro desembargadores de segundo grau e juízes de primeiro grau de jurisdição.

5.3. INSTRUMENTO

Os instrumentos utilizados são um conjunto de ferramentas desenvolvidas em linguagem de programação Java que se integram com o objetivo de indexar automaticamente documentos de acórdãos. Uma delas é um analisador denominado Qtag, disponível em http://www.english.bham.ac.uk/staff/omason/software/qtag.html, que tem por objetivo realizar o processamento de linguagem natural dos textos. Essa ferramenta encontra-se no catálogo da Linguateca e é desenvolvida pela Universidade de Birmingham. A mesma é um etiquetador probabilístico morfológico construído genericamente para qualquer língua. A ferramenta é um software livre.

O Qtag, por possuir uma característica de independência da linguagem, demanda um corpus da língua que será analisada, com o objetivo de realizar o treinamento. Esse corpus representa o dicionário que o Qtag vai utilizar para realizar suas inferências. Assim, um conjunto de unidades léxicas, com suas respectivas classes morfológicas deve ser oferecido à ferramenta. Não é realizada lematização. Cada unidade lexical é armazenada no dicionário com sua respectiva classificação morfológica, nas mais diversas derivações da língua. O Quadro 5 mostra uma pequena amostra do corpus construído a partir dos acórdãos selecionados para a pesquisa.

Unidade Lexical	Classificação Morfológica
Por	PRE
meio	SUB
Do	PRE
presente	ADJ
habeas	SUB
corpus	SUB
o	ART
impetrante	SUB
tedson	SUB
queiroz	SUB
narra	VER

que	PRO
ricardo	SUB
de	PRE
melo	SUB
foi	VER
denunciado	VER
juntamente	ADV
com	PRE

Quadro 5 – Corpus de Língua Portuguesa Construído a Partir de Acórdãos de Direito Penal do TJDFT

O analisador morfológico funciona de acordo com um mecanismo deveras simples. Ao ler um texto, o analisador tokeniza cada palavra e, caso a mesma seja encontrada no dicionário, é classificada pela entrada morfológica. Duas situações problemas surgem. A primeira relativa ao evento de uma palavra não ser encontrada no dicionário e a segunda quando a unidade lexical possui mais de uma classe gramatical.

Para solucionar essas situações a ferramenta invoca seu módulo probabilístico. Como a construção do corpus é realizada a partir de sentenças bem construídas na língua, o analisador tem condições de reconhecer estruturas mais complexas do que as estritamente morfológicas. Se a unidade lexical “furto” vier precedida de um pronome e sucedida por um substantivo, por exemplo, como em “eu furto veículos”, a ferramenta avalia a probabilidade de se tratar de um verbo, e a classifica como tal. Se a mesma unidade estiver entre um artigo e um adjetivo, como em “o furto planejado”, entretanto, então será classificada como substantivo. Com esse tipo de análise, as ambigüidades são resolvidas e unidades léxicas não conhecidas podem ser classificadas probabilisticamente.

Para gerar esse corpus, foi desenvolvida uma ferramenta de construção de corpus a qual recebe os acórdãos, analisa sua macro-estrutura textual e define sua área de atuação. Essa análise é realizada de acordo com as partes do acórdão descritas anteriormente. A primeira parte do acórdão, qual seja a tabela de identificação do processo, possui informações parametrizadas na base de dados, ou seja, não demanda análise lingüística. A segunda parte, que é a ementa, é um

resumo do conteúdo das outras partes, optando-se assim por ignorá-la para extrair a informação mais completa das outras fontes. A terceira, por fim, representa o texto da composição do órgão colegiado e da decisão, o que também é informação completamente parametrizada no banco de dados. Dessa forma, essas três primeiras partes são identificadas e ignoradas para autuação da construção do corpus.

As partes que são fundamentais para a análise lingüística são o relatório, voto do relator e voto do revisor. Essas partes contêm o texto do acórdão, o relato do magistrado sobre o evento, seu posicionamento jurídico, e a fundamentação do voto convergente ou divergente do magistrado revisor. Essa área possui o vocabulário jurídico que interessa para o corpus, e a informação não se encontra nos bancos de dados. Por esses motivos, essas áreas são as identificadas e isoladas para a construção do corpus. As duas últimas partes, quais sejam o voto do vogal e a decisão do julgado são informações parametrizadas no banco de dados que não possuem texto. Na grande maioria dos julgados, o voto do vogal é apenas concordando com o relator ou revisor, sendo que essas manifestações são padrão no sistema. A decisão é apenas uma linha enunciando o resultado, não trazendo ganho estratégico para o vocabulário. Postas essas considerações, essas duas partes também são desconsideradas para a análise. A saída dessa ferramenta é o insumo de entrada para o Qtag realizar suas inferências.

Um sistema de análise, ademais, foi desenvolvido para extração de estruturas a partir de texto analisado pelo Qtag. Após a etiquetação que a ferramenta realiza, um analisador tem condições de montar estruturas complexas em formatos previamente definidos. Essas estruturas são tais como Substantivo – Substantivo, Substantivo – Preposição – Substantivo, Substantivo – Adjetivo, Substantivo – Adjetivo – Preposição – Substantivo, Substantivo – Preposição – Substantivo – Adjetivo, selecionadas a partir da pesquisa de Medeiros (1999). Para fins de indexação automática, uma gramática complexa não seria necessária. Kuramoto (1999) defende a utilização de sintagmas nominais para a seleção de descritores em contraposição ao uso de palavras isoladas, porém essa abordagem exigiria a utilização de uma gramática mais complexa. Assim, essas composições são buscadas e listadas como candidatas a índices do documento. A ferramenta foi

desenvolvida, entretanto, para ser genérica, tendo condições de analisar qualquer tipo de forma.

Por fim, uma ferramenta de atribuição de índices baseada no tesauro jurídico realiza a parte final do processo. O apoio que o tesauro oferece garante parametrização e relevância para os índices extraídos. Isso se justifica pelo fato de que a análise e extração das estruturas evita conjunto de termos adjacentes sem carga semântica, tais como “a homologação dos” que seria do tipo Artigo – Substantivo – Preposição. Ainda assim, no entanto, recupera composições sem relevância para indexação, como “tribunais do País” do tipo Substantivo – Preposição – Substantivo. Um tesauro consegue filtrar essas recuperações com o objetivo de validar uma composição como relevante. O fato de a mesma estar presente no esquema de representação significa que ela tem potencialidade de ser descritor. Além disso, como dito anteriormente, as relações que o tesauro apresenta podem oferecer mecanismos para aprimorar os índices de revocação e precisão das pesquisas.

Essas ferramentas são consolidadas em um sistema denominado Sistema de Indexação Automática de Acórdãos, todo desenvolvido em linguagem de programação Java. A tela inicial dessa aplicação encontra-se na Figura 6 a seguir:

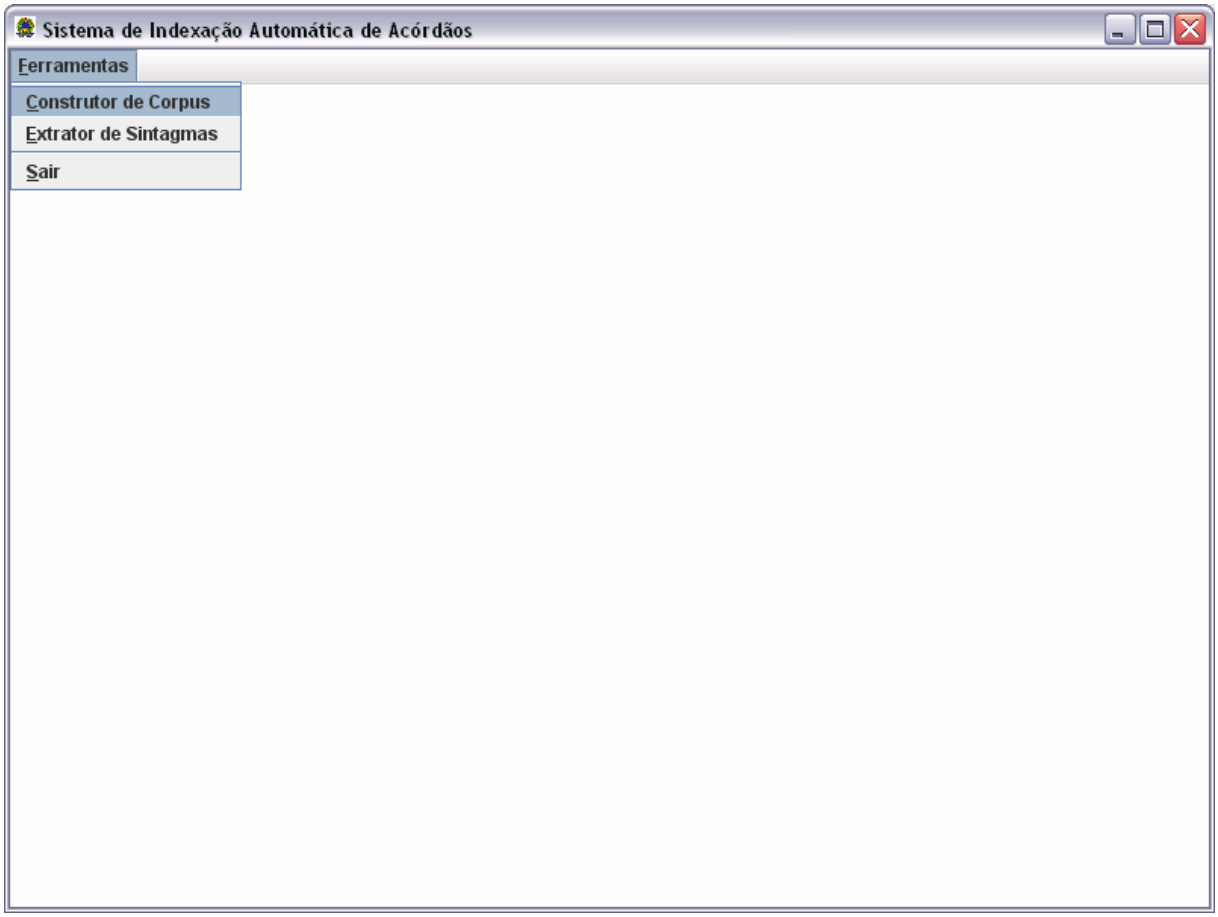


Figura 6 – Sistema de Indexação Automática de Acórdãos

5.4. PROCEDIMENTO

O primeiro procedimento adotado para realização dessa pesquisa foi a seleção dos acórdãos da base de dados jurisprudencial do Tribunal de Justiça do Distrito Federal e Territórios que foram analisados. Como mencionado, esses acórdãos são sobre Direito Penal. A base de jurisprudência, contudo, possui aproximadamente 270.000 acórdãos, numerados sequencialmente, os quais são divididos em três faixas. Os acórdãos de número 1 até o 70.000, representam informação de 02/06/1971 até 10/02/1994, não possuem informação do inteiro teor em formato digital. Isso significa que o texto do acórdão só existe em microfilme ou cópia física dentro do processo, no arquivo de segundo grau do Tribunal. Já os documentos de 70.001 até 102.676, cujo período compreende 13/04/1994 até 25/06/1997, possuem o inteiro teor digitalizado em imagens padrão JPEG. Esse tipo de formato não permite manipulação textual, como cópia de parte do documento. A última faixa, por fim, recorta a base a partir do acórdão 102.677, de 12/02/1998, até o último número atual, o que representa 260.082 de 30/08/2006. Essa terceira faixa

possui informação integral do inteiro teor do acórdão em formato DOC documento do *Microsoft Word*. A base de jurisprudência do Tribunal possui um crescimento de aproximadamente 100 acórdãos por dia, acompanhando a evolução da quantidade de processos entrados e julgados no TJDFT.

Essas faixas determinaram a seleção dos processos. Como o objetivo é a realização de indexação automática, não seria possível a realização da tarefa com sucesso somente a partir da informação do espelho do acórdão, tais como a ementa do julgado, ou a decisão do processo. Sendo imperativa a manipulação do texto integral do acórdão, foram extraídos apenas documentos da terceira faixa, relativos ao assunto desejado, Direito Penal, nos órgãos escolhidos. Como a jurisprudência evolui ao longo do tempo, entendimentos atuais acabam por sobrepor os mais antigos, confirmando a escolha da base mais recente como mais relevante para o trabalho. Os acórdãos são os de número 101.091 até 271.822.

O segundo procedimento consistiu na construção de um corpus de língua portuguesa baseado no jargão jurídico. Há alguns corpus de português existentes, o maior deles o denominado Ceten-Folha extraído de texto jornalístico da Folha de São Paulo. Esse corpus é base para algumas pesquisas de PLN em língua portuguesa, porém o mesmo não possui, por sua origem, o jargão jurídico tão tradicional na construção dos acórdãos. Dessa forma, percebeu-se a necessidade de construir um corpus próprio, a partir do inteiro teor dos acórdãos, que permita a aplicação de ferramentas de PLN com melhor acurácia. Para isso, foi totalmente desenvolvida uma ferramenta de construção de corpus, por meio da qual foi possível analisar todos os documentos extraídos e formatar uma saída que pudesse ser utilizada por ferramenta de PLN. Essa aplicação também foi desenvolvida em linguagem de programação Java e a Figura 7 abaixo apresenta a tela da ferramenta:

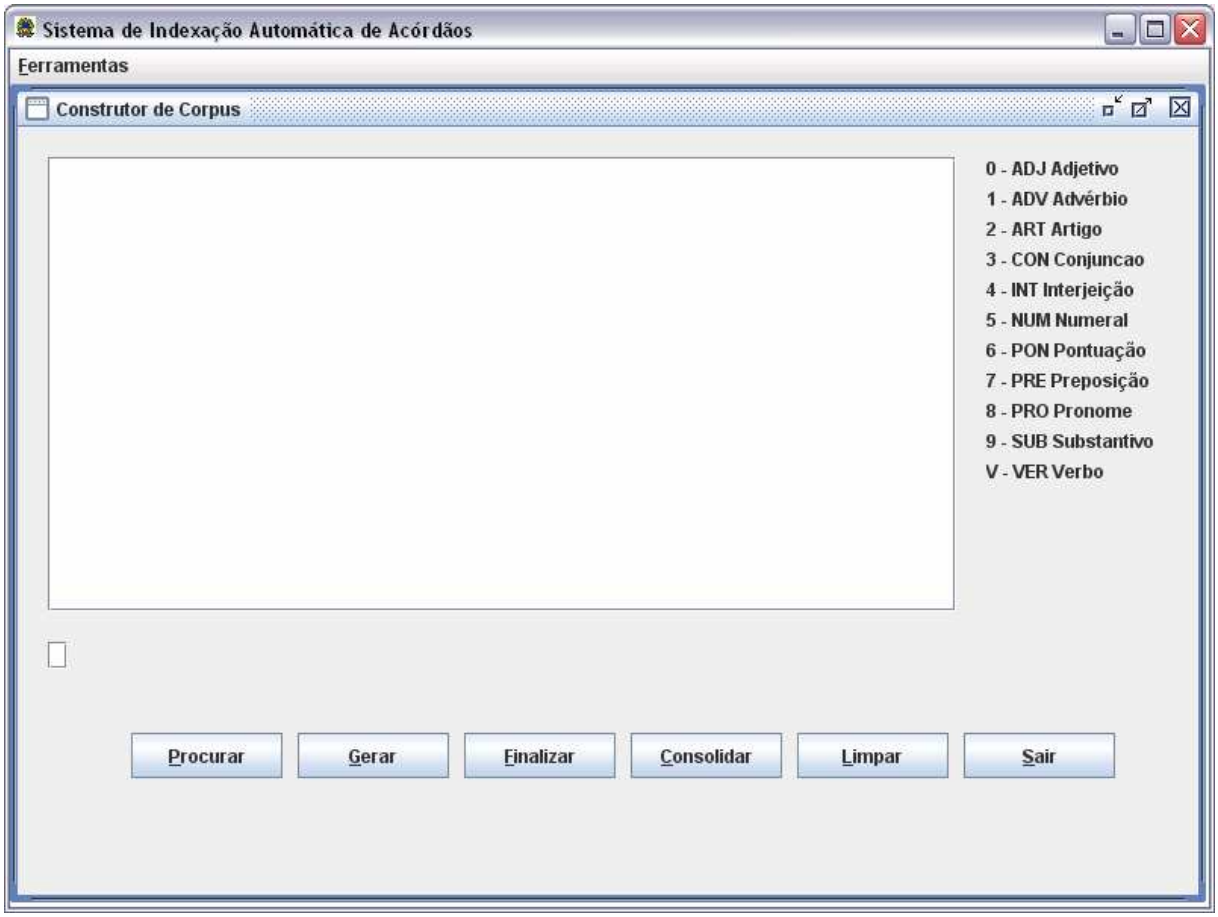


Figura 7 – *Print Screen* da Ferramenta de Construção de Corpus de Língua Portuguesa

Essa ferramenta possui diversas funcionalidades. A primeira delas é procurar acórdãos. Isso é feito abrindo-se uma janela por meio da qual é possível carregar um documento de acórdão que será analisado. Esse documento trata de um arquivo padrão .txt sem formatação o qual contém todo o texto do acórdão de um julgamento. Esse conteúdo será exibido na grande caixa que se encontra na janela para que o operador possa ler a totalidade do conteúdo do documento.

Uma outra funcionalidade é tal que se inicia o processo de classificação morfológica. O sistema vai recuperar cada uma das unidades léxicas do texto, selecionadas na área que interessa para a indexação automática, quais sejam o relatório e votos do relator e revisor, como discutido anteriormente, e o usuário vai classificá-los utilizando a tabela descrita à direita da janela. Essa tabela foi construída tendo por base as classes morfológicas discutidas no referencial teórico.

A próxima função representa a finalização do processe e é aquela em que a análise morfológica está completa e o resultado é armazenado em memória principal da estação de trabalho. Dessa forma, é possível construir a base do corpus incrementalmente, por meio da análise de cada um dos acórdãos selecionados para o treinamento da ferramenta de PLN.

A última funcionalidade, por fim, trata da consolidação do corpus. A análise morfológica realizada pelos passos anteriores grava seus resultados individualmente para os acórdãos da base. Isso garante melhor controle sobre a evolução dos trabalhos e também a possibilidade de correção ou substituição de documentos. Uma ferramenta que consolide esse insumo, portanto, faz-se necessária. Além disso, essa aplicação também já converte o formato do corpus para um esquema proprietário que é utilizado pelo processador de linguagem natural como banco de dados de dicionário. O dicionário possui entrada para as unidades léxicas e suas respectivas classificações, assim como as probabilidades para inferências de ambigüidade e não ocorrência. Por esse motivo, o esquema de armazenamento é bastante complexo e inteligível apenas pelo analisador. Isso justifica a necessidade de um tradutor para esse formato. Esse banco de dados é o corpora de língua portuguesa utilizado pelo Qtag.

Uma janela, então, é aberta por meio da qual é possível carregar os documentos que contêm o resultado da análise morfológica realizada nos acórdãos. Esses documentos também são arquivos padrão .txt sem formatação, os quais contêm todas as palavras de cada um dos acórdãos e sua respectiva classe morfológica. Ao selecionar o conjunto de documentos, a aplicação consolida a informação e converte o formato para o dicionário do Qtaq. Também há possibilidade de limpar os conteúdos da janela, reiniciando o estado da aplicação, e também fechar a ferramenta.

O terceiro procedimento foi a análise textual dos acórdãos com o objetivo de etiquetar (*parser*) os componentes do texto. O Qtag é usado para esse fim. Esse etiquetador, associado a uma ferramenta desenvolvida com o objetivo de servir de interface para suas complexas estruturas, foi utilizado com o intuito de analisar o texto e permitir a extração de diversas estruturas e sintagmas que se candidatam a

índices para o documento. Essas estruturas podem ser mais simples, como Substantivo ou Substantivo – Adjetivo, ou mais complexas como Substantivo – Adjetivo – Substantivo, ou Substantivo – Preposição – Substantivo – Adjetivo, entre outras. A ferramenta genérica permite extração de qualquer tipo de estrutura e apóia a montagem de índices a partir das mesmas. A Figura 8 abaixo demonstra a tela da ferramenta:

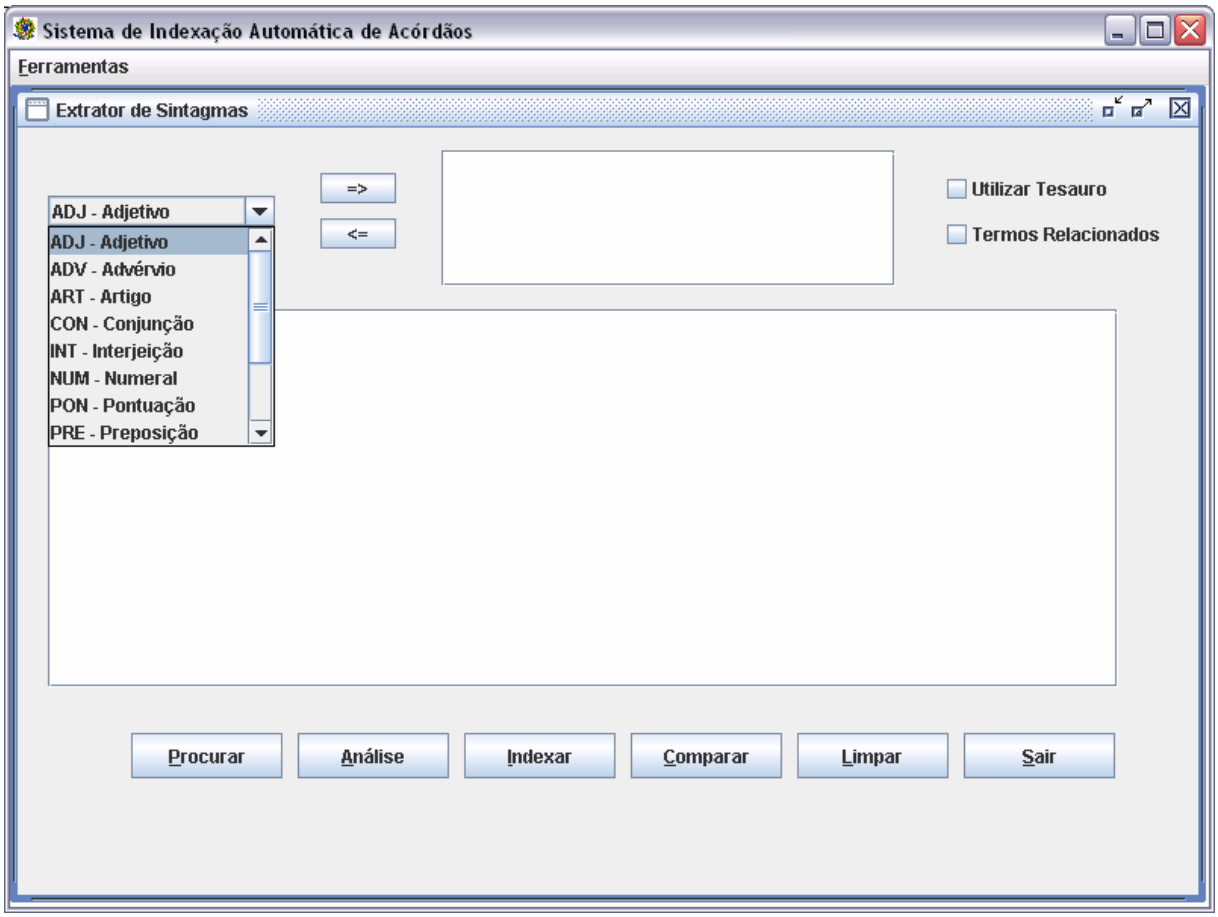


Figura 8 – *Print Screen* da Ferramenta de Extração de Sintagmas de Acórdãos

Há nessa ferramenta duas áreas, uma para seleção dos parâmetros e outra para execução dos procedimentos. A primeira área contém uma caixa de seleção com todas as classes morfológicas selecionadas para essa pesquisa. Além disso, dois botões com o objetivo de selecionar ou remover da seleção são oferecidos. Deve-se escolher a classe gramatical e acrescentá-la à seleção à direita. Assim, é possível montar uma seqüência de classes que serão procuradas no texto. Uma opção de separação da seqüência (linha tracejada) também existe a fim de permitir que vários padrões sejam buscados em uma mesma varredura do texto.

Isso aumenta significativamente a performance do programa. Ainda na área de parâmetros, duas caixas de opção definem como o sistema vai realizar a busca. A primeira – Utilizar Tesauro – indica que qualquer padrão reconhecido deverá existir no tesauro, senão será desconsiderado. A segunda – Termos Relacionados – informa que, caso um termo seja selecionado, todos os termos relacionados ao mesmo serão exibidos no resultado junto com o original.

Após essa configuração, uma janela é aberta para seleção do acórdão. O documento deve ser um arquivo texto formato .TXT sem formatação, o qual será lido, analisado, etiquetado (*parser*) e procurado pelos padrões definidos pelo sistema. O resultado é exibido na tela podendo ser trabalhado por meio de mudanças das configurações.

Já o quarto procedimento descreve-se pela utilização de um vocabulário controlado baseado em tesauro para atribuição dos índices. O esquema de representação utilizado é o tesauro do Superior Tribunal de Justiça. O tesauro formatado pelo Congresso Nacional era distribuído a todos os tribunais de justiça do País com o objetivo de uniformizar a indexação de jurisprudência de Segunda Instância da Justiça Brasileira. Até o início do ano de 2007 esse era o tesauro utilizado no TJDFT. O mesmo teve sua produção interrompida, contudo, e o STJ montou uma equipe responsável para atualizar e acompanhar a evolução desse esquema de representação. Na opinião dos analistas de jurisprudência dos tribunais de justiça consultados, quais sejam Distrito Federal, Goiás, Roraima e Paraná, o tesauro do STJ é o mais atualizado tesauro jurídico brasileiro, sendo adotado ou em adoção por essas Casas. Isso justifica a escolha do mesmo para essa pesquisa.

Acredita-se, entretanto, que uma ontologia ofereceria um resultado bastante mais favorável para a seleção de índices do que um tesauro, uma vez que a riqueza de relações que a ontologia possui, acrescido do fato de que sua construção seria baseada nos próprios acórdãos, teria a potencialidade de melhorar a qualidade do resultado. Essa construção, todavia, demandaria um tempo grande e a ajuda de especialistas em Direito, o que infelizmente, para o contexto desta pesquisa, não foi possível.

Esse vocabulário controlado é utilizado para o reconhecimento dos candidatos a índice selecionados no procedimento anterior. No momento da extração de estruturas sintagmáticas do texto, algumas estruturas encontradas têm efetiva relevância para descrição do documento, enquanto outras são irrelevantes. Unidades léxicas que são reconhecidas pelo tesauro, ou até remetidos a outras estruturas mais gerais, ou equivalentes, têm maior chance de serem descritores do documento, e são, portanto, bons candidatos a índices. Isso se dá porque a construção do tesauro concentra descritores relevantes para o Direito os quais, quando encontrados em documentos em linguagem natural, representam informação potencialmente importante.

As relações estabelecidas pelo tesauro, tais como as relações hierárquicas ou as de equivalência, são importantes para a qualidade da indexação. Com o objetivo de melhorar a revocação das pesquisas, por exemplo, pode-se utilizar os termos gerais, aumentando-se assim o espaço amostral. Se, por outro lado, o que se deseja é melhorar a precisão, então se reforça a utilização dos termos específicos. Percebe-se que a utilização das relações favorece a sintonização dos resultados de pesquisa.

As relações de equivalência têm forte influência nos resultados de pesquisa. Um exemplo disso é uma pesquisa na base de dados sobre acórdãos do crime “tráfico de drogas”. Na indexação humana, percebeu-se uma grande quantidade de acórdãos que, dentro os vários descritores selecionados para os documentos, um deles é “tráfico de droga”, no singular. Outra quantidade apresenta “tráfico de drogas”, no plural. E uma menor parte, por fim, utiliza a expressão “tráfico de entorpecente” e “tráfico de entorpecentes” também descritas no singular e plural. Como o sistema de pesquisa textual é baseado estritamente na unidade lexical, os índices de revocação das pesquisas prejudicam-se grandemente.

Dentro os termos mencionados anteriormente, a expressão oficial, descrita por Mirabete (2001), é “Tráfico de Entorpecentes”. No tesauro do STJ, a mesma é encontrada e as construções “tráfico de entorpecente” e “tráfico de droga” são equivalentes àquela. Já a estrutura “tráfico de drogas” não há no tesauro, demandando do sistema de indexação automática um pequeno passo de

lematização para o singular para captura desse tipo de desvio. Assim, o processo de indexação automática converte automaticamente qualquer expressão encontrada para a equivalente, garantindo unificação de vocabulário e aumento da revocação das pesquisas.

A unificação da linguagem de indexação e a utilização dessa mesma estrutura na ferramenta de pesquisa possibilita melhores respostas. Lancaster (2004) concorda que a utilização de um vocabulário único no sistema, que seja utilizado tanto na indexação quanto na estratégia de busca, favorece os resultados das pesquisas. Além disso, esse tesauro é o mesmo utilizado pela indexação humana, o que torna o mecanismo automático mais fácil de ser comparado com a metodologia atual, no sentido que os índices gerados por ambos são, necessariamente, da mesma fonte.

A ferramenta, portanto, abre uma janela para seleção dos acórdãos os quais sofrerão os mesmos processos de leitura, análise, etiquetação (*parser*) e procura pelos padrões definidos pelo sistema, com suporte do tesauro. O resultado é gravado em arquivos textos com os índices para cada um dos acórdãos. Também é possível limpar os campos da tela, reinicializando todas as configurações e análise e fechar a aplicação.

A última tarefa, por fim, foi a comparação dos índices gerados por indexação manual com os índices automáticos. Além disso, os resultados de pesquisas realizadas nas duas bases de dados com o mesmo motor de busca permitem avaliar, com suporte nos conceitos de revocação e precisão, a qualidade da indexação. Considerando-se que a base de dados é relativamente pequena, e que é um ambiente controlado, é possível reconhecer, a priori, quais documentos são relevantes para pesquisas conhecidas. Dessa forma, foi possível calcular os índices de revocação e de precisão nas bases de dados indexadas manualmente e automaticamente comparando-os entre si. Essa comparação permitiu avaliar o quanto o modelo de indexação automática mostra-se efetivo e subsidiará a decisão estratégica quanto a sua provável utilização.

Os programas de indexação do TJDFT, bem como a pesquisa textual dos sistemas coorporativos da casa, Intranet e Internet foram alterados para criação de novo parâmetro de filtro. Esse filtro seleciona precisamente esse corte da base de dados de produção. Assim é possível remeter qualquer tipo de pesquisa para o banco de dados e o mesmo retorna os resultados para três tipos de bases: o banco de dados completo; o corte na base de dados para os acórdãos selecionados para a amostra; e a base de dados reindexada com os descritores selecionados automaticamente. Por esse mecanismo foi possível a comparação dos resultados.

Além disso, houve estratégia incremental de avaliação do modelo, sendo que foram utilizados 1.037 acórdãos na primeira avaliação, 2.248 acórdãos na segunda e 3.340 documentos na avaliação final. Foram sempre utilizados incrementos de aproximadamente 1.000 acórdãos em cada iteração, de forma que na segunda rodada, por exemplo, foram utilizados os mesmos 1.037 acórdãos da primeira iteração acrescidos de mais 1.211 documentos. Na terceira, sucessivamente, os mesmos 2.248 documentos da rodada anterior acrescendo-se mais 1.092 acórdãos.

Para a primeira iteração foram utilizados os acórdãos da Primeira Turma Criminal. Na segunda, a esses documentos foram somados os acórdãos da Segunda Turma Criminal. Na última iteração todos os documentos de Direito Penal da base de jurisprudência do TJDFT foram analisados.

Segue Figura 9 com um resumo das etapas e atividades desenvolvidas no procedimento dessa pesquisa. Esse quadro tem por objetivo facilitar o entendimento da metodologia adotada desenhando graficamente os passos desenvolvidos até a conclusão da mesma.

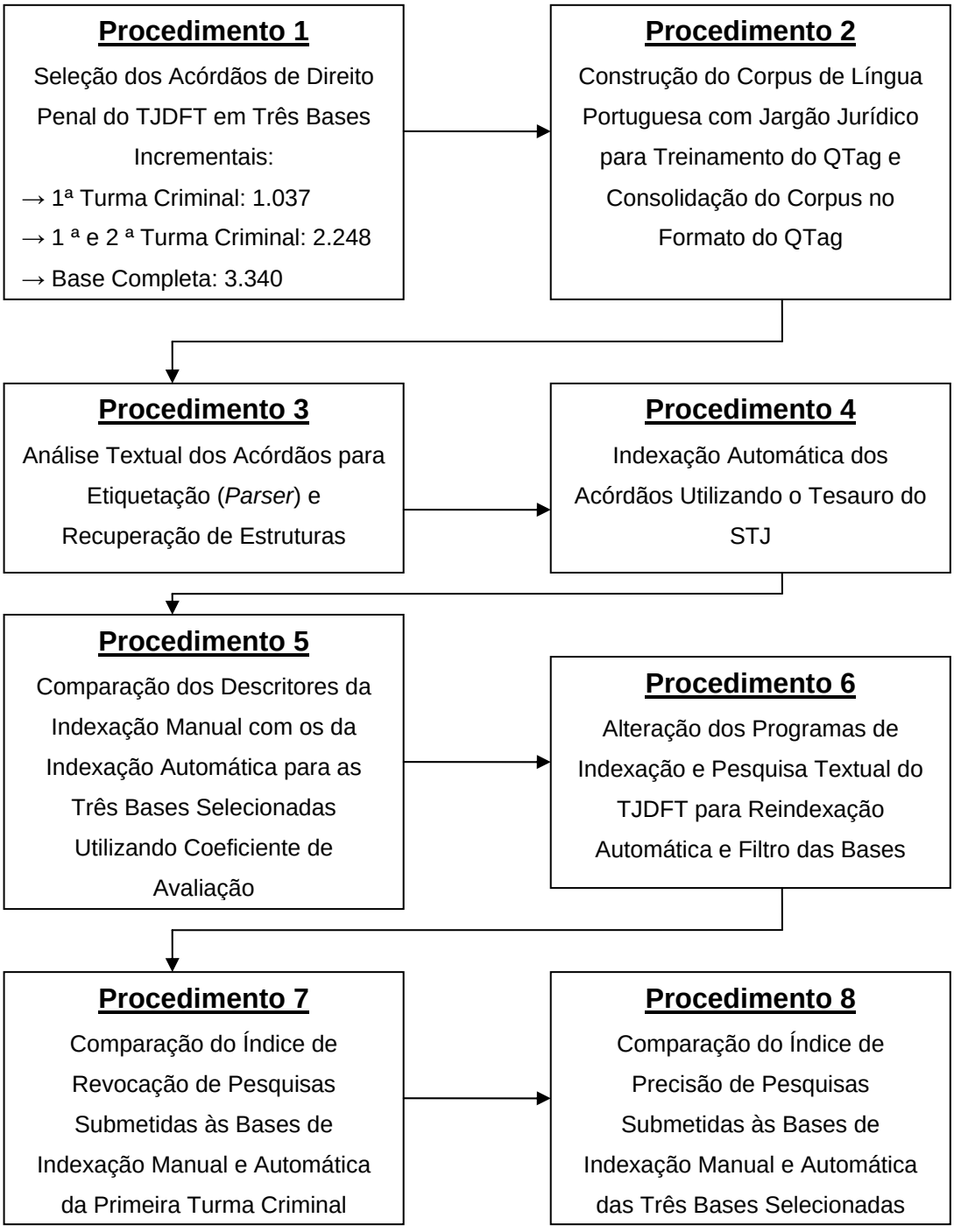


Figura 9 – Tipificação das Etapas / Atividades do Procedimento

6. ANÁLISE DOS RESULTADOS

Nessa seção são apresentados e discutidos os resultados dessa pesquisa. Como os objetivos específicos descreveram, três objetos de estudo foram analisados. Esses são a comparação dos índices atribuídos pela indexação automática com os da manual e a comparação dos cálculos de precisão e revocação em pesquisas textuais. Como a pesquisa utiliza arquivos texto sem formatação padrão .TXT para armazenar os acórdãos e os resultados, padrões de nomenclatura desses arquivos foram criados para organizar os conteúdos. Esses padrões são descritos no Quadro 6 a seguir:

Padrão de Nomenclatura	Definição e Conteúdo do Arquivo
NNNNNN.txt	Arquivos que contêm o conteúdo dos acórdãos. O nome do arquivo é o número do acórdão, um número de 6 dígitos que, no contexto dessa pesquisa, varia entre 101.091 até 271.822
NNNNNN_Corpus.txt	Arquivos que contêm o resultado da análise morfológica do acórdão. Esses arquivos possuem em cada uma de suas linhas cada unidade lexical extraída do acórdão com sua respectiva classe morfológica.
NNNNNN_Indexação_Manual.txt	Arquivos que contêm a indexação manual realizada pela equipe de jurisprudência do TJDFT. Os termos são separados pelo caractere vírgula ‘,’.
NNNNNN_Indexação_Automática.txt	Arquivos que contêm a indexação automática realizada pelo sistema. Os termos também são separados pelo caractere vírgula ‘,’.

Quadro 6 – Padronização da Nomenclatura de Arquivos

Percebe-se, destarte, que os arquivos de acórdãos são utilizados para construir o corpus e para indexação automática. Esses são extraídos da base de

jurisprudência do TJDFT. Já os arquivos de corpus são gerados pelo sistema e consolidados com fins de tradução para o dicionário do Qtag. Os arquivos de indexação manual, por sua vez, foram extraídos dos sistemas de informação do Tribunal. Os arquivos de indexação automática, por fim, são gravados pelo sistema de Indexação Automática e são utilizados para comparação de índices e reindexação da base de jurisprudência do TJ.

O direito penal, como todos os outros ramos do direito, é constituído por várias matérias as quais são classificadas por meio dos crimes. Assim, a recuperação dos acórdãos é realizada pelos delitos que foram julgados. Parâmetros de pesquisa com a descrição das ilegalidades, recuperadas do código de Mirabete (2001), permitem a recuperação dos documentos que apresentam decisões sobre as matérias desejadas. Por tratar-se de uma base de documentos controlada, foi possível classificar os documentos relevantes para as pesquisas de crimes contabilizando cada matéria tratada pelos acórdãos. Essa tarefa foi realizada apenas para os acórdãos da Primeira Turma Criminal, pois, por limitação de tempo, o cálculo da revocação só foi realizado para esse extrato da base. O Quadro 7 apresenta esse quantitativo. É importante observar que o somatório final é maior do que os 1.037 documentos da base da Primeira Turma Criminal, uma vez que um mesmo acórdão pode tratar de concurso de crimes, ou seja, ser relevante para mais de uma pesquisa.

Crime	Acórdãos
Aborto	2
Acidente de Trânsito	45
Atentado Violento ao Pudor	82
Constrangimento Ilegal	183
Corrupção de Menor	4
Crime Hediondo	160
Denunciação Caluniosa	3
Estelionato	30
Estupro	18
Extorsão	10

Furto de Veículo	11
Furto Qualificado	75
Habeas Corpus *	320
Homicídio Qualificado	117
Latrocínio	36
Lesão Corporal	55
Porte de Arma de Fogo	250
Receptação	22
Roubo à Residência	48
Roubo Qualificado	200
Seqüestro	5
Tortura	22
Tráfico de Entorpecente	127
Uso de Documento Falso	20
Violação de Direito Autoral	4

Quadro 7 – Quantidades de Acórdãos da Primeira Turma Criminal por Crime

Uma importante observação se refere à linha do Quadro 7 marcada por asterisco (*). Essa expressão não se refere a crime propriamente dito, mas é um evento muito relevante para o Direito Penal. O pedido de *Habeas Corpus* garante o relaxamento de prisão para o acusado, e é um recurso recorrentemente solicitado nos pleitos. É grande a ocorrência de acórdãos que descrevem casos onde as partes solicitam esse direito, de forma que essa informação demonstrou-se importante para a pesquisa.

Constrói-se, assim, um parâmetro bem definido de relevância para os documentos. Essa informação será utilizada no cálculo dos índices de revocação para avaliação da indexação automática dos acórdãos.

6.1. COMPARAÇÃO DOS ÍNDICES ATRIBUÍDOS

A indexação automática foi realizada de acordo com o procedimento descrito no capítulo anterior. Após a execução dos passos, a comparação dos índices foi realizada por meio dos conteúdos dos arquivos _Indexação_Manual.txt e

_Indexação_Automática.txt. Foi utilizada a fórmula para cálculo do coeficiente de avaliação descrito no referencial teórico, o qual oferece um parâmetro para análise dos índices.

Esse procedimento realiza-se utilizando um novo tokenizador baseado no caractere vírgula ‘,’ para separação dos termos. Duas tabelas são montadas, cada uma com os descritores de cada tipo de indexação, e seus registros são comparados para encontrar congruências entre si. O cálculo do índice é realizado para cada um dos 3.340 acórdãos, obtendo-se assim uma base de avaliação. O Quadro 8 mostra um pequeno extrato da base com os cálculos do coeficiente de avaliação.

Acórdão	Índices Manuais	Índices Automáticos	Congruências	Coeficiente de Avaliação
187623	1	12	0	0,0000
105765	10	15	0	0,0000
187636	1	12	0	0,0000
159859	8	19	1	3,8461
150959	8	15	1	4,5454
198277	12	19	2	6,8965
182272	11	18	2	7,4074
133818	14	22	3	9,0909
182609	10	16	3	13,0434
205688	20	28	6	14,2857
133216	6	9	2	15,3846
179641	8	19	4	17,3913
174292	13	18	5	19,2307
178373	27	20	8	20,5128
131396	5	16	4	23,5294
191160	8	11	4	26,6666
Médias	**7,6065**	**16,5380**	**0,7888**	**2,8958**

Quadro 8 – Cálculo dos Coeficientes de Avaliação para Primeira Turma Criminal

Várias situações há que demandam análise, e esse Quadro 8 resume algumas delas. A primeira refere-se aos acórdãos que apresentaram zero congruência entre os índices selecionados manualmente e automaticamente. Como esse evento ocorreu com um número muito grande de vezes, aproximadamente a metade da base, tornou-se importante investigação mais aprofundada das causas. Para isso torna-se necessário a discussão do procedimento de indexação realizado pelos analistas do TJDFT.

A Subsecretaria de Doutrina e Jurisprudência do Tribunal possui atualmente 56 Analistas de Direito para realização da indexação dos acórdãos. Esses analistas estudam cada um dos documentos e selecionam os descritores baseado em sua experiência, percepção e formação jurídica. Há um sistema de informação que armazena esses dados em uma base publicada na Internet para pesquisa textual.

Esse sistema, no entanto, foi desenvolvido em 1997 e apresenta algumas deficiências decorrentes da evolução dos trabalhos. Uma delas é representada pelo precedente – sucessivo. A equipe lançou um projeto de vanguarda que consiste na criação de *clusters* de acórdãos a partir da informação de acórdãos precedentes e seus sucessivos. Um acórdão precedente é aquele que primeiramente define uma decisão para determinada matéria jurídica. Todos os julgamentos posteriores que tomarem a mesma inclinação geram acórdãos sucessivos àquele precedente. O objetivo é classificar a jurisprudência, diminuindo, assim, o tamanho da base facilitando a recuperação da informação. Ocorre, contudo, que não há um campo específico para esse fim no sistema de informação. Decidiu-se, então, que o campo de indexação seria utilizado. O descritor selecionado para a indexação do documento de um acórdão sucessivo é o número do acórdão precedente. Esse número, evidentemente, não existe em tesauro e nunca é semelhante a um descritor selecionado automaticamente. Esses acórdãos sucessivos indexados dessa forma nunca são recuperados por meio de pesquisa textual, prejudicando fortemente o índice de revocação de pesquisas.

Outra inadequação do sistema de informação apresenta-se na opção que o indexador faz por utilizar a própria ementa como índice. Em algumas situações, o

analista de jurisprudência acredita que a ementa do acórdão contém um bom conjunto de descritores para o mesmo. Por causa disso, no campo de indexação do sistema é digitado o termo “Vide ementa”. Essa expressão é indexada e também não possui correspondente em tesauro, ocasionando baixa taxa de congruência e também reduzindo o índice de revocação das pesquisas. Os acórdãos cadastrados dessa maneira poderiam se excluídos numa fase de limpeza dos dados, melhorando, conseqüentemente, o resultado da pesquisa.

No Quadro 8, a primeira e terceira linha apresentam exemplos dessas situações descritas. A segunda linha contém outro exemplar onde foram selecionados índices manuais, dez em particular, mas nenhum deles foi igual aos selecionados automaticamente, no caso quinze. Isso ocorre por vários motivos, um deles o fato que a indexação humana não é restritiva em tesauro, ou seja, é feita com vocabulário livre. O tesauro é utilizado apenas para mostrar ao indexador que o termo selecionado existe no esquema de representação, conquanto não seja bloqueada a inserção de termos que não haja. Assim, a ocorrência de termos fora do tesauro na indexação manual é muito alta o que inexiste na indexação automática. Além disso, a seleção manual prima pela escolha de descritores com uma única unidade lexical. Na indexação automática optou-se por construir estruturas mais complexas, gerando diferenças na comparação.

Uma outra observação refere-se à quantidade de descritores selecionados para indexação. A média de todos os documentos selecionados para essa pesquisa é igual a 7,6065, ou seja, há normalmente de sete a oito descritores para indexar um documento. O processo de indexação automática dobrou esse resultado, chegando entre dezesseis e dezessete descritores em média. Percebe-se que a grande quantidade de acórdãos com apenas uma única unidade lexical selecionada para descritor forçou a diminuição do cálculo da média para baixo na indexação humana. Esse fato é o maior defeito que se apresentou na indexação manual.

Para tornar possível o estudo desses resultados, foram agrupados os índices de mesmo valor, e contabilizado suas ocorrências. O objetivo de realizar essa distribuição é analisar a quantidade de acórdãos com bons coeficientes

avaliando quantitativamente a qualidade da indexação. Observar cada um dos acórdãos individualmente permite uma análise qualitativa que procura explicar o motivo de baixos coeficientes de avaliação. O Quadro 9 a seguir demonstra essa distribuição para os acórdãos da Primeira Turma Criminal.

Coeficiente de Avaliação	Freqüência
0	573
1	5
2	32
3	65
4	74
5	72
6	39
7	30
8	34
9	23
10	26
11	9
12	16
13	11
14	6
15	7
16	8
20	4
22	2
25	1

Quadro 9 – Distribuição dos Coeficientes de Avaliação para a Primeira Turma Criminal

Demonstra-se, assim, a quantidade de acórdãos com coeficiente de avaliação zero, ou seja, não encontrada nenhuma congruência entre a indexação manual e automática. A maior freqüência apresenta-se nos coeficientes entre o intervalo três e cinco. Plotando-se esses resultados em um gráfico, é possível

construir uma curva demonstrativa da distribuição dos coeficientes, tal qual a Figura 10 apresenta.

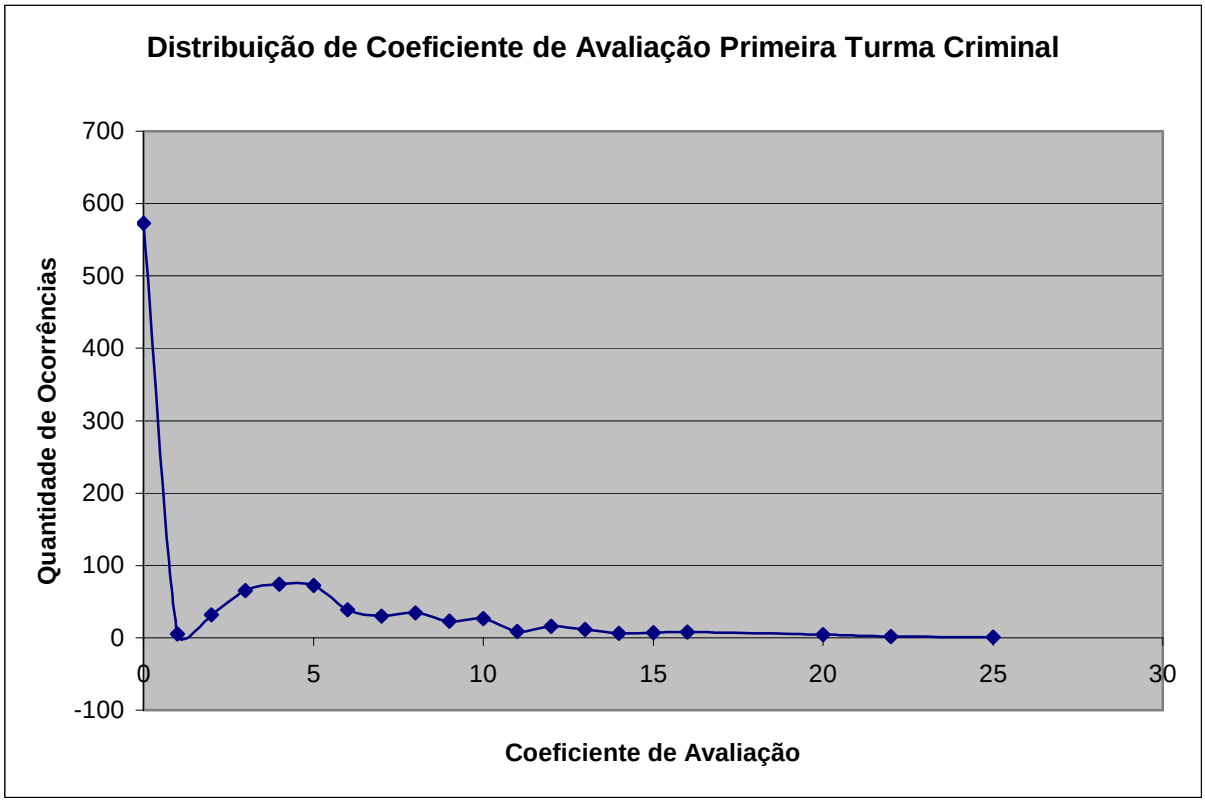


Figura 10 – Distribuição dos Coeficientes de Avaliação para a Primeira Turma Criminal

Após o cálculo do coeficiente de avaliação para os acórdãos da Primeira Turma Criminal, foi realizado o mesmo cálculo incrementando-se à base os documentos da Segunda Turma Criminal. O objetivo dessa segunda iteração é avaliar o modelo em um outro ambiente, com uma maior quantidade de documentos. O Quadro 10 abaixo apresenta as médias dos resultados para essa nova amostra:

Acórdão	Índices Manuais	Índices Automáticos	Congruências	Coeficiente de Avaliação
Médias	**8,7578**	**16,2103**	**0,8692**	**3,1675**

Quadro 10 – Cálculo dos Coeficientes de Avaliação para Base da Primeira e Segunda Turmas Criminais

Percebe-se que nos acórdãos da Segunda Turma Criminal há uma maior quantidade de descritores selecionados pelos analistas de jurisprudência, o que

acaba aumentando o valor da média da quantidade de índices manuais. Já em relação aos índices automáticos, o sistema de indexação automática mantém-se aproximado ao mesmo resultado anterior, demonstrando coerência em situações diferentes. A quantidade de congruências, ou seja, de descritores selecionados semelhantemente entre as duas indexações também apresenta uma ligeira melhora, o que também bem influencia o cálculo da média ponderada do coeficiente de avaliação. Uma importante observação cabe na escolha da média ponderada para o coeficiente de avaliação. A média simples ignora a freqüência dos valores, o que mascara o resultado. O fato de 573 acórdãos apresentarem 0 congruência na base da Primeira Turma Criminal, ou 1.119 documentos na base da segunda iteração, tem de ser considerado no cálculo da média.

O seguinte Quadro 11, portanto, foi construído com os valores dos coeficientes de avaliação para a base da primeira e segunda turmas criminais e sua distribuição:

Coeficiente de Avaliação	**Freqüência**
0	1119
1	6
2	88
3	174
4	192
5	159
6	103
7	62
8	78
9	53
10	60
11	32
12	34
13	22
14	13
15	13

16	20
18	4
19	3
20	7
21	1
22	2
23	2
25	1

Quadro 11 – Distribuição dos Coeficientes de Avaliação para a Base da Segunda Iteração

Informação essa que também pode ser plotada em um gráfico com a curva de distribuição de freqüência:

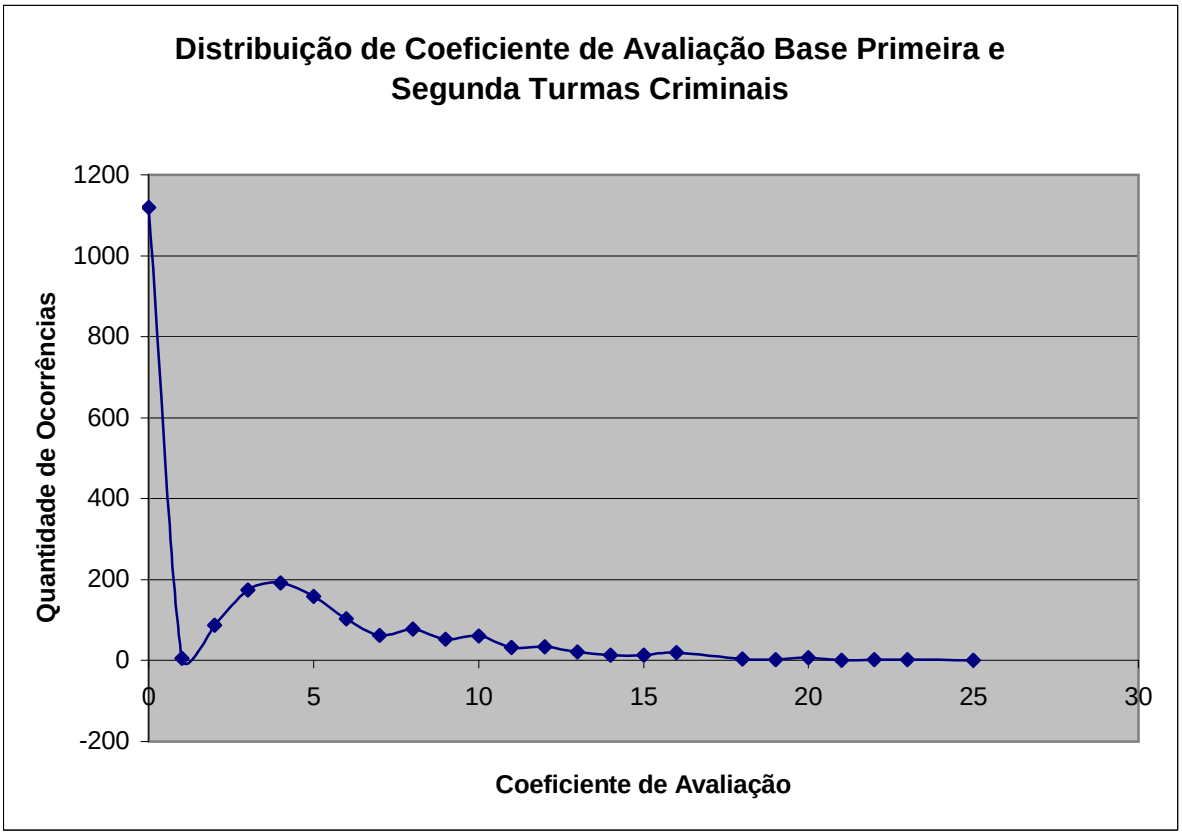


Figura 11 – Distribuição dos Coeficientes de Avaliação para a Base da Primeira e Segunda Turmas Criminais

As curvas de distribuição dos gráficos das figuras 10 e 11 apresentam o mesmo delineamento com aumento da freqüência, evidentemente, por efeito do

aumento da base. Isso demonstra que o comportamento do sistema de indexação automática não foi fortemente afetado pela alteração do ambiente.

Para finalizar a análise da comparação dos índices atribuídos, o processo foi realizado novamente com todos os 3.340 documentos da base. Para isso foram somados aos documentos da iteração anterior, quais sejam os acórdãos da Primeira Turma Criminal e Segunda Turma Criminal, os outros acórdãos de Direito Penal da base de jurisprudência do TJDFT, quais sejam os da Câmara Criminal, Conselho Especial e Conselho da Magistratura. Assim, foi possível avaliar o comportamento do modelo em uma última iteração. O Quadro 12 apresenta as médias dos resultados para essa última amostra.

Acórdão	Índices Manuais	Índices Automáticos	Congruências	Coeficiente de Avaliação
Médias	**8,6353**	**16,7595**	**0,8404**	**2,8928**

Quadro 12 – Cálculo dos Coeficientes de Avaliação para Base Completa

Há uma estabilização da variação das médias para a base completa. A quantidade de índices manuais manteve-se estável em relação à segunda iteração, e novamente o sistema de indexação automática foi coerente com a quantidade de descritores atribuídos. A congruência também não apresentou grande diferencial enquanto a média ponderada do coeficiente de avaliação aproximou-se do resultado da primeira base de testes. Demonstrou-se, assim, o quanto o modelo é estável ao aumento do banco de dados.

Seguem, para reforçar essa tese, Quadro 13 demonstrativo da distribuição dos coeficientes de avaliação pela freqüência dos acórdãos e o respectivo gráfico de dispersão plotado desse:

Coeficiente de Avaliação	Freqüência
0	1780
1	19
2	133

3	239
4	265
5	212
6	136
7	97
8	110
9	74
10	75
11	44
12	47
13	29
14	16
15	17
16	23
17	3
18	4
19	3
20	7
21	2
22	2
23	2
25	1

Quadro 13 – Distribuição dos Coeficientes de Avaliação para a Base de Acórdãos de Direito Penal do TJDFT

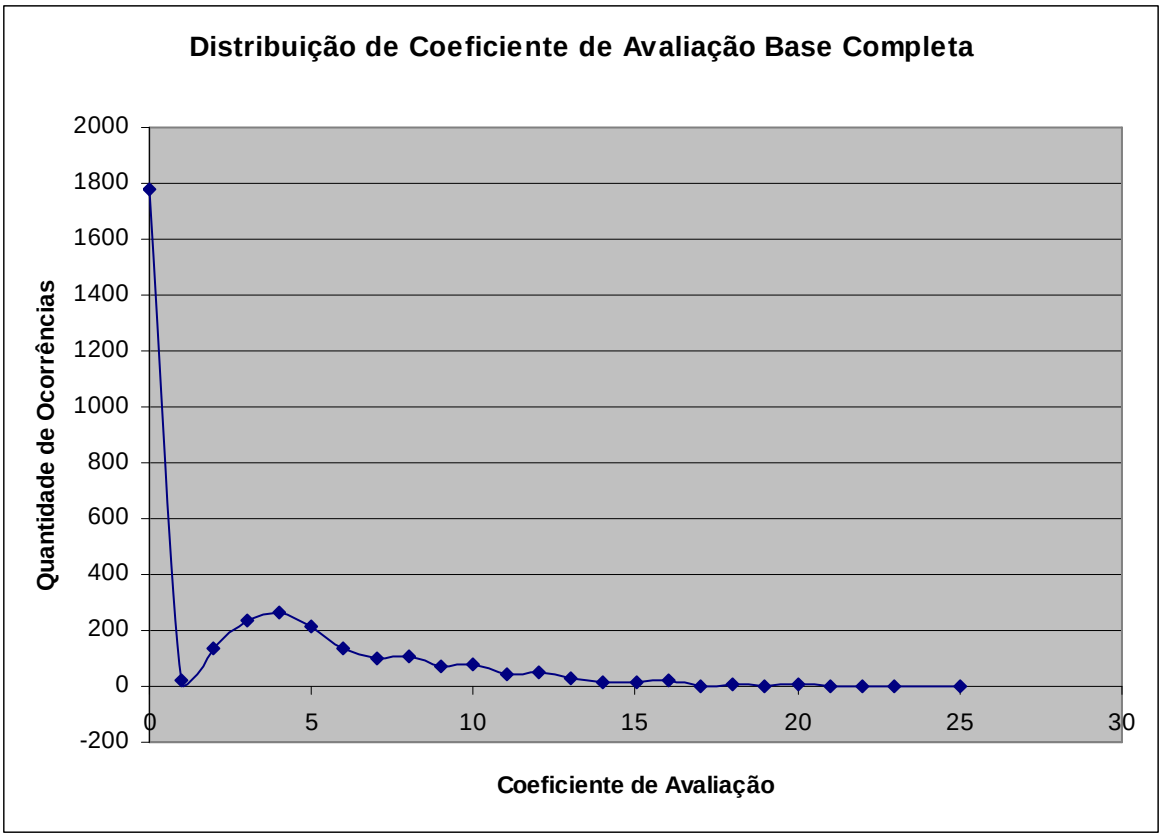


Figura 12 – Distribuição dos Coeficientes de Avaliação para a Base de Acórdãos de Direito Penal do TJDFT

Dessa forma, conclui-se que a comparação entre os índices selecionados pela indexação manual e automática apresentou resultados bastante díspares. A qualidade da indexação manual prejudicou o cálculo comparativo. O modelo, contudo, demonstrou-se coerente sob a alteração da quantidade de documentos, o que representa bom cumprimento de objetivos para um sistema de indexação automática. A grande diferença entre os descritores vai influenciar os resultados das pesquisas, e essa confirmação será analisada nas próximas seções.

6.2. COMPARAÇÃO DO ÍNDICE DE REVOCAÇÃO DE PESQUISAS

Para cálculo do índice de revocação, foram submetidas pesquisas à base de dados de jurisprudência do TJDFT semelhantes às informações utilizadas para descoberta de relevância dos documentos. Sabendo-se quais documentos são relevantes para uma pesquisa do tipo “tráfico de entorpecente”, por exemplo, é possível calcular os índices nas duas bases de indexação e comparar os resultados.

Para cálculo da revocação foi utilizado apenas o extrato da base representado pelos acórdãos da Primeira Turma Criminal. Isso se deu porque para calcular a revocação é necessário reconhecer os documentos relevantes da base, e essa tarefa demanda um tempo considerável. Foi possível, portanto, classificar a relevância apenas desse extrato.

Foi gerado um quadro, dessa maneira, com os parâmetros de buscas e seus respectivos retornos em quantidade. As pesquisas foram realizadas nos dois bancos de dados, de indexação manual e automática, e o cálculo da revocação também contabilizado em ambos. O Quadro 14 apresenta esses resultados.

Parâmetro de Pesquisa	QtdDRB	QtdDRRIM	IRIM	QtdDRRIA	IRIA
Aborto	2	2	1,00	1	0,50
Acidente de Trânsito	45	27	0,06	26	0,57
Atentado Violento ao Pudor	82	27	0,32	82	1,00
Constrangimento Ilegal	187	93	0,49	183	0,97
Corrupção de Menor	4	4	1,00	4	1,00
Crime Hediondo	160	111	0,69	160	1,00
Denunciação Caluniosa	3	2	0,66	2	0,66
Estelionato	30	27	0,90	21	0,70
Estupro	18	18	1,00	15	0,83
Extorsão	10	6	0,60	5	0,50
Furto de Veículo	11	11	1,00	8	0,72
Furto Qualificado	75	65	0,86	64	0,85
Habeas Corpus	320	320	1,00	311	0,97
Homicídio Qualificado	117	63	0,53	58	0,49
Latrocínio	36	36	1,00	28	0,77
Lesão Corporal	55	34	0,61	45	0,81
Porte de Arma de Fogo	250	123	0,49	233	0,93
Receptação	22	22	1,00	18	0,81
Roubo à Residência	48	23	0,47	36	0,75
Roubo Qualificado	200	146	0,73	109	0,54
Seqüestro	5	5	1,00	4	0,80

Tortura	22	22	1,00	20	0,90
Tráfico de Entorpecente	127	68	0,53	127	1,00
Uso de Documento Falso	20	16	0,80	20	1,00
Violação de Direito Autoral	4	4	1,00	4	1,00
Médias Ponderadas dos Índices			**0,67**		**0,85**

Quadro 14 – Comparação dos Índices de Revocação de Pesquisas

Onde as colunas são representadas por acrônimos com os seguintes significados:

QtdDRB : Quantidade de documentos relevantes na base

QtdDRRIM : Quantidade de documentos relevantes recuperados na base de indexação manual

IRIM : Índice de revocação na base de indexação manual

QtdDRRIA : Quantidade de documentos relevantes recuperados na base de indexação automática

IRIA : Índice de revocação na base de indexação automática

Os resultados apresentados no Quadro 14 são plotados no seguinte gráfico de barras para facilitar a visualização e discussão dos resultados:

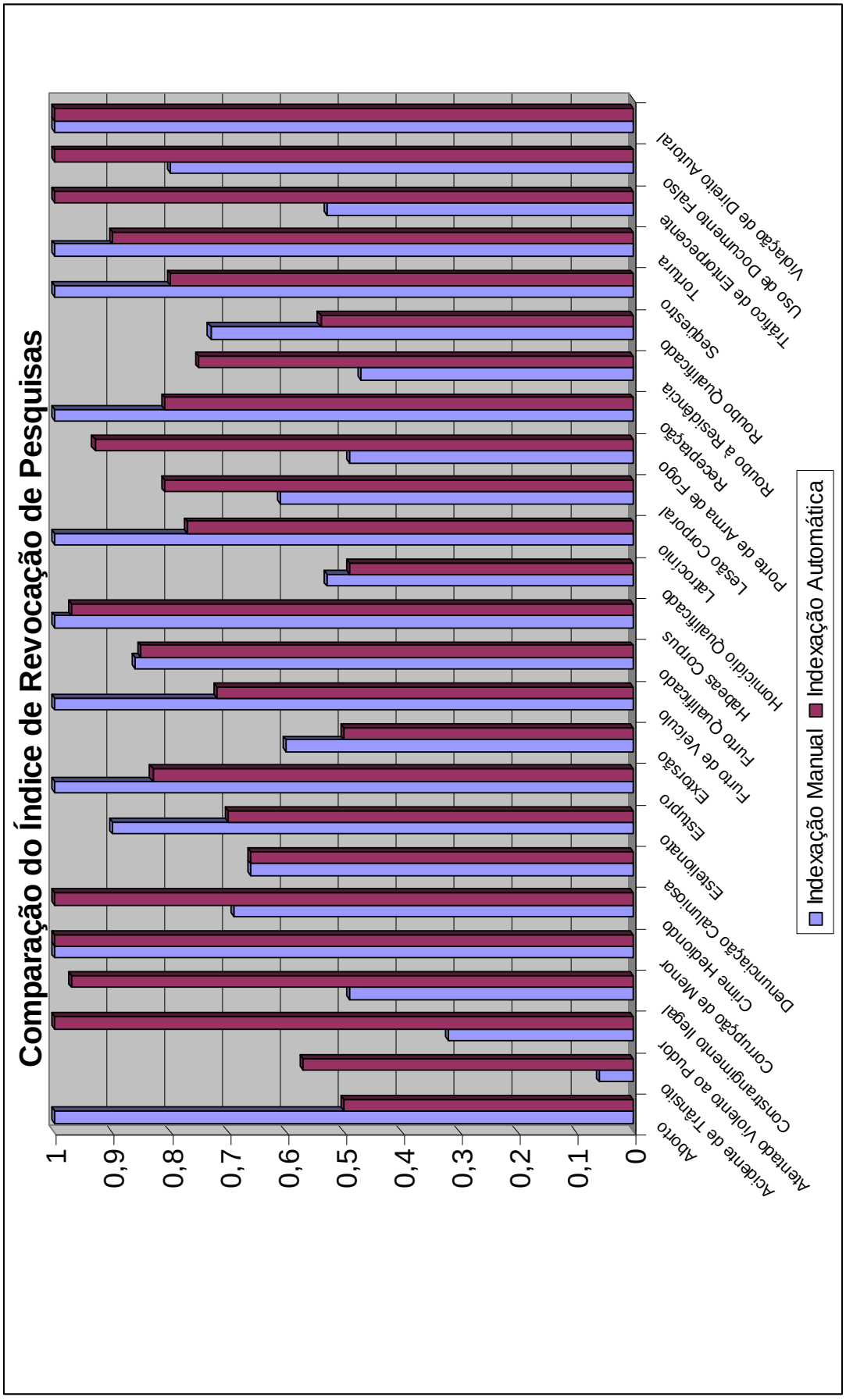


Figura 13 – Comparação dos Índices de Revocação de Pesquisas

O cálculo do índice de revocação apresenta uma avaliação precisa para a qualidade da indexação automática. De maneira geral, as bases indexadas automaticamente e manualmente chegaram a resultados de pesquisas semelhantes, com o banco automático demonstrando média ligeiramente melhor. Isso se justifica por meio da arquitetura de dados existente na jurisprudência do TJDFT. Os documentos de acórdãos são manualmente indexados por meio de seu espelho, ementa, e seleção dos analistas. O espelho contém informações parametrizadas no banco de dados, tais como a classe processual, o nome das partes e advogados, o órgão julgador do processo, o quorum do julgamento, as referências legislativas utilizadas pelos magistrados, entre diversas outras. A ementa é o resumo do julgado, e esse resumo é inteiramente utilizado para extração de descritores do documento, sem nenhuma análise adicional. Finalmente, um campo indexação é disponibilizado aos analistas onde esses adicionam descritores para melhorar os índices de pesquisa.

Dessa forma, o motor de busca utiliza várias informações para realizar as pesquisas, e esses outros campos minimizam o efeito do campo indexação. Assim, a indexação automática oferece alguma melhoria onde ela apresenta diferencial. Os melhores exemplos são os crimes que se descrevem por meio das estruturas utilizadas para PLN, como “atentado violento ao pudor” ou “crime hediondo”. Nessas pesquisas, a indexação automática recupera mais documentos relevantes do que a indexação manual, pois as únicas chances dessa última encontrar documentos desse tipo são o relator expor tal termo na ementa ou o indexador recupera-lo na indexação, já que o texto completo não é utilizado. A indexação automática sempre vai recuperar o descritor, pois o mesmo está no texto completo e é reconhecido pelo tesauro.

Além disso, como já dito anteriormente, as relações de equivalência que o tesauro apresenta melhoram significativamente o índice de revocação das pesquisas. Na indexação manual, para recuperar os documentos relevantes para “tráfico de drogas”, devem-se realizar pesquisas utilizando todas as variações de plural e singular de “droga”, “entorpecente”, “psicotrópico” e quaisquer outros sinônimos. A expressão “tráfico de entorpecentes”, na indexação automática,

abrange todos os documentos, e o sistema realiza o processo automaticamente. A associação do motor de busca ao tesauro garante que mesmo que o usuário final informe ao sistema um parâmetro de busca como “tráfico de drogas”, o próprio programa reconhece o texto, altera para a expressão equivalente e submete a pesquisa.

Nos crimes cuja descrição não se encaixe nos padrões selecionados para PLN, aqueles que são apenas um substantivo, como “aborto”, “estelionato” ou “seqüestro”, por exemplo, a indexação manual apresenta resultados ligeiramente melhores do que a automática. Essa última não recupera um descritor desse formato mesmo varrendo o texto completo, porém as outras informações utilizadas para indexação do acórdão não permitem que tais textos sejam completamente ignorados nas pesquisas. Há, entretanto, a situação em que o tesauro relaciona um termo equivalente fora do padrão de pesquisa. As expressões “fato delituoso” e “infração penal”, por exemplo, são equivalentes a “delito”. O sistema ao encontrar qualquer dessas estruturas, converte para o equivalente e o utiliza para indexação. Na pesquisa, o mesmo processo é realizado. Assim, mesmo na indexação automática a unidade lexical “delito” é utilizada como descritor e os documentos são recuperados com melhor índice de revocação do que na indexação manual.

A conclusão a que se chega na análise da revocação, portanto, é que a indexação automática, no que tem de mais forte, traz muito benefício para o resultado das pesquisas. Mesmo nas situações em que ela não apresenta grande diferencial, a perda de informação é pequena, de forma que os resultados ficam bastante próximos dos atingidos pela indexação manual. Para os assuntos com poucas amostras na base, como “corrupção de menor” ou “violação de direito autoral”, por exemplo, as duas bases demonstraram resultados equivalentes, enquanto as maiores diferenças identificadas foram a favor da indexação automática, como em “constrangimento ilegal” ou “porte de arma de fogo”.

6.3. COMPARAÇÃO DO ÍNDICE DE PRECISÃO DE PESQUISAS

A análise dos índices de precisão de pesquisas deve ser precedida pela exposição de uma falha no motor de busca de jurisprudência do TJDFT. Essa

limitação é reportada pelos analistas de pesquisa daquele setor como o maior problema apresentado pela ferramenta, o qual reflete diretamente na precisão de algumas pesquisas.

Para indexação, cada descritor selecionado é passado por um filtro. Esse procedimento é realizado devido à infra-estrutura de banco de dados utilizada pelo TJDFT a qual, por ser muito antiga, possui algumas restrições. O filtro tem por objetivo remover todos os caracteres especiais das unidades léxicas ou estruturas selecionadas para descritores. Cada um desses caracteres é trocado por um espaço em branco ‘ ‘, o qual não oferece dificuldades ao banco. Assim, um descritor do tipo “queixa-crime” é indexado como “queixa crime”, ou então “10,87%” como “10 87”.

O assunto “10,87%” refere-se a um mandado de segurança do funcionalismo público federal para reposição salarial do Plano Collor. Essa é uma discussão recorrente no TJDFT, e essa pesquisa é submetida à base de jurisprudência com grande freqüência. A mesma não faz parte do escopo dessa pesquisa, mas a idiossincrasia que ocorre com ela é muito reportada pelos analistas de pesquisa de jurisprudência e ajuda a explicar alguns resultados.

Ao ignorar tais caracteres, a pesquisa de determinados assuntos fica prejudicada. Ao submeter o parâmetro “10,87%” para o motor de busca, esse pesquisa todos os documentos que possuem os termos “10” e “87” trazendo um enorme ruído para o resultado. Existe um operador denominado “ADJ” o qual indica a distância de adjacência entre os parâmetros de pesquisa. Assim, submetendo-se “10 adj1 87” aproxima-se do resultado desejado, mas ainda sim não é satisfatório o índice de precisão.

Esse problema reflete-se na indexação automática proposta por essa pesquisa. A pesquisa por “queixa-crime”, por exemplo, devolve um resultado com bastante ruído. Isso ocorre porque uma grande quantidade dos documentos da base possui a unidade lexical “queixa”, e praticamente todos os acórdãos de direito penal têm a palavra “crime”. Logo, essa abordagem prejudica fortemente as buscas, e ajustes no motor são demandados para melhorar, de maneira geral, os resultados oferecidos pelo TJDFT.

Postas essas considerações, segue Quadro 15 com os parâmetros de pesquisa, resultados e cálculos dos índices de precisão nas bases indexadas manualmente e automaticamente da Primeira Turma Criminal:

Parâmetro de Pesquisa	**QtdDRIM**	**QtdDRRIM**	**IPIM**	**QtdDRIA**	**QtdDRRIA**	**IPIA**
Aborto	2	2	1,00	1	1	1,00
Acidente de Trânsito	30	27	0,90	27	26	0,96
Atentado Violento ao Pudor	28	27	0,96	82	82	1,00
Constrangimento Ilegal	99	93	0,93	184	183	0,99
Corrupção de Menor	15	4	0,26	4	4	1,00
Crime Hediondo	125	111	0,88	163	160	0,98
Denunciação Caluniosa	3	2	0,66	3	2	0,66
Estelionato	27	27	1,00	21	21	1,00
Estupro	18	18	1,00	15	15	1,00
Extorsão	10	6	0,60	7	5	0,71
Furto de Veículo	20	11	0,55	8	8	1,00
Furto Qualificado	66	65	0,98	66	64	0,96
Habeas Corpus	327	320	0,97	311	311	1,00
Homicídio Qualificado	73	63	0,86	67	58	0,86
Latrocínio	36	36	1,00	28	28	1,00
Lesão Corporal	40	34	0,85	45	45	1,00
Porte de Arma de Fogo	123	123	1,00	238	233	0,97
Receptação	22	22	1,00	18	18	1,00

Roubo à Residência	46	23	0,50	37	36	0,97
Roubo Qualificado	160	146	0,91	111	109	0,98
Seqüestro	5	5	1,00	5	4	0,80
Tortura	22	22	1,00	22	20	0,90
Tráfico de Entorpecente	71	68	0,95	127	127	1,00
Uso de Documento Falso	30	16	0,53	28	20	0,71
Violação de Direito Autoral	4	4	1,00	4	4	1,00
Médias Ponderadas dos Índices			**0,92**			**0,97**

Quadro 15 – Comparação dos Índices de Precisão de Pesquisas na Base da Primeira Turma Criminal

Onde as colunas são representadas por acrônimos com os seguintes significados:

QtdDRIM : Quantidade de documentos recuperados na indexação manual

QtdDRRIM : Quantidade de documentos relevantes recuperados na base de indexação manual

IPIM : Índice de precisão na base de indexação manual

QtdDRIA : Quantidade de documentos recuperados na indexação automática

QtdDRRIA : Quantidade de documentos relevantes recuperados na base de indexação automática

IPIA : Índice de precisão na base de indexação automática

Os resultados apresentados no Quadro 15 são plotados no seguinte gráfico para facilitar a visualização e discussão dos resultados:

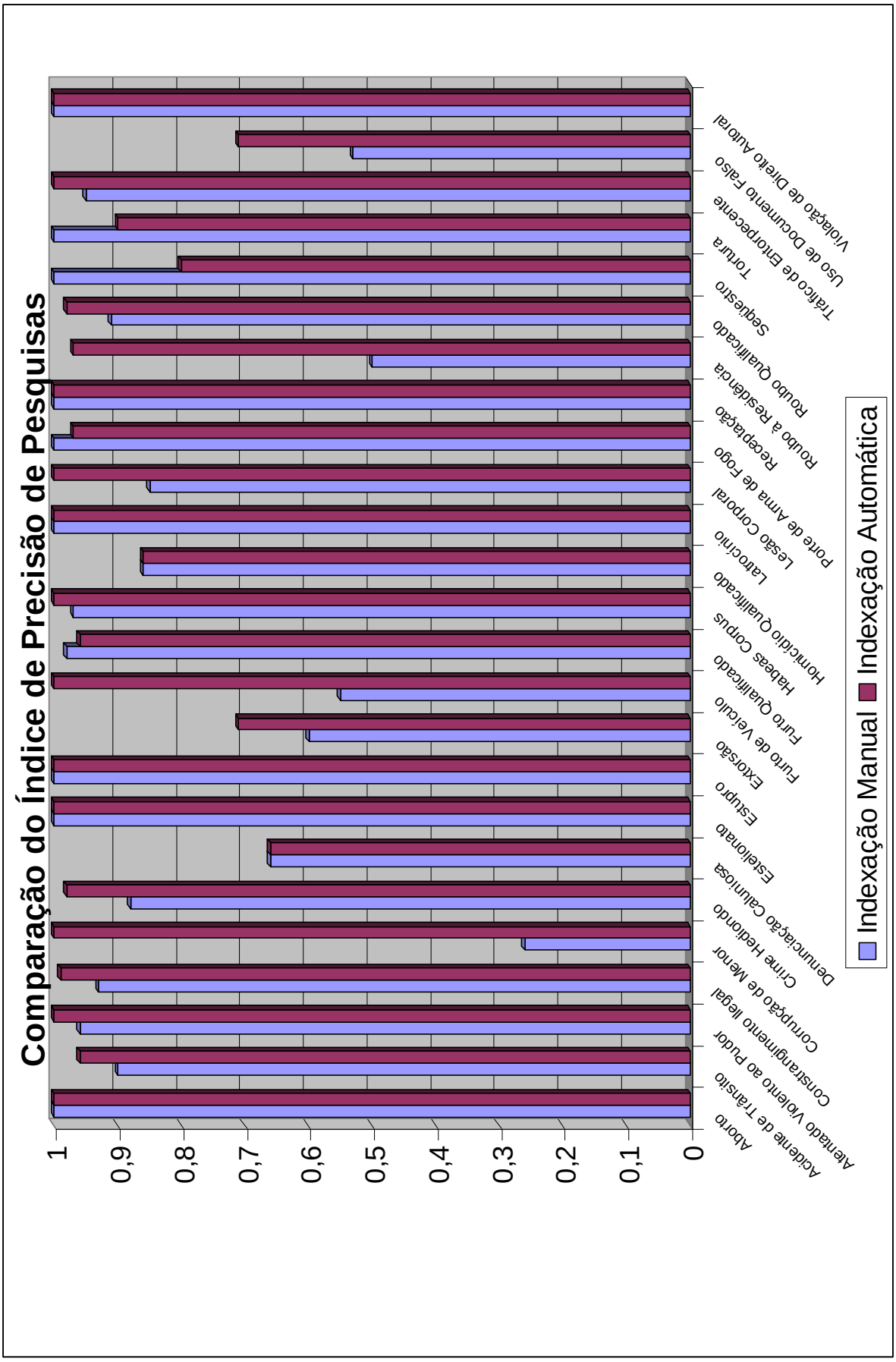


Figura 14 – Comparação dos Índices de Precisão de Pesquisas na Base da Primeira Turma Criminal

Os índices de precisão, assim como os índices de revocação, também apresentaram resultados semelhantes entre as bases de indexação automática e manual. O defeito apresentado pelo motor de busca não se mostrou efetivo uma vez que os parâmetros de pesquisa selecionados não são palavras compostas, nem contêm números. Assim pôde-se obter um retorno satisfatório para as buscas.

De maneira geral, os cálculos da precisão foram melhores do que os cálculos de revocação na base da Primeira Turma Criminal. Enquanto nessa última as médias ponderadas para indexação manual e automática foram 0,67 e 0,85 respectivamente, naquela os resultados foram 0,92 e 0,97. Isso demonstra o quanto a indexação automática melhorou os índices de precisão que já apresentavam bons resultados, conquanto algumas idiossincrasias fossem inseridas no processo.

Uma delas é representada pela pesquisa com parâmetro “Distrito Federal”. Na base de indexação manual, são retornados os acórdãos cuja parte autora ou ré do processo seja o Distrito Federal. Já a base de indexação automática retorna todos os 1.037 documentos. Isso ocorre porque é praxe nos relatórios e votos dos magistrados a menção ao Tribunal de Justiça do Distrito Federal e dos Territórios. Com o padrão de estruturas Substantivo – Preposição – Substantivo, a expressão “Tribunal de Justiça” é recuperada pelo analisador de sintagmas, porém como a mesma não existe no tesauro jurídico, ela é ignorada. Já a construção “Distrito Federal” é recuperada pelo padrão Substantivo – Adjetivo, e essa existe no tesauro. Dessa forma, todos os acórdãos da base possuem um descritor “df”, pois o tesauro apresenta esse termo equivalente ao “Distrito Federal”.

Essa situação não é um problema impeditivo uma vez que não é realizada tradicionalmente pesquisa de partes autoras ou rés de processos na jurisprudência. Para isso há pesquisas específicas que não varrem texto, mas sim bancos de dados normalizados dos processos judiciais e administrativos do TJDFT. Logo, uma pesquisa com parâmetro “Distrito Federal” na base de jurisprudência é fictícia, contudo seu resultado na base manual apresenta um índice de precisão enormemente melhor que na base de indexação automática.

Ainda assim, entretanto, isso aponta para outros resultados não esperados que possam ter sido inseridos indesejavelmente pela indexação automática. Como mencionado no referencial teórico, a união de abordagens de indexação automática diferentes pode oferecer algumas vantagens. Como o descritor foi selecionado para todos os documentos, então uma varredura estatística poderia detectar esse evento e excluir tal índice. Essa pesquisa não utilizou tal recurso, mas percebeu-se a necessidade de algum mecanismo externo para controlar esse tipo de evento.

Por outro lado, os melhores resultados de índice de precisão foram obtidos nos parâmetros de pesquisa que apresentam os mesmos padrões das estruturas de busca automática. A pesquisa por “tráfico de entorpecente” na base de indexação manual apresenta um bom índice de precisão, apesar do resultado ser bastante incompleto sob o ponto de vista da revocação. Já no banco de indexação automática, o resultado da busca é mais completo e, ainda assim, mais preciso. Esse resultado é possível por meio da utilização recorrente das relações de equivalência oferecidas pelo tesauro, o que melhora a cobertura sem prejudicar a precisão.

Concluída a análise da primeira iteração, passou-se à pesquisa na base de dados mais populosa. Essa contém os acórdãos de Direito Penal da Primeira Turma Criminal e da Segunda Turma Criminal. Um novo Quadro 16 com os parâmetros de pesquisa, quantidade de documentos retornados e cálculo dos índices de precisão é apresentado abaixo:

Parâmetro de Pesquisa	**QtdDRIM**	**QtdDRRIM**	**IPIM**	**QtdDRIA**	**QtdDRRIA**	**IPIA**
Aborto	4	4	1,00	2	2	1,00
Acidente de Trânsito	57	53	0,92	51	48	0,94
Atentado Violento ao Pudor	67	65	0,97	139	139	1,00

Constrangimento Ilegal	152	140	0,92	282	279	0,98
Corrupção de Menor	35	12	0,34	6	6	1,00
Crime Hediondo	217	203	0,93	235	215	0,91
Denunciação Caluniosa	6	5	0,83	10	10	1,00
Estelionato	67	67	1,00	54	54	1,00
Estupro	57	57	1,00	49	48	0,97
Extorsão	14	11	0,78	10	7	0,70
Furto de Veículo	40	30	0,75	26	25	0,96
Furto Qualificado	168	166	0,98	160	159	0,99
Habeas Corpus	549	537	0,97	530	527	0,99
Homicídio Qualificado	183	167	0,91	168	155	0,92
Latrocínio	83	82	0,98	72	72	1,00
Lesão Corporal	79	70	0,88	106	100	0,94
Porte de Arma de Fogo	265	260	0,98	538	533	0,99
Receptação	42	39	0,92	37	36	0,97
Roubo à Residência	70	32	0,45	51	50	0,98
Roubo Qualificado	364	358	0,98	282	274	0,97
Seqüestro	8	8	1,00	7	7	1,00
Tortura	37	37	1,00	33	33	1,00
Tráfico de Entorpecente	151	139	0,92	257	255	0,99
Uso de Documento Falso	40	24	0,60	30	23	0,76
Violação de Direito Autoral	6	6	1,00	6	6	1,00

Médias Ponderadas dos Índices	**0,94**			**0,97**

Quadro 16 – Comparação dos Índices de Precisão de Pesquisas na Base da Segunda Iteração

Onde as colunas são representadas por acrônimos com os seguintes significados:

QtdDRIM : Quantidade de documentos recuperados na indexação manual

QtdDRRIM : Quantidade de documentos relevantes recuperados na base de indexação manual

IPIM : Índice de precisão na base de indexação manual

QtdDRIA : Quantidade de documentos recuperados na indexação automática

QtdDRRIA : Quantidade de documentos relevantes recuperados na base de indexação automática

IPIA : Índice de precisão na base de indexação automática

De maneira geral, a avaliação da segunda etapa apresentou resultados bastante semelhantes aos da primeira iteração. Alguns eventos interessantes merecem observação mais cuidadosa, contudo. Para o parâmetro de pesquisa “crime hediondo” houve uma quantidade de documentos recuperada menor do que o esperado. Considerando-se que a base da segunda iteração tem o dobro de documentos do banco de dados da Primeira Turma Criminal, esperava-se uma quantidade de aproximadamente 326 registros retornados na indexação automática. Esse número revelou-se 235, indicando uma provável perda de índice de revocação. O índice de precisão manteve-se alto, mas apresentou leve queda em relação à base anterior. Isso demonstra que o modelo não foi tão efetivo na base maior quanto foi na base menor para esse parâmetro.

Já a pesquisa por “denunciação caluniosa” demonstrou uma quantidade de documentos relevantes recuperados na base de indexação automática maior do que na base de indexação manual. No banco de dados da Primeira Turma Criminal, tanto a indexação manual quanto automática apresentaram dois documentos

relevantes recuperados. Já no banco da Primeira e Segunda Turma Criminal, a indexação manual retornou cinco acórdãos, enquanto a automática, dez registros. Isso indicou uma melhora significativa no índice de revocação da pesquisa, sem nenhum prejuízo para o cálculo da precisão. O modelo, para esse parâmetro, foi bem mais efetivo na base da segunda iteração.

Essa variação da qualidade do comportamento do modelo foi observada em alguns outros parâmetros de pesquisa. A pesquisa por “*habeas corpus*” retornou menos registros do que o esperado na base da segunda iteração, a qual é duas vezes maior do que a base da primeira. O cálculo do índice de precisão é precisamente o mesmo nos dois bancos, mas a revocação tem forte indício de ter sido prejudicada. Outrossim, enquanto a pesquisa “homicídio qualificado” fora esperada para 120 acórdãos em média, retornaram-se 168 registros, sendo 155 relevantes. O banco de dados da Primeira e Segunda Turma Criminal retornou resultado mais favorável para esse parâmetro de busca.

Segue Figura 15 que apresenta o gráfico de barras plotado para o cálculo dos índices de precisão da base da segunda iteração do processo, qual seja, a base de acórdãos de Direito Penal da Primeira e Segunda Turma Criminal. Esse gráfico utiliza os valores do Quadro 16.

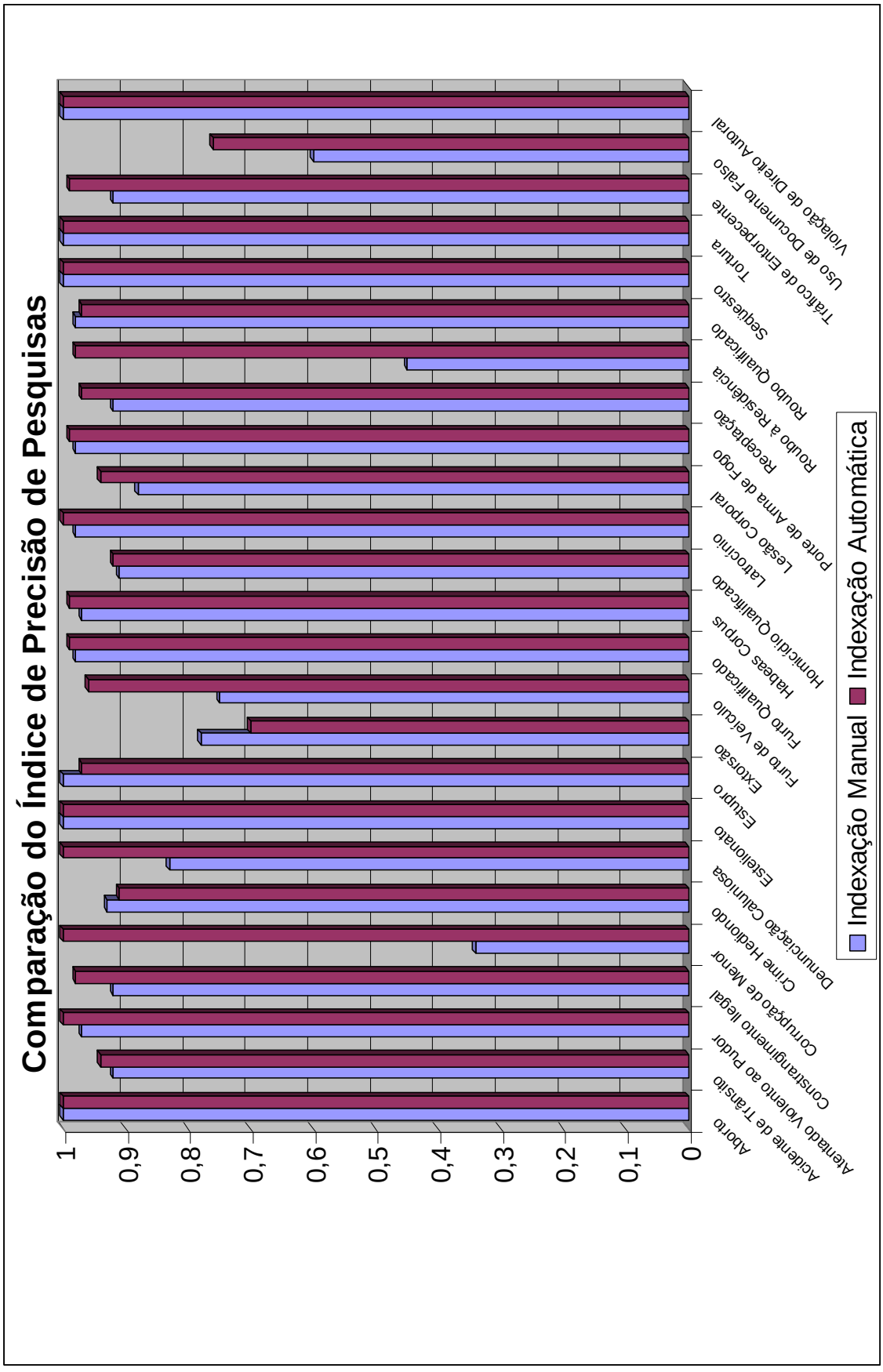


Figura 15 – Comparação dos Índices de Precisão de Pesquisas na Base da Primeira e Segunda Turma Criminal

Concluídas as análises da primeira e segunda iteração, passou-se à pesquisa na base de dados da última rodada. Essa contém os acórdãos de Direito Penal de todos os órgãos do TJDFT. O Quadro 17 apresenta novamente os parâmetros de pesquisa, quantidade de documentos retornados e cálculo dos índices de precisão para essa base mais completa.

Parâmetro de Pesquisa	**QtdDRIM**	**QtdDRRIM**	**IPIM**	**QtdDRIA**	**QtdDRRIA**	**IPIA**
Aborto	5	4	0,80	2	2	1,00
Acidente de Trânsito	83	77	0,92	80	77	0,96
Atentado Violento ao Pudor	74	73	0,98	156	152	0,97
Constrangimento Ilegal	204	202	0,99	416	415	0,99
Corrupção de Menor	72	21	0,29	9	9	1,00
Crime Hediondo	242	227	0,93	272	248	0,91
Denunciação Caluniosa	9	7	0,77	14	11	0,78
Estelionato	81	81	1,00	64	63	0,98
Estupro	68	68	1,00	58	56	0,96
Extorsão	17	14	0,82	12	8	0,66
Furto de Veículo	47	32	0,68	27	26	0,96
Furto Qualificado	188	183	0,97	181	179	0,98
Habeas Corpus	755	736	0,97	745	739	0,99
Homicídio Qualificado	194	176	0,90	186	171	0,91
Latrocínio	87	84	0,96	77	76	0,98
Lesão Corporal	123	110	0,89	162	155	0,95
Porte de Arma	321	314	0,97	665	657	0,98

de Fogo						
Receptação	58	55	0,94	51	47	0,92
Roubo à Residência	90	43	0,47	81	76	0,93
Roubo Qualificado	418	408	0,97	345	331	0,95
Seqüestro	15	15	1,00	13	12	0,92
Tortura	47	47	1,00	43	41	0,95
Tráfico de Entorpecente	203	176	0,86	338	334	0,98
Uso de Documento Falso	44	25	0,56	42	37	0,88
Violação de Direito Autoral	10	10	1,00	8	8	1,00
Médias Ponderadas dos Índices			**0,93**			**0,96**

Quadro 17 – Comparação dos Índices de Precisão de Pesquisas na Base de Direito Penal do TJDFT

Onde as colunas são representadas por acrônimos com os seguintes significados:

QtdDRIM : Quantidade de documentos recuperados na indexação manual

QtdDRRIM : Quantidade de documentos relevantes recuperados na base de indexação manual

IPIM : Índice de precisão na base de indexação manual

QtdDRIA : Quantidade de documentos recuperados na indexação automática

QtdDRRIA : Quantidade de documentos relevantes recuperados na base de indexação automática

IPIA : Índice de precisão na base de indexação automática

Também de maneira geral, o cálculo do índice de precisão em pesquisas na base completa de Direito Penal do TJDFT apresentou resultados semelhantes aos resultados obtidos nas iterações anteriores. Houve uma pequena queda de um ponto percentual na média ponderada da indexação automática dessa última rodada. Isso é explicado pelo fato dessa base mais completa possuir uma grande quantidade de documentos bastante diferentes daqueles utilizados para o treinamento da ferramenta. O modelo comportou-se bem sob esse aspecto, onde novos documentos, alguns deles até com desvios de macro-estrutura, foram bem recebidos e analisados pela ferramenta de PLN.

Segue Figura 16 apresentando o gráfico de barras gerado a partir do Quadro 17 com os resultados da comparação dos índices de precisão de pesquisas realizadas nas bases de indexação manual e automática do banco de dados completo de Direito Penal do TJDFT.

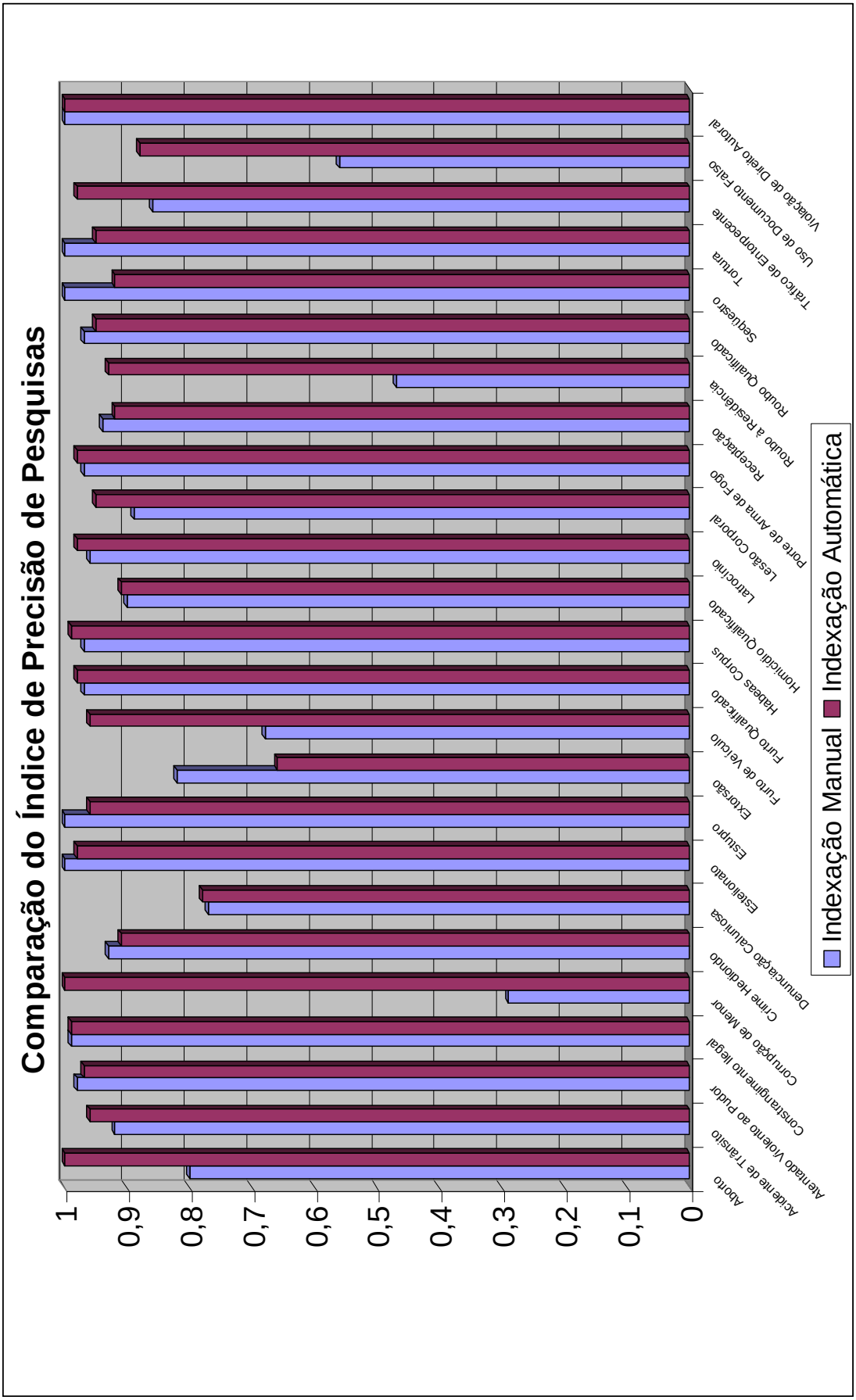


Figura 16 – Comparação dos Índices de Precisão de Pesquisas na Base de Direito Penal do TJDFT

Uma interessante observação deve ser feita em alguns parâmetros de pesquisa dessa base. Como a base completa possui 3.340 registros, enquanto a base da iteração anterior possuía 2.248 documentos, espera-se, por extensão, que a quantidade de acórdãos retornados em qualquer pesquisa tenha aproximadamente 50% a mais registros na última base do que na anterior. Isso é válido considerando que haja uma distribuição uniforme dos assuntos de Direito entre os órgãos julgadores do Tribunal, o que é garantido pela Distribuição Processual discutida no referencial teórico.

Em alguns parâmetros, contudo, tais como “furto de veículo”, o aumento da quantidade de documentos foi bastante menor do que o esperado. Isso se dá porque os órgãos colegiados utilizados para composição da terceira base são Câmara Criminal, Conselho Especial e Conselho da Magistratura, os quais apresentam características diferentes das Turmas. Essas julgam recursos de causas de Primeira Instância, enquanto aquelas julgam recursos de causas do próprio Segundo Grau ou originárias no Segundo Grau. Isso significa que os órgãos recursais e originários tratam de ações mais onerosas, ou contra autoridades. Dificilmente uma ação de “furto de veículo” ou “roubo à residência” ou também “uso de documento falso” chega nesses colegiados. Um pedido de “*habeas corpus*”, entretanto, é solicitado linearmente em todos esses órgãos julgadores. Por isso, esse parâmetro, entre outros, apresentou um crescimento próximo do esperado na última iteração.

Concluindo, portanto, os resultados obtidos com os cálculos dos índices de precisão das pesquisas na base de indexação automática foram equivalentes aos mesmos no banco de indexação manual, nas três iterações realizadas com bancos de dados incrementais. Percebe-se que, de maneira geral, o motor de busca textual do TJDFT oferece bom retorno às pesquisas submetidas que não estejam em determinados padrões, como aqueles discutidos anteriormente que têm caracteres especiais. A indexação automática não prejudicou essa qualidade, apresentando ligeira melhora.

7. CONCLUSÕES

Essa pesquisa conclui que a indexação automática mostrou-se equivalente à indexação manual para o contexto analisado. Algumas pequenas diferenças apresentaram-se quanto aos índices de revocação e precisão a favor da indexação manual para alguns tipos de parâmetros de pesquisa, enquanto outros parâmetros beneficiaram a indexação automática. De maneira geral, contudo, os resultados de ambas indexações mostraram-se bastante semelhantes. O comportamento do modelo perante o aumento da base foi coerente. Isso demonstra que o grande benefício oferecido pela indexação automática, qual seja a capacidade de realizar a tarefa de indexação perante o aumento da demanda do Judiciário sem aumento de custos, foi atingido. É possível reduzir o tamanho das equipes de indexação, dirimir gastos com treinamento e manter uma base consistente de indexação para documentos.

O modelo construído baseado em PLN mostrou resultados favoráveis. As abordagens estritamente estatísticas têm sido pesquisadas há bastante mais tempo do que as lingüísticas, porém essas têm potencialidade para renovar e melhorar os produtos existentes. A garantia da qualidade do modelo, entretanto, depende profundamente do esquema de representação do conhecimento utilizado. O tesauro do STJ escolhido para essa pesquisa atendeu a contento a demanda.

Com a decisão de aplicação do modelo em incrementos no banco de testes, cinco órgãos colegiados do TJDFT foram cobertos pelo sistema. Dessa forma, aumentou-se a diversidade de linguagem na construção dos textos dos acórdãos. Considerando que a segunda instância possui trinta e cinco membros, metade desse contingente teve pelo menos algum de seus acórdãos indexado automaticamente, o que representa boa extensão da base.

Entre as limitações desse estudo encontra-se a seleção da base de testes. Foi definido o escopo de Direito Penal, o qual representa uma pequena extração da base dos acórdãos do TJDFT. Conquanto o modelo tenha apresentado bons resultados para essa área do Direito, não foram realizados testes em outras

bases. A extensão dos magistrados utilizados foi ampla, mas a linguagem utilizada para o Direito Criminal por vezes apresenta diferenças estruturais em relação ao Direito Civil, Previdenciário ou Tributário, por exemplo. Isso pode degradar a efetividade do sistema, mas tal teste não foi realizado.

Além disso, não foram realizadas pesquisas qualitativas para demonstração dos resultados finais com indexadores da jurisprudência ou usuários finais do TJ. Os comentários dos analistas de jurisprudência poderiam corroborar ou criticar os resultados do modelo com a opinião dos especialistas da área, enquanto os usuários finais de pesquisas do TJDFT poderiam definir o grau de melhoria ou prejuízo oferecido em ambiente real de produção. Essas tarefas não foram realizadas por causa do tempo de execução dessa pesquisa, no entanto trariam ganho para a avaliação do modelo.

Como trabalhos futuros sugere-se a expansão do modelo para outras áreas do Direito. Utilizando o mesmo tesauro do STJ, podem ser realizados testes em acórdãos de Direito de outros ramos. Com um pequeno esforço de treinamento do analisador de PLN para outro tipo de vocabulário, é esperado que o sistema também se comporte bem em outras linhas da legislação. Além disso, com esse mesmo esquema de representação pode-se expandir a aplicação para acórdãos de outros tribunais.

Uma outra proposta é de aplicar o sistema de indexação automática para outros documentos jurídicos no âmbito dos tribunais, que não apenas os acórdãos. Os despachos monocráticos ou juízos de admissibilidade são decisões que também geram documentos com relevância para pesquisas textuais. Tais expedientes, se indexados automaticamente, podem trazer os mesmos benefícios que a base de jurisprudência usufrui. Para isso, o sistema deve ser adaptado para análise da macro-estrutura textual desses novos documentos.

E ainda, qualquer área do conhecimento que possua um esquema de representação formalizado pode ser automaticamente indexada com o modelo proposto. Na área de ciência da informação ou medicina, por exemplo, a produção bibliográfica pode ser indexada computacionalmente melhorando os índices de

revocação e precisão de pesquisas textuais submetidas a suas bases. A utilização desse tipo de abordagem para pré-processamento de textos, com o objetivo de reduzir o tamanho da BOW para submissão a redes de classificação é uma proposta de trabalhos futuros em realização que tem potencial para oferecer bons resultados.

Por se tratar de uma pesquisa estritamente acadêmica, uma outra proposta de trabalho futuro é a efetiva implantação em ambiente de produção do TJDFT. Para isso, uma maior amostra da base deve ser testada e contato com usuário final realizado para validação do modelo. Além disso, um planejamento para manutenção evolutiva seria importante, pois o ambiente muda ao longo do tempo e modelos devem ser constantemente adaptados para acompanhar essa evolução, garantindo a manutenção da qualidade dos resultados oferecidos.

Concluindo, a pesquisa provou as questões a que se propunha estudar. A indexação automática é uma fascinante área que se insere num campo prolífico de resultados necessários para a evolução da disponibilização de conhecimento. O estudo de PLN em línguas com morfologia e sintaxe reconhecidamente difíceis, como português ou francês, é um desafio que se impõe para pesquisadores da área de IA. Inúmeras aplicações, entretanto, terão benefícios com a evolução desses resultados. A indexação automática, bem como diversas outras áreas de pesquisa em inteligência artificial, não têm o objetivo de substituir o trabalho humano, mas sim garantir níveis mínimos de atendimento a demandas de trabalho não estritamente intelectual. Permite-se, destarte, que os esforços sejam efetivamente engendrados nos problemas que exijam cognição humana, melhorando, por fim, a qualidade de vida das pessoas. O controle exercido por máquinas, longe de escravizar os seres humanos, vai libertar homens e mulheres para realização de tarefas que somente eles podem fazer.

REFERÊNCIAS

ABRAHÃO, P. R. C. **Modelagem e implementação de um léxico semântico para o português**. Dissertação de Mestrado, Instituto de Informática, Pontifica Universidade Católica do Rio Grande do Sul, 1997.

AIRES, R. V. X. **Implementação, adaptação, combinação e avaliação de etiquetadores para o português do Brasil**. Dissertação de Mestrado, Instituto de Ciências Matemáticas de São Carlos, Universidade de São Paulo, 2000.

AIRES, R. V. X. **Uso de marcadores estilísticos para a busca na Web em português**. Tese de Doutorado, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2005.

ANDERSON, J. D.; PÉREZ-CARBALLO, J. The nature of indexing: how humans and machines analyse messages and texts for retrieval. Part II: machine indexing, and the allocation of human versus machine effort. **Information Processing and Management**, v. 37, p. 255 – 277, 2001.

BASTOS, S. B. **Análise comparativa entre indexação automática e manual da literatura Brasileira de ciência da informação**. Dissertação de Mestrado, Departamento de Ciência da Informação e Documentação, Universidade de Brasília, Brasília, 1984.

BRACHMAN, R. J.; LEVESQUE, H. J. **Knowledge representation and reasoning**. San Francisco: Elsevier, 2004.

BRAGA JÚNIOR, M. S. **Proposta de modelo RBC para a recuperação inteligente de jurisprudência na Justiça Federal**. Dissertação de Mestrado, Programa de Pós-Graduação em Engenharia de Produção, Universidade Federal de Santa Catarina, Florianópolis, 2001.

GIL, A. C. **Como elaborar projetos de pesquisa**. 4 ed. São Paulo: Atlas, 2002.

HLAVA, M. M. Automatic indexing: a matter of degree. **Bulletin of the American Society for Information Science and Technology**, v. 29, n. 1, p. 12 – 15, out. / nov. 2002.

KROVETZ, R. Viewing morphology as an inference process. In: proceedings of **Conference on Research and Development in Information Retrieval**, p 191 – 202, jun / jul 1993.

KURAMOTO, H. **Proposition d'un système de recherche d'information assistée par ordinateur**. Tese de Doutorado, L'Université Lumière, Lyon, 1999.

LANCASTER, F. W. **Indexação e resumos**: teoria e prática. 2 ed. Brasília: Briquet de Lemos, 2004.

MARCONI, M. A.; LAKATOS, E. M. **Metodologia Científica**. 4 ed. São Paulo: Atlas, 2004.

MEDEIROS, M. B. B. **Tratamento automático de ambigüidades na recuperação da informação**. Tese de Doutorado, Departamento de Ciência da Informação e Documentação, Universidade de Brasília, Brasília, 1999.

MIRABETE, J. F. **Direito penal interpretado**. 2 ed. São Paulo: Atlas, 2001.

MOENS, M. F. **Automatic indexing and abstracting of document texts**. Massachusetts: Kluwer Academic Publishers, 2000.

MOENS, M. F.; DE BUSSER, R. First steps in building a model for retrieval of court decisions. **International Journal of Human – Computer Studies**, n. 57, p. 429 – 446, 2002.

MOENS, M. F.; UYTTENDAELE, C.;DUMORTIER, J. Information extraction from legal texts: the potential of discourse analysis. **Internation Journal of Human – Computer Studies**, n. 51, p. 1155 – 1171, 1999.

PARDO, T. A. S. **DMSumm**: Um gerador automático de sumários. Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de São Carlos, São Carlos, 2002.

ROBREDO, J. **Documentação de hoje e de amanhã**: uma abordagem revisitada e contemporânea da Ciência da Informação e de suas aplicações biblioteconômicas, documentárias, arquivísticas e museológicas. 4 ed. Brasília: Reproart, 2005.

ROWLEY, J. E. **Abstracting and Indexing**. 2 ed. Londres: Clive Bingley, 1988.

SANTOS, E. F. **Manual de direito processual civil**. V. 1. 9 ed. São Paulo: Saraiva, 2002.

SCHLICHTING, A. M. **Teoria geral do processo**: Livro 1. 2 ed. Florianópolis: Momento Atual, 2004.

ANEXO A – CÓDIGO FONTE CLASSE MENU PRINCIPAL

```
package janelas;

import java.awt.Container;
import java.awt.Dimension;
import java.awt.Image;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JDesktopPane;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;

import logica.Tesauro;

/**
 * Classe que descreve a janela principal do sistema cuja funcionalidade é a
abertura do Menu de Opções
 * @author Júnior
 */
public class MenuPrincipal extends JFrame {

	/**
	 * Identificador serial da janela
	 */
	private static final long serialVersionUID = 1;

	/**
	 * Painel contenedor das subjanelas do sistema
	 */
	private JDesktopPane objDesktop = null;

	/**
	 * Contenedor dos objetos da janela
	 */
	private Container objContainer = null;

	/**
	 * Objeto da classe Tesauro que contém todo o esquema de representação do
conhecimento
	 */
	private Tesauro objTesauro = null;

	/**
	 * Método principal de execução inicial do sistema
	 * @param args Argumentos para inicialização do sistema
	 */
	public static void main(String[] args) {
		MenuPrincipal janela = new MenuPrincipal();
		janela.setVisible(true);
	}

	/**
	 * Método construtor da classe
	 */
	public MenuPrincipal() {
		objTesauro = new Tesauro();
		try {
			objTesauro.montarTesauro();
```

```
			}
			catch (Exception e) {
				JOptionPane.showMessageDialog(null, e.toString(), "Erro na
Leitura de Arquivos !", JOptionPane.ERROR_MESSAGE);
			}

			setTitle("Sistema de Indexação Automática de Acórdãos");
			setSize(800, 600);
			setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

			Toolkit kit = Toolkit.getDefaultToolkit();
			Dimension screenSize = kit.getScreenSize();
			int screenHeight = screenSize.height;
			int screenWidth = screenSize.width;

			setLocation((screenWidth - 800) / 2, (screenHeight - 600) / 2);

			Image img = kit.getImage("Brasao.gif");
			setIconImage(img);

			objContainer = getContentPane();
			objDesktop = new JDesktopPane();
			objContainer.add(objDesktop);

			JMenuBar barraDeMenu = new JMenuBar();

			JMenu menu = new JMenu("Ferramentas");
			menu.setMnemonic('F');

			JMenuItem itemConstrutor = new  JMenuItem("Construtor de Corpus");
			itemConstrutor.setMnemonic('C');
			itemConstrutor.addActionListener(new ActionListener() {
				public void actionPerformed(ActionEvent evt) {
					ConstrutorDeCorpus janela = new
ConstrutorDeCorpus("Construtor de Corpus", true, true, true, true);
					objDesktop.add(janela);
					janela.setVisible(true);
				}
			});
			menu.add(itemConstrutor);

			JMenuItem itemExtrator = new JMenuItem("Extrator de Sintagmas");
			itemExtrator.setMnemonic('E');
			itemExtrator.addActionListener(new ActionListener() {
				public void actionPerformed(ActionEvent evt) {
					ExtratorDeSintagmas janela = new
ExtratorDeSintagmas("Extrator de Sintagmas", true, true, true, true, objTesauro);
					objDesktop.add(janela);
					janela.setVisible(true);
				}
			});
			menu.add(itemExtrator);

			menu.addSeparator();

			JMenuItem itemSair = new JMenuItem("Sair");
			itemSair.setMnemonic('S');
			itemSair.addActionListener(new ActionListener() {
				public void actionPerformed(ActionEvent evt) {
					System.exit(0);
				}
			});

			menu.add(itemSair);
			barraDeMenu.add(menu);
			setJMenuBar(barraDeMenu);
		}
}
```

ANEXO B – CÓDIGO FONTE CLASSE CONSTRUTOR DE CORPUS

```
package janelas;

import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;

import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JInternalFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;

import logica.ExampleFileFilter;
import logica.UtilString;

/**
 * Classe que descreve a janela cuja funcionalidade é a construção do Corpus de
Língua Portuguesa
 * @author Júnior
 */
public class ConstrutorDeCorpus extends JInternalFrame {

	/**
	 * Identificador serial da janela
	 */
	private static final long serialVersionUID = 2;

	/**
	 * Painel contenedor dos objetos da janela
	 */
	private JPanel objPainel = null;

	/**
	 * Área de texto para a exibição do texto
	 */
	private JTextArea txtTexto = null;

	/**
	 * Scroll para a área de texto
	 */
	private JScrollPane jspTexto = null;

	/**
	 * Mensagem para o acrônimo de adjetivo
	 */
	private JLabel lblAdjetivo = null;

	/**
```

```
 * Mensagem para o acrônimo de advérbio
 */
private JLabel lblAdverbio = null;

/**
 * Mensagem para o acrônimo de artigo
 */
private JLabel lblArtigo = null;

/**
 * Mensagem para o acrônimo de conjunção
 */
private JLabel lblConjuncao = null;

/**
 * Mensagem para o acrônimo de interjeição
 */
private JLabel lblInterjeicao = null;

/**
 * Mensagem para o acrônimo de numeral
 */
private JLabel lblNumeral = null;

/**
 * Mensagem para o acrônimo de pontuação
 */
private JLabel lblPontuacao = null;

/**
 * Mensagem para o acrônimo de preposição
 */
private JLabel lblPreposicao = null;

/**
 * Mensagem para o acrônimo de pronome
 */
private JLabel lblPronome = null;

/**
 * Mensagem para o acrônimo de substantivo
 */
private JLabel lblSubstantivo = null;

/**
 * Mensagem para o acrônimo de verbo
 */
private JLabel lblVerbo = null;

/**
 * Caixa de texto para recepção do código da classe morfológica
 */
private JTextField txtCodigo = null;

/**
 * Mensagem para a palavra selecionada para análise
 */
private JLabel lblPalavra = null;

/**
 * Variável que contém todo o texto do acórdão
 */
private String acordao = null;

/**
 * Variável contadora
 */
private int contador = 0;
```

```
	/**
	 * Vetor com todas as palavras do texto que já foram analisadas com suas
respectivas classes morfológicas
	 */
	private ArrayList<String> vetorPalavra = null;

	/**
	 * Vetor com todas as classes gramaticais
	 */
	private ArrayList<String> vetorClasseGramatical = new ArrayList<String>();

	/**
	 * Nome do arquivo de saída
	 */
	private String arquivoDeSaida = "";

	/**
	 * Método construtor da classe
	 * @param titulo Título da Janela
	 * @param tamanho Permissão para alterar o tamanho da janela
	 * @param fechamento Permissão para fechar a janela
	 * @param maximizacao Permissão para maximizar a janela
	 * @param iconizacao Permissão para redefinição do ícone da janela
	 */
	public ConstrutorDeCorpus(String titulo, boolean tamanho, boolean fechamento,
boolean maximizacao, boolean iconizacao) {
		super(titulo, tamanho, fechamento, maximizacao, iconizacao);

		setSize(790, 540);
		setTitle(titulo);
		setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);

		Container objContainer = getContentPane();
		objPainel = new JPanel();
		objPainel.setLayout(null);

		txtTexto = new JTextArea();
		txtTexto.setLineWrap(true);
		jspTexto = new JScrollPane(txtTexto);
		jspTexto.setBounds(20, 20, 600, 300);
		objPainel.add(jspTexto);

		lblAdjetivo = new JLabel();
		lblAdjetivo.setBounds(640, 20, 160, 15);
		lblAdjetivo.setText("0 - ADJ Adjetivo");
		objPainel.add(lblAdjetivo);

		lblAdverbio = new JLabel();
		lblAdverbio.setBounds(640, 40, 160, 15);
		lblAdverbio.setText("1 - ADV Advérbio");
		objPainel.add(lblAdverbio);

		lblArtigo = new JLabel();
		lblArtigo.setBounds(640, 60, 160, 15);
		lblArtigo.setText("2 - ART Artigo");
		objPainel.add(lblArtigo);

		lblConjuncao = new JLabel();
		lblConjuncao.setBounds(640, 80, 160, 15);
		lblConjuncao.setText("3 - CON Conjuncao");
		objPainel.add(lblConjuncao);

		lblInterjeicao = new JLabel();
		lblInterjeicao.setBounds(640, 100, 160, 15);
		lblInterjeicao.setText("4 - INT Interjeição");
		objPainel.add(lblInterjeicao);
```

```
		lblNumeral = new JLabel();
		lblNumeral.setBounds(640, 120, 160, 15);
		lblNumeral.setText("5 - NUM Numeral");
		objPainel.add(lblNumeral);

		lblPontuacao = new JLabel();
		lblPontuacao.setBounds(640, 140, 160, 15);
		lblPontuacao.setText("6 - PON Pontuação");
		objPainel.add(lblPontuacao);

		lblPreposicao = new JLabel();
		lblPreposicao.setBounds(640, 160, 160, 15);
		lblPreposicao.setText("7 - PRE Preposição");
		objPainel.add(lblPreposicao);

		lblPronome = new JLabel();
		lblPronome.setBounds(640, 180, 160, 15);
		lblPronome.setText("8 - PRO Pronome");
		objPainel.add(lblPronome);

		lblSubstantivo= new JLabel();
		lblSubstantivo.setBounds(640, 200, 160, 15);
		lblSubstantivo.setText("9 - SUB Substantivo");
		objPainel.add(lblSubstantivo);

		lblVerbo = new JLabel();
		lblVerbo.setBounds(640, 220, 160, 15);
		lblVerbo.setText("V - VER Verbo");
		objPainel.add(lblVerbo);

		txtCodigo = new JTextField();
		txtCodigo.setBounds(20, 340, 13, 18);
		objPainel.add(txtCodigo);
		txtCodigo.addKeyListener(new KeyListener() {
			public void keyPressed(KeyEvent evt) {
			}
			public void keyReleased(KeyEvent evt) {
			}
			public void keyTyped(KeyEvent evt) {
				if ("0123456789vV".indexOf(evt.getKeyChar()) > -1) {
					vetorPalavra.add(lblPalavra.getText() + "
".substring(0, 29 - lblPalavra.getText().length()) +
decodificaClasseGramatical(evt.getKeyChar()));
					lblPalavra.setText(pegaProximaPalavra());
				}
			}
		});

		lblPalavra = new JLabel();
		lblPalavra.setBounds(40, 340, 560, 15);
		objPainel.add(lblPalavra);

		JButton btnProcurar = new JButton("Procurar");
		btnProcurar.setBounds(75, 400, 100, 30);
		btnProcurar.setMnemonic('P');
		objPainel.add(btnProcurar);
		btnProcurar.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent evt) {
				acaoProcurar();
			}
		});

		JButton btnGerar = new JButton("Gerar");
		btnGerar.setBounds(185, 400, 100, 30);
		btnGerar.setMnemonic('G');
		objPainel.add(btnGerar);
		btnGerar.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent evt) {
```

```
                acaoGerar();
            }
        });

        JButton btnFinalizar = new JButton("Finalizar");
        btnFinalizar.setBounds(295, 400, 100, 30);
        btnFinalizar.setMnemonic('F');
        objPainel.add(btnFinalizar);
        btnFinalizar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                acaoFinalizar();
            }
        });

        JButton btnConsolidar = new JButton("Consolidar");
        btnConsolidar.setBounds(405, 400, 100, 30);
        btnConsolidar.setMnemonic('C');
        objPainel.add(btnConsolidar);
        btnConsolidar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                acaoConsolidar();
            }
        });

        JButton btnLimpar = new JButton("Limpar");
        btnLimpar.setBounds(515, 400, 100, 30);
        btnLimpar.setMnemonic('L');
        objPainel.add(btnLimpar);
        btnLimpar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                acaoLimpar();
            }
        });

        JButton btnSair = new JButton("Sair");
        btnSair.setBounds(625, 400, 100, 30);
        btnSair.setMnemonic('S');
        objPainel.add(btnSair);
        btnSair.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                dispose();
            }
        });

        objContainer.add(objPainel);

        vetorClasseGramatical.add("ADJ Adjetivo");
        vetorClasseGramatical.add("ADV Advérbio");
        vetorClasseGramatical.add("ART Artigo");
        vetorClasseGramatical.add("CON Conjunção");
        vetorClasseGramatical.add("INT Interjeição");
        vetorClasseGramatical.add("NUM Numeral");
        vetorClasseGramatical.add("PON Pontuação");
        vetorClasseGramatical.add("PRE Preposição");
        vetorClasseGramatical.add("PRO Pronome");
        vetorClasseGramatical.add("SUB Substantivo");
        vetorClasseGramatical.add("VER Verbo");
    }

    /**
     * Método que implementa a funcionalidade do botão Procurar,
     * selecionando um arquivo para leitura
     */
    private void acaoProcurar() {
        try {
            JFileChooser escolhedorDeArquivo = new JFileChooser();
            ExampleFileFilter filtro = new ExampleFileFilter();
```

```
                filtro.addExtension("txt");
                filtro.setDescription("Arquivos de Acórdão");
                escolhedorDeArquivo.setFileFilter(filtro);
                escolhedorDeArquivo.setCurrentDirectory(new File("../"));
                if (escolhedorDeArquivo.showOpenDialog(null) ==
JFileChooser.APPROVE_OPTION) {
                    arquivoDeSaida =
escolhedorDeArquivo.getSelectedFile().getPath().substring(0,
escolhedorDeArquivo.getSelectedFile().getPath().length() - 4) + "_Corpus.txt";
                        FileInputStream arquivoEntrada = new
FileInputStream(escolhedorDeArquivo.getSelectedFile().getPath());
                        InputStreamReader leitorEntrada = new
InputStreamReader(arquivoEntrada);
                        BufferedReader leitorBuferizado = new
BufferedReader(leitorEntrada);
                        String leitor = "";

                        txtTexto.setText("");
                        while ((leitor = leitorBuferizado.readLine()) != null) {
                              txtTexto.append(leitor + "\n");
                        }
                        txtTexto.setCaretPosition(0);

                        leitorBuferizado.close();
                        leitorEntrada.close();
                        arquivoEntrada.close();
                }
            }
            catch(Exception e) {
                  JOptionPane.showMessageDialog(null, e.toString(), "Erro na
Leitura de Arquivos !", JOptionPane.ERROR_MESSAGE);
            }
      }

      /**
       * Método que implementa a funcionalidade do botão Gerar,
       * iniciando o processo de classificação morfológica
       */
      private void acaoGerar() {
            int inicio = 0;

            acordao = txtTexto.getText().replace("\n", " ");
            contador = 0;
            vetorPalavra = new ArrayList<String>();

            inicio = acordao.indexOf("RELATORIO");
            if (inicio > -1) {
                  inicio = inicio + 10;
            }
            else {
                  inicio = acordao.indexOf("RELATÓRIO");
                  if (inicio > -1) {
                        inicio = inicio + 10;
                  }
                  else {
                        inicio = acordao.indexOf("R E L A T O R I O");
                        if (inicio > -1) {
                              inicio = inicio + 17;
                        }
                        else {
                              inicio = acordao.indexOf("R E L A T Ó R I O");
                              if (inicio > -1) {
                                    inicio = inicio + 17;
                              }
                              else {
                                    inicio = acordao.indexOf("EXPOSICAO");
                                    if (inicio > -1) {
                                          inicio = inicio + 10;
```

```
                                        }
                                        else {
                                               inicio = acordao.indexOf("EXPOSIÇAO");
                                               if (inicio > -1) {
                                                      inicio = inicio + 10;
                                               }
                                               else {
                                                      inicio =
acordao.indexOf("EXPOSICÃO");
                                                      if (inicio > -1) {
                                                             inicio = inicio + 10;
                                                      }
                                                      else {
                                                             inicio =
acordao.indexOf("EXPOSIÇÃO");
                                                             if (inicio > -1) {
                                                                    inicio = inicio +
10;
                                                             }
                                                             else {
                                                                    inicio =
acordao.indexOf("E X P O S I C A O");
                                                                    if (inicio > -1) {
                                                                           inicio =
inicio + 17;
                                                                    }
                                                                    else {
                                                                           inicio =
acordao.indexOf("E X P O S I Ç A O");
                                                                           if (inicio
> -1) {
       inicio = inicio + 17;
                                                                           }
                                                                           else {
       inicio = acordao.indexOf("E X P O S I C Ã O");
                                                                                  if
(inicio > -1) {
       inicio = inicio + 17;
                                                                                  }
                                                                                  else
{
       inicio = acordao.indexOf("E X P O S I Ç Ã O");
       if (inicio > -1) {
              inicio = inicio + 17;
       }
                                                                                  }
                                                                           }
                                                                    }
                                                             }
                                                      }
                                               }
                                        }
                                 }
                          }
                   }
            }
            if (inicio == -1) {
                   JOptionPane.showMessageDialog(null, "Não foi possível encontrar
a marca de início do Relatório", "Erro no Documento de Acórdão !",
JOptionPane.ERROR_MESSAGE);
            }
```

```
			else {
				acordao = acordao.substring(inicio, acordao.length());
				lblPalavra.setText(pegaProximaPalavra());
				txtCodigo.requestFocus();
			}
		}

		/**
		 * Método que implementa a funcionalidade do botão Finalizar,
		 * concluindo o processo de análise morfológica
		 */
		private void acaoFinalizar() {
			if (vetorPalavra != null && vetorPalavra.size() > 0) {
				try {
					FileOutputStream arquivoSaida = new
FileOutputStream(arquivoDeSaida);
					DataOutputStream escritorSaida = new
DataOutputStream(arquivoSaida);

					for (int i = 0 ; i < vetorPalavra.size() ; i++) {
						escritorSaida.writeBytes(vetorPalavra.get(i) +
"\r\n");
					}

					escritorSaida.close();
					arquivoSaida.close();

					JOptionPane.showMessageDialog(null, "Corpus Gerado com
Sucesso", "Sucesso !", JOptionPane.INFORMATION_MESSAGE);
					acaoLimpar();
				}
				catch(Exception e) {
					JOptionPane.showMessageDialog(null, e.toString(), "Erro
na Gravação de Arquivos !", JOptionPane.ERROR_MESSAGE);
				}
			}
		}

		/**
		 * Método que implementa a funcionalidade do botão Consolidar,
		 * selecionando um conjunto de arquivos para leitura e consolidação do Corpus
		 */
		private void acaoConsolidar() {
			try {
				JFileChooser escolhedorDeArquivo = new JFileChooser();
			    ExampleFileFilter filtro = new ExampleFileFilter();

			    filtro.addExtension("txt");
			    filtro.setDescription("Arquivos de Corpus");
			    escolhedorDeArquivo.setFileFilter(filtro);
			    escolhedorDeArquivo.setCurrentDirectory(new File("../"));
			    escolhedorDeArquivo.setMultiSelectionEnabled(true);
			    if (escolhedorDeArquivo.showOpenDialog(null) ==
JFileChooser.APPROVE_OPTION) {
					FileOutputStream arquivoSaida = new
FileOutputStream("Corpus.txt");
					DataOutputStream escritorSaida = new
DataOutputStream(arquivoSaida);

					for (File arquivoDaSelecao :
escolhedorDeArquivo.getSelectedFiles()) {
					    if
(arquivoDaSelecao.getName().contains("_Corpus.txt")) {
						FileInputStream arquivoEntrada = new
FileInputStream(arquivoDaSelecao.getPath());
						InputStreamReader leitorEntrada = new
InputStreamReader(arquivoEntrada);
```

```
                                        BufferedReader leitorBuferizado = new
BufferedReader(leitorEntrada);
                                        String leitor = "";

                                        while ((leitor =
leitorBuferizado.readLine()) != null) {
                                                escritorSaida.writeBytes(leitor.trim()
+ "\r\n");
                                        }

                                        leitorBuferizado.close();
                                        leitorEntrada.close();
                                        arquivoEntrada.close();
                                }
                        }

                                escritorSaida.close();
                                arquivoSaida.close();

                                Runtime.getRuntime().exec("java -classpath qtag.jar
qtag.ResourceCreator Corpus.txt Portugues");
                                JOptionPane.showMessageDialog(null, "Corpus Consolidado
com Sucesso", "Sucesso !", JOptionPane.INFORMATION_MESSAGE);
                            }
                        }
                        catch(Exception e) {
                                JOptionPane.showMessageDialog(null, e.toString(), "Erro na
Leitura de Arquivos !", JOptionPane.ERROR_MESSAGE);
                        }
        }

        /**
         * Método que implementa a funcionalidade do botão Limpar,
         * limpando os campos da tela e retornando ao estado inicial
         */
        private void acaoLimpar() {
                txtTexto.setText("");
                txtCodigo.setText("");
                lblPalavra.setText("");

                contador = 0;
                vetorPalavra = null;
        }

        /**
         * Método que recupera palavra por palavra do texto para designação da
classificação morfológica
         * @return Próxima palavra do texto
         */
        private String pegaProximaPalavra() {
                String palavra = "";
                boolean flag = true;

                while (flag)
                {
                        contador = contador + 1;
                        palavra = UtilString.getPiece(acordao, " ", contador);
                        palavra = UtilString.removeCaracteresEspeciais(palavra);
                        if (palavra.length() > 0) flag = false;
                }

                return palavra;
        }

        /**
         * Método que decodifica uma classe gramatical a partir do código numérico
estabelecido
         * @param classe Código estabelecido para a classe gramatical
```

```
         * @return Acrônimo que define a classe gramatical
         */
        private String decodificaClasseGramatical(char classe) {
              String retorno = "";

              if (classe == '0')
                    retorno = vetorClasseGramatical.get(0);
              else if (classe == '1')
                    retorno = vetorClasseGramatical.get(1);
              else if (classe == '2')
                    retorno = vetorClasseGramatical.get(2);
              else if (classe == '3')
                    retorno = vetorClasseGramatical.get(3);
              else if (classe == '4')
                    retorno = vetorClasseGramatical.get(4);
              else if (classe == '5')
                    retorno = vetorClasseGramatical.get(5);
              else if (classe == '6')
                    retorno = vetorClasseGramatical.get(6);
              else if (classe == '7')
                    retorno = vetorClasseGramatical.get(7);
              else if (classe == '8')
                    retorno = vetorClasseGramatical.get(8);
              else if (classe == '9')
                    retorno = vetorClasseGramatical.get(9);
              else if (classe == 'v')
                    retorno = vetorClasseGramatical.get(10);
              else if (classe == 'V')
                    retorno = vetorClasseGramatical.get(10);

              return UtilString.getPiece(retorno, " ", 1);
        }
}
```

ANEXO C – CÓDIGO FONTE CLASSE EXTRATOR DE SINTAGMAS

```
package janelas;

import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.StringTokenizer;

import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JInternalFrame;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

import logica.ExampleFileFilter;
import logica.Tesauro;
import logica.UtilString;

/**
 * Classe que descreve a janela cuja funcionalidade é a extração dos sintagmas do
texto que estejam no padrão selecionado
 * @author Júnior
 */
public class ExtratorDeSintagmas extends JInternalFrame{

	/**
	 * Identificador serial da janela
	 */
	private static final long serialVersionUID = 3;

	/**
	 * Separador de sintagmas
	 */
	private static final String SEPARADOR = "------------------------------";

	/**
	 * Objeto da classe Tesauro que contém todo o esquema de representação do
conhecimento
	 */
	private Tesauro objTesauro = null;

	/**
	 * Painel contenedor dos objetos da janela
	 */
	private JPanel objPainel = null;

	/**
	 * Caixa de seleção com as classes gramaticais
```

```
         */
        private JComboBox cboClasse = null;

        /**
         * Lista de seleção com as classes gramaticais selecionadas na ordem da
montagem do sintagma
         */
        private JList lstClasses = null;

        /**
         * Modelo para a lista de seleção
         */
        private DefaultListModel dlmClasses = null;

        /**
         * Caixa de opção para indicação da utilização do tesauro
         */
        private JCheckBox chkTesauro = null;

        /**
         * Caixa de opção para indicação da utilização de termos relacionados
         */
        private JCheckBox chkTermosRelacionados = null;

        /**
         * Área de texto para exibição do resultado
         */
        private JTextArea txtTexto = null;

        /**
         * Scroll para a área de texto
         */
        private JScrollPane jspTexto = null;

        /**
         * Vetor com o resultado da análise do processamento de linguagem natural
         */
        private String[] resultadoAnalise = null;

        /**
         * Vetor com todo o conteúdo do texto
         */
        private String[] entradaAnalise = null;

        /**
        /**
         * Método construtor da classe
         * @param titulo Título da Janela
         * @param tamanho Permissão para alterar o tamanho da janela
         * @param fechamento Permissão para fechar a janela
         * @param maximizacao Permissão para maximizar a janela
         * @param iconizacao Permissão para redefinição do ícone da janela
         * @param objTesauro Objeto da classe Tesauro que contém todo o esquema de
representação do conhecimento
         */
        public ExtratorDeSintagmas(String titulo, boolean tamanho, boolean
fechamento, boolean maximizacao, boolean iconizacao, Tesauro objTesauro) {

                super(titulo, tamanho, fechamento, maximizacao, iconizacao);

                this.objTesauro = objTesauro;

                setSize(790, 540);
                setTitle(titulo);
                setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);

                Container objContainer = getContentPane();
                objPainel = new JPanel();
```

```
objPainel.setLayout(null);

cboClasse =  new JComboBox();
cboClasse.setBounds(20, 45, 150, 20);
cboClasse.addItem("ADJ - Adjetivo");
cboClasse.addItem("ADV - Advérvio");
cboClasse.addItem("ART - Artigo");
cboClasse.addItem("CON - Conjunção");
cboClasse.addItem("INT - Interjeição");
cboClasse.addItem("NUM - Numeral");
cboClasse.addItem("PON - Pontuação");
cboClasse.addItem("PRE - Preposição");
cboClasse.addItem("PRO - Pronome");
cboClasse.addItem("SUB - Substantivo");
cboClasse.addItem("VER - Verbo");
cboClasse.addItem(SEPARADOR);
objPainel.add(cboClasse);

JButton btnInserir = new JButton("=>");
btnInserir.setBounds(200, 30, 50, 20);
objPainel.add(btnInserir);
btnInserir.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent evt) {
		dlmClasses.addElement(cboClasse.getSelectedItem());
	}
});

JButton btnRemover = new JButton("<=");
btnRemover.setBounds(200, 60, 50, 20);
objPainel.add(btnRemover);
btnRemover.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent evt) {
		int indice = lstClasses.getSelectedIndex();

		if (indice > -1) {
			dlmClasses.remove(indice);
		}
	}
});

dlmClasses = new DefaultListModel();
lstClasses = new JList(dlmClasses);
JScrollPane jspClasses = new JScrollPane();
jspClasses.getViewport().setView(lstClasses);
jspClasses.setBounds(280, 15, 300, 90);
objPainel.add(jspClasses);

txtTexto = new JTextArea();
txtTexto.setLineWrap(true);
jspTexto = new JScrollPane(txtTexto);
jspTexto.setBounds(20, 120, 707, 250);
objPainel.add(jspTexto);

chkTesauro = new JCheckBox("Utilizar Tesauro");
chkTesauro.setBounds(610, 30, 200, 20);
objPainel.add(chkTesauro);

chkTermosRelacionados = new JCheckBox("Termos Relacionados");
chkTermosRelacionados.setBounds(610, 60, 200, 20);
objPainel.add(chkTermosRelacionados);

JButton btnProcurar = new JButton("Procurar");
btnProcurar.setBounds(75, 400, 100, 30);
btnProcurar.setMnemonic('P');
objPainel.add(btnProcurar);
btnProcurar.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent evt) {
		acaoProcurar();
```

```
			}
		});

		JButton btnAnalise = new JButton("Análise");
		btnAnalise.setBounds(185, 400, 100, 30);
		btnAnalise.setMnemonic('A');
		objPainel.add(btnAnalise);
		btnAnalise.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent evt) {
				acaoAnalise();
			}
		});

		JButton btnIndexar = new JButton("Indexar");
		btnIndexar.setBounds(295, 400, 100, 30);
		btnIndexar.setMnemonic('I');
		objPainel.add(btnIndexar);
		btnIndexar.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent evt) {
				acaoIndexar();
			}
		});

		JButton btnComparar = new JButton("Comparar");
		btnComparar.setBounds(405, 400, 100, 30);
		btnComparar.setMnemonic('C');
		objPainel.add(btnComparar);
		btnComparar.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent evt) {
				acaoComparar();
			}
		});

		JButton btnLimpar = new JButton("Limpar");
		btnLimpar.setBounds(515, 400, 100, 30);
		btnLimpar.setMnemonic('L');
		objPainel.add(btnLimpar);
		btnLimpar.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent evt) {
				acaoLimpar();
			}
		});

		JButton btnSair = new JButton("Sair");
		btnSair.setBounds(625, 400, 100, 30);
		btnSair.setMnemonic('S');
		objPainel.add(btnSair);
		btnSair.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent evt) {
				dispose();
			}
		});

		objContainer.add(objPainel);
	}

	/**
	 * Método que implementa a funcionalidade do botão Procurar,
	 * selecionando um arquivo para leitura e realizando o processamento
	 * de linguagem natural do texto
	 */
	private void acaoProcurar() {
		try {
			ArrayList<String> vetor = new ArrayList<String>();
			String leitor = "";

			JFileChooser escolhedorDeArquivo = new JFileChooser();
		    ExampleFileFilter filtro = new ExampleFileFilter();
```

```
                filtro.addExtension("txt");
                filtro.setDescription("Arquivos de Acórdão");
                escolhedorDeArquivo.setFileFilter(filtro);
                escolhedorDeArquivo.setCurrentDirectory(new File("../"));
                if (escolhedorDeArquivo.showOpenDialog(null) ==
JFileChooser.APPROVE_OPTION) {
                        FileInputStream arquivoEntrada = new
FileInputStream(escolhedorDeArquivo.getSelectedFile().getPath());
                        InputStreamReader leitorEntrada = new
InputStreamReader(arquivoEntrada);
                        BufferedReader leitorBuferizado = new
BufferedReader(leitorEntrada);

                        while ((leitor = leitorBuferizado.readLine()) != null) {
                              StringTokenizer objTokenizador = new
StringTokenizer(leitor);
                              while (objTokenizador.hasMoreTokens()) {
                                    vetor.add(objTokenizador.nextToken());
                              }
                        }

                        entradaAnalise = new String[vetor.size()];
                        for (int i = 0 ; i < vetor.size() ; i++)
                              entradaAnalise[i] = vetor.get(i).toString();

                        qtag.Tagger objTagger = new qtag.Tagger("Portugues");
                        resultadoAnalise = objTagger.tag(entradaAnalise);

                        leitorBuferizado.close();
                        leitorEntrada.close();
                        arquivoEntrada.close();
                }
            }
            catch(Exception e) {
                  JOptionPane.showMessageDialog(null, e.toString(), "Erro na
Leitura de Arquivos !", JOptionPane.ERROR_MESSAGE);
            }
      }

      /**
       * Método que implementa a funcionalidade do botão Análise,
       * procurando no texto todas as ocorrências dos sintagmas escolhidos
       */
      private void acaoAnalise() {
            ArrayList<ArrayList<String>> matriz = new
ArrayList<ArrayList<String>>();
            ArrayList<String> linha = new ArrayList<String>();

            if (resultadoAnalise == null) {
                  JOptionPane.showMessageDialog(null, "Para realizar a análise é
necessário selecionar um arquivo", "Erro na Análise !", JOptionPane.ERROR_MESSAGE);
                  return;
            }

            for (int i = 0 ; i < dlmClasses.getSize() ; i++) {
                  if (dlmClasses.getElementAt(i).equals(SEPARADOR)) {
                        matriz.add(linha);
                        linha = new ArrayList<String>();
                  }
                  else {

      linha.add(UtilString.getPiece(dlmClasses.getElementAt(i).toString(), " ",
1));
                  }
            }
            if (linha.size() > 0) {
                  matriz.add(linha);
```

```
            }

            txtTexto.setText(procuraPadroes(matriz, entradaAnalise,
resultadoAnalise, "\r\n", chkTesauro.isSelected(), objTesauro,
chkTermosRelacionados.isSelected()));
      }

      /**
       * Método que implementa a funcionalidade do botão Indexar,
       * selecionando um conjunto de arquivos para indexação automática
       */
      private void acaoIndexar() {
            try {
                  String leitor = "";

                  JFileChooser escolhedorDeArquivo = new JFileChooser();
                ExampleFileFilter filtro = new ExampleFileFilter();

                filtro.addExtension("txt");
                filtro.setDescription("Arquivos de Acórdãos");
                escolhedorDeArquivo.setFileFilter(filtro);
                escolhedorDeArquivo.setCurrentDirectory(new File("../"));
                escolhedorDeArquivo.setMultiSelectionEnabled(true);
                if (escolhedorDeArquivo.showOpenDialog(null) ==
JFileChooser.APPROVE_OPTION) {
                        for (File arquivoDaSelecao :
escolhedorDeArquivo.getSelectedFiles()) {
                              if
(UtilString.isNumerico(UtilString.getPiece(arquivoDaSelecao.getName(),".",1))) {
                                    ArrayList<String> vetor = new
ArrayList<String>();
                              String arquivoDeSaida =
arquivoDaSelecao.getPath().substring(0, arquivoDaSelecao.getPath().length() - 4) +
"_Indexação_Automatica.txt";

                              FileInputStream arquivoEntrada = new
FileInputStream(arquivoDaSelecao.getPath());
                                    InputStreamReader leitorEntrada = new
InputStreamReader(arquivoEntrada);
                                    BufferedReader leitorBuferizado = new
BufferedReader(leitorEntrada);

                                    while ((leitor =
leitorBuferizado.readLine()) != null) {
                                          StringTokenizer objTokenizador = new
StringTokenizer(leitor);
                                          while (objTokenizador.hasMoreTokens())
{

      vetor.add(objTokenizador.nextToken());
                                          }
                                    }

                                    entradaAnalise = new String[vetor.size()];
                                    for (int i = 0 ; i < vetor.size() ; i++)
                                          entradaAnalise[i] =
vetor.get(i).toString();


                                    qtag.Tagger objTagger = new
qtag.Tagger("Portugues");
                                    resultadoAnalise =
objTagger.tag(entradaAnalise);


                                    leitorBuferizado.close();
                                    leitorEntrada.close();
                                    arquivoEntrada.close();
```

```
                                        ArrayList<ArrayList<String>> matriz = new
ArrayList<ArrayList<String>>();
                                        ArrayList<String> linha = new
ArrayList<String>();

                                        linha.add("SUB");
                                        linha.add("ADJ");
                                        matriz.add(linha);
                                        linha = new ArrayList<String>();
                                        linha.add("SUB");
                                        linha.add("PRE");
                                        linha.add("SUB");
                                        matriz.add(linha);
                                        linha = new ArrayList<String>();
                                        linha.add("SUB");
                                        linha.add("ADJ");
                                        linha.add("PRE");
                                        linha.add("SUB");
                                        matriz.add(linha);
                                        linha = new ArrayList<String>();
                                        linha.add("SUB");
                                        linha.add("PRE");
                                        linha.add("SUB");
                                        linha.add("ADJ");
                                        matriz.add(linha);

                                        FileOutputStream arquivoSaida = new
FileOutputStream(arquivoDeSaida);
                                        DataOutputStream escritorSaida = new
DataOutputStream(arquivoSaida);


      escritorSaida.writeBytes(procuraPadroes(matriz, entradaAnalise,
resultadoAnalise, ", ", true, objTesauro, false));

                                        escritorSaida.close();
                                        arquivoSaida.close();

                                        entradaAnalise = null;
                                        resultadoAnalise = null;
                                    }
                                }
                                JOptionPane.showMessageDialog(null, "Indexação Automática
Concluída com Sucesso", "Sucesso !", JOptionPane.INFORMATION_MESSAGE);
                            }
                        }
                        catch(Exception e) {
                                JOptionPane.showMessageDialog(null, e.toString(), "Erro na
Leitura de Arquivos !", JOptionPane.ERROR_MESSAGE);
                        }
      }

      /**
       * Método que implementa a funcionalidade do botão Comparar,
       * selecionando um conjunto de arquivos para comparação dos índices
       */
      private void acaoComparar() {
            try {
                  String leitor = "";
                  String acordao = "";
                  ArrayList<String> acordaos = new ArrayList<String>();
                  Hashtable<String, ArrayList<String>> tabelaIndexacaoHumana = new
Hashtable<String, ArrayList<String>>();
                  Hashtable<String, ArrayList<String>> tabelaIndexacaoAutomatica =
new Hashtable<String, ArrayList<String>>();
                  int[] frequencia = new int[100];

                  JFileChooser escolhedorDeArquivo = new JFileChooser();
```

```
                ExampleFileFilter filtro = new ExampleFileFilter();

                filtro.addExtension("txt");
                filtro.setDescription("Arquivos de Índices");
                escolhedorDeArquivo.setFileFilter(filtro);
                escolhedorDeArquivo.setCurrentDirectory(new File("../"));
                escolhedorDeArquivo.setMultiSelectionEnabled(true);
                if (escolhedorDeArquivo.showOpenDialog(null) ==
JFileChooser.APPROVE_OPTION) {
                        for (File arquivoDaSelecao :
escolhedorDeArquivo.getSelectedFiles()) {
                                if
(arquivoDaSelecao.getName().contains("_Indexação_")) {
                                        acordao =
UtilString.getPiece(arquivoDaSelecao.getName(),"_",1);
                                        ArrayList<String> vetor = new
ArrayList<String>();

                                FileInputStream arquivoEntrada = new
FileInputStream(arquivoDaSelecao.getPath());
                                        InputStreamReader leitorEntrada = new
InputStreamReader(arquivoEntrada);
                                        BufferedReader leitorBuferizado = new
BufferedReader(leitorEntrada);

                                        while ((leitor =
leitorBuferizado.readLine()) != null) {
                                                StringTokenizer objTokenizador = new
StringTokenizer(leitor, ",");
                                                while (objTokenizador.hasMoreTokens())
{

       vetor.add(UtilString.removeCaracteresEspeciais(objTokenizador.nextToken().tri
m()));
                                                }
                                        }

                                        leitorBuferizado.close();
                                        leitorEntrada.close();
                                        arquivoEntrada.close();

                                        if (! acordaos.contains(acordao))
                                                acordaos.add(acordao);
                                        if
(arquivoDaSelecao.getName().contains("_Indexação_Humana.txt")) {
                                                tabelaIndexacaoHumana.put(acordao,
vetor);
                                        }
                                        else {
                                                tabelaIndexacaoAutomatica.put(acordao,
vetor);
                                        }
                                }
                        }

                        if (acordaos.size() == 0) {
                                JOptionPane.showMessageDialog(null, "Não Foram
Selecionados Arquivos de Índices", "Erro na Seleção de Arquivos !",
JOptionPane.ERROR_MESSAGE);
                        }
                        else {
                                txtTexto.setText("");
                                txtTexto.append("Acórdão\tÍndices Humanos\tÍndices
Automáticos\tCongruências\tCoeficiente de Avaliação\r\n");
                                for (int i = 0 ; i < acordaos.size() ; i++) {
                                        int indexacaoHumana =
tabelaIndexacaoHumana.get(acordaos.get(i)).size();
```

```
                                        int indexacaoAutomatica =
tabelaIndexacaoAutomatica.get(acordaos.get(i)).size();
                                        int congruencia = 0;

                                        for (int j = 0 ; j < indexacaoHumana ; j++)
{
                                              String temp =
tabelaIndexacaoHumana.get(acordaos.get(i)).get(j);
                                              for (int k = 0 ; k <
indexacaoAutomatica ; k++) {
                                                    if
(temp.equals(tabelaIndexacaoAutomatica.get(acordaos.get(i)).get(k))) {
                                                          congruencia++;
                                                    }
                                              }
                                        }

                                        float coeficienteDeAvaliacao = 100 *
congruencia/(indexacaoHumana + indexacaoAutomatica - congruencia);
                                        txtTexto.append(acordaos.get(i) + "\t" +
indexacaoHumana + "\t\t" + indexacaoAutomatica + "\t\t" +
                                         congruencia + "\t" +
Float.toString(coeficienteDeAvaliacao) + "\r\n");


      frequencia[Integer.parseInt(UtilString.getPiece(Float.toString(coeficienteDeA
valiacao),".",1))]++;
                                  }
                                  txtTexto.append("\r\n\r\n\r\nCoeficiente de
Avaliacao\tFreqüência\r\n");
                                  for (int i = 0 ; i < 100 ; i++) {
                                        if (frequencia[i] > 0) {
                                              txtTexto.append(i + "\t\t" +
frequencia[i] + "\r\n");
                                        }
                                  }
                                  JOptionPane.showMessageDialog(null, "Comparação de
Índices Concluída com Sucesso", "Sucesso !", JOptionPane.INFORMATION_MESSAGE);
                            }
                      }
                }
                catch(Exception e) {
                      JOptionPane.showMessageDialog(null, e.toString(), "Erro na
Leitura de Arquivos !", JOptionPane.ERROR_MESSAGE);
                }
      }

      /**
       * Método que implementa a funcionalidade do botão Limpar,
       * limpando os campos da tela e retornando ao estado inicial
       */
      private void acaoLimpar() {
            cboClasse.setSelectedIndex(0);
            dlmClasses.removeAllElements();
            chkTesauro.setSelected(false);
            chkTermosRelacionados.setSelected(false);
            txtTexto.setText("");
            resultadoAnalise = null;
      }

      /**
       * Método que procura padrões de estruturas sintagmáticas no texto
       * @param matriz Estrutura bidimensional que contém os padrões que se deseja
procurar
       * @param entradaAnalise Vetor cada unidade lexical do texto
       * @param resultadoAnalise Vetor com a análise morfológica de cada unidade
lexical do texto
       * @param separador Separador que será utilizado no retorno do método
```

```
        * @param utilizarTesauro Booleano que indica se o tesauro deve ser utilizado
para permitir os termos
        * @param objTesauro Objeto da classe Tesauro que contém todo o esquema de
representação do conhecimento
        * @param utilizarTermoRelacionado Booleano que indica se os termos
relacionados no tesauro serão utilizados
        * @return String com os padrões encontrados separados pelo separador
        */
       private String procuraPadroes(ArrayList<ArrayList<String>> matriz, String[]
entradaAnalise, String[] resultadoAnalise, String separador, boolean
utilizarTesauro, Tesauro objTesauro, boolean utilizarTermoRelacionado) {
              boolean flag = false;
              String temp = "";
              String saida = "";
              String use = "";
              String palavra = "";
              String palavraSingular = "";
              ArrayList<String> linha = new ArrayList<String>();

              for (int i = 0 ; i < matriz.size() ; i++) {
                     linha = matriz.get(i);
                     for (int j = 0 ; j < resultadoAnalise.length - linha.size() ;
j++) {
                           flag = true;
                           for (int k = 0 ; k < linha.size() ; k++) {
                                  if (! resultadoAnalise[j + k].equals(linha.get(k)))
{
                                         flag = false;
                                  }
                           }

                           if (flag) {
                                  temp = "";
                                  for (int k = 0 ; k < linha.size() ; k++) {
                                         palavra =
UtilString.removeCaracteresEspeciais(entradaAnalise[j + k].trim().toUpperCase());
                                         palavraSingular = "";

                                         if (palavra.length() > 0) {
                                                if (palavra.charAt(palavra.length()-1)
== 'S') {
                                                       palavraSingular =
palavra.substring(0,palavra.length() - 1);
                                                }
                                         }

                                         if (utilizarTesauro &&
objTesauro.isTermo(palavra)) {
                                                use =
objTesauro.getTermo(palavra).getUse();
                                                if (use.length() > 0) {
                                                       palavra = use;
                                                }
                                         }

                                         if (palavraSingular.length() > 0 &&
utilizarTesauro && objTesauro.isTermo(palavraSingular)) {
                                                use =
objTesauro.getTermo(palavraSingular).getUse();
                                                if (use.length() > 0) {
                                                       palavra = use;
                                                }
                                         }

                                         temp = temp + palavra + " ";
                                  }
                                  temp = temp.trim();
```

```
                                    if ((! utilizarTesauro) ||
(objTesauro.isTermo(temp))) {
                                          if (utilizarTesauro) {
                                                use =
objTesauro.getTermo(temp).getUse();
                                                if (use.length() > 0) {
                                                      temp = use;
                                                }
                                          }
                                          if (! saida.contains(temp)) {
                                                saida = saida + temp + separador;
                                                if (utilizarTesauro &&
utilizarTermoRelacionado) {
                                                      for (String termo :
objTesauro.getTermo(temp).getTr()) {
                                                            saida = saida + "     "
+ termo + separador;
                                                      }
                                                }
                                          }
                                    }
                              }
                        }
                  }
            if (saida.length() > 0)
                  saida = saida.substring(0, saida.length() - separador.length());

            return saida;
      }
}
```

ANEXO D – CÓDIGO FONTE CLASSE TESAURO

```
package logica;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Hashtable;

/**
 * Classe que modela um esquema de representação do conhecimento baseado em Tesauro
 * @author Júnior
 */
public class Tesauro {

	/**
	 * Tabela hash que contém todos os termos do tesauro. Esta tabela é indexada
pela unidade lexical que define o termo
	 */
	private Hashtable<String, Termo> tabelaDeTermos = null;

	/**
	 * Método construtor da classe
	 */
	public Tesauro() {
		tabelaDeTermos = new Hashtable<String, Termo>();
	}

	/**
	 * Método que testa se um termo faz parte do tesauro
	 * @param termo Unidade lexical que será testada
	 * @return Verdadeiro ou falso para existência ou inexistência do termo no
tesauro respectivamente
	 */
	public boolean isTermo(String termo) {
		return tabelaDeTermos.containsKey(termo);
	}

	/**
	 * Método que retorna um termo do Tesauro
	 * @param termo Unidade lexical que será utilizado como chave para pesquisa
no tesauro
	 * @return Termo do tesauro com todas as suas relações
	 */
	public Termo getTermo(String termo) {
		return tabelaDeTermos.get(termo);
	}

	/**
	 * Método que monta o tesauro a partir do arquivo texto formatado com as
unidades lexicais e suas relações
	 * @throws Exception Exceções de leitura de arquivo
	 */
	public void montarTesauro() throws Exception {
		int estado = 77;
		Termo objTermo = null;
		ArrayList<String> objTemp = null;

		FileInputStream arquivoEntrada = new FileInputStream("Tesauro.txt");
		InputStreamReader leitorEntrada = new
InputStreamReader(arquivoEntrada);
		BufferedReader leitorBuferizado = new BufferedReader(leitorEntrada);
		String leitor = "";
```

```
			while ((leitor = leitorBuferizado.readLine()) != null) {
				leitor = leitor.trim();
				if (leitor.charAt(0) != ';') {
					if (estado > 0) {
						if (estado != 77) {
							for (int i = 0 ; i < objTemp.size() ; i++) {
								if (objTemp.get(i).charAt(0) != ';') {
									objTermo = new Termo();

	objTermo.setTermo(objTemp.get(i));
								}
								else if
(UtilString.getPiece(objTemp.get(i), ";", 1).trim().equals("CAT")) {

	objTermo.setCat(UtilString.getPiece(objTemp.get(i), ";", 2).trim());
								}
								else if
(UtilString.getPiece(objTemp.get(i), ";", 1).trim().equals("NOTA")) {

	objTermo.setNota(UtilString.getPiece(objTemp.get(i), ";", 2).trim());
								}
								else if
(UtilString.getPiece(objTemp.get(i), ";", 1).trim().equals("USE")) {

	objTermo.setUse(UtilString.getPiece(objTemp.get(i), ";", 2).trim());
								}
								else if
(UtilString.getPiece(objTemp.get(i), ";", 1).trim().equals("UP")) {

	objTermo.setUp(UtilString.getPiece(objTemp.get(i), ";", 2).trim());
								}
								else if
(UtilString.getPiece(objTemp.get(i), ";", 1).trim().equals("TR")) {

	objTermo.adicionaTr(UtilString.getPiece(objTemp.get(i), ";", 2).trim());
								}
								else if
(UtilString.getPiece(objTemp.get(i), ";", 1).trim().substring(0,2).equals("TG")) {

	objTermo.adicionaTg(UtilString.getPiece(objTemp.get(i), ";", 2).trim());
								}
								else if
(UtilString.getPiece(objTemp.get(i), ";", 1).trim().substring(0,2).equals("TE")) {

	objTermo.adicionaTe(UtilString.getPiece(objTemp.get(i), ";", 2).trim());
								}
							}
							tabelaDeTermos.put(objTermo.getTermo(),
objTermo);
						}
						objTemp = new ArrayList<String>();
						objTemp.add(leitor);
						estado = 0;
					}
					else {
						objTemp.set(objTemp.size() - 1,
objTemp.get(objTemp.size() - 1) + " " + leitor);
					}
				}
				else {
					estado = 1;
					if (leitor.charAt(1) == ';') {
						estado = 2;
						objTemp.set(objTemp.size() - 1,
objTemp.get(objTemp.size() - 1) + " " + leitor.substring(2));
					}
					if (estado == 1) {
```

```
                                objTemp.add(leitor);
                            }
                        }
                    }

                    leitorBuferizado.close();
                    leitorEntrada.close();
                    arquivoEntrada.close();
                }
            }
```

ANEXO E – CÓDIGO FONTE CLASSE TERMO

```
package logica;

import java.util.ArrayList;

/**
 * Classe que modela um Termo de um Tesauro
 * @author Júnior
 */
public class Termo {

	/**
	 * Unidade lexical do termo
	 */
	private String termo = "";

	/**
	 * Categoria do termo
	 */
	private String cat = "";

	/**
	 * Notas de observação sobre o termo
	 */
	private String nota = "";

	/**
	 * Termo que deverá necessariamente substituir o termo atual
	 */
	private String use = "";

	/**
	 * Termo superior
	 */
	private String up = "";

	/**
	 * Conjunto de termos relativos
	 */
	private ArrayList<String> tr = null;

	/**
	 * Conjunto de termos gerais
	 */
	private ArrayList<String> tg = null;

	/**
	 * Conjunto de termos específicos
	 */
	private ArrayList<String> te = null;

	/**
	 * Método construtor da classe
	 */
	public Termo() {
		tr = new ArrayList<String>();
		tg = new ArrayList<String>();
		te = new ArrayList<String>();
	}

	/**
	 * Método que retorna o termo
	 * @return Unidade lexical propriamente dito
```

```
	 */
	public String getTermo() {
		return termo;
	}

	/**
	 * Método que define o termo
	 * @param termo Unidade lexical propriamente dito
	 */
	public void setTermo(String termo) {
		this.termo = termo;
	}

	/**
	 * Método que retorna a categoria do termo
	 * @return Categoria do termo
	 */
	public String getCat() {
		return cat;
	}

	/**
	 * Método que define a categoria do termo
	 * @param cat Categoria do termo
	 */
	public void setCat(String cat) {
		this.cat = cat;
	}

	/**
	 * Método que retorna notas de observação sobre o termo
	 * @return Observações do termo
	 */
	public String getNota() {
		return nota;
	}

	/**
	 * Método que define notas de observação sobre o termo
	 * @param nota Observações do termo
	 */
	public void setNota(String nota) {
		this.nota = nota;
	}

	/**
	 * Método que retorna o termo que deverá necessariamente substituir o termo
atual
	 * @return O novo termo
	 */
	public String getUse() {
		return use;
	}

	/**
	 * Método que define o termo que deverá necessariamente substituir o termo
atual
	 * @param use O novo termo
	 */
	public void setUse(String use) {
		this.use = use;
	}

	/**
	 * Método que recupera o termo superior
	 * @return Termo superior
	 */
	public String getUp() {
```

```
		return up;
	}

	/**
	 * Método que define o termo superior
	 * @param up Termo superior
	 */
	public void setUp(String up) {
		this.up = up;
	}

	/**
	 * Método que retorna o conjunto de termos específicos do termo atual
	 * @return Conjunto de termos específicos
	 */
	public ArrayList<String> getTe() {
		return te;
	}

	/**
	 * Método que adiciona um termo ao conjunto de termos específicos do termo
atual
	 * @param te Termo específico
	 */
	public void adicionaTe(String te) {
		this.te.add(te);
	}

	/**
	 * Método que retorna o conjunto de termos gerais do termo atual
	 * @return Conjunto de termos gerais
	 */
	public ArrayList<String> getTg() {
		return tg;
	}

	/**
	 * Método que adiciona um termo ao conjunto de termos gerais do termo atual
	 * @param tg Termo geral
	 */
	public void adicionaTg(String tg) {
		this.tg.add(tg);
	}

	/**
	 * Método que retorna o conjunto de termos relativos do termo atual
	 * @return Conjunto de termos relativos
	 */
	public ArrayList<String> getTr() {
		return tr;
	}

	/**
	 * Método que adiciona um termo ao conjunto de termos relativos do termo
atual
	 * @param tr Termo relativo
	 */
	public void adicionaTr(String tr) {
		this.tr.add(tr);
	}
}
```

ANEXO F – CÓDIGO FONTE CLASSE UTIL STRING

```
package logica;

import java.util.Collection;
import java.util.Iterator;
import java.util.StringTokenizer;

/**
 * Classe que implementa diversas funcionalidades úteis para tratamento de Strings
 * @author Tribunal de Justiça do Distrito Federal e Territórios
 */
public class UtilString {

  /**
   * Conjunto de vogais acentuadas
   */
  private static final String VOGAIS_ACENTUADAS[] = {"á", "à", "â", "ã", "é",
      "ê", "í", "î", "ó", "ô", "õ", "ú", "û", "Á", "À", "Â", "Ã", "É", "Ê",
      "Í", "Î", "Ó", "Ô", "Õ", "Ú", "Û"};

  /**
   * Conjunto de vogais não acentuadas
   */
  private static final String VOGAIS_NAO_ACENTUADAS[] = {"a", "a", "a", "a",
      "e", "e", "i", "i", "o", "o", "o", "u", "u", "A", "A", "A", "A", "E",
      "E", "I", "I", "O", "O", "O", "U", "U"};

  /**
   * Método que retorna uma cópia à direita da string inicial com determinado
tamanho
   * @param texto String inicial
   * @param numeroDeCaracteres Tamanho do retorno
   * @return Substring à direita
   */
  public static String right(String texto, int numeroDeCaracteres) {
    if (numeroDeCaracteres > texto.length()) {
      return texto.toString();
    }
    return texto.substring(texto.length() - numeroDeCaracteres);
  }

  /**
   * Método que retorna uma cópia à esquerda da string inicial com determinado
tamanho
   * @param texto String inicial
   * @param numeroDeCaracteres Tamanho do retorno
   * @return Substring à esquerda
   */
  public static String left(String texto, int numeroDeCaracteres) {
    if (numeroDeCaracteres > texto.length()) {
      return texto.toString();
    }
    return texto.substring(0, numeroDeCaracteres);
  }

  /**
   * Método que retorna um pedaço da string inicial identificado pela ordem e
marcado por um separador
   * @param texto String inicial
   * @param separador Caracter que será marcado como separador de pedaços
   * @param pos Ordem do pedaço
   * @return Pedaço selecionado
   */
```

```
  public static String getPiece(String texto, String separador, int pos) {
    StringTokenizer token = new StringTokenizer(texto, separador);
    String pedaco = "";
    for (; pos > 0 && token.hasMoreTokens(); pos--) {
      pedaco = token.nextToken();
    }
    if (pos == 0) {
      return pedaco;
    }
    return "";
  }

  /**
   * Método que remove determinado caracter de uma string
   * @param caracter Caracter que será removido
   * @param texto String inicial
   * @return Substring com caracteres removidos
   */
  public static String trocaCaracter(String caracter, String texto) {
    StringBuilder retorno = new StringBuilder();
    for (int i = 0; i < texto.length(); i++) {
      if (texto.charAt(i) != caracter.charAt(0)) {
        retorno.append(texto.charAt(i));
      }
    }
    return retorno.toString();
  }

  /**
   * Método que transforma uma string em um vetor de substrings
   * @param linhasConcatenadas String inicial
   * @param separador Caracter que será marcado como separador de strings
   * @return Vetor de substrings
   */
  public static String[] stringToArray(String linhasConcatenadas,
      String separador) {
    StringTokenizer to = new StringTokenizer(linhasConcatenadas, separador);
    int posicoes = to.countTokens();
    String[] linhas = new String[posicoes];
    for (int i = 0; i < posicoes; i++) {
      linhas[i] = to.nextToken();
    }
    return linhas;
  }

  /**
   * Método que avalia se uma string é composta exclusivamente por números
   * @param numero String inicial
   * @return Verdadeiro ou falso para indicar se trata-se de um número ou não
respectivamente
   */
  public static boolean isNumerico(String numero) {
    for (int i = 0; i < numero.length(); i++) {
      if (Character.isDigit(numero.charAt(i)) == false) {
        return false;
      }
    }
    return true;
  }

  /**
   * Método que avalia se uma string é composta exclusivamente de caracteres
alfanuméricos
   * @param numero String inicial
   * @return Verdadeiro ou falso para indicar se trata-se de uma string
alfanumérica ou não respectivamente
   */
  public static boolean isAlfaNumerico(String numero) {
```

```
    for (int i = 0; i < numero.length(); i++) {
      if (Character.isDigit(numero.charAt(i)) == false
          && Character.isLetter(numero.charAt(i)) == false) {
        return false;
      }
    }
    return true;
  }

  /**
   * Método que avalia se um caracter é uma letra
   * @param letra Caracter a ser avaliado
   * @return Verdadeiro ou falso para indicar se o caracter é uma letra ou não
respectivamente
   */
  public static boolean isLetra(char letra) {
    return Character.isLetter(letra);
  }


  /**
   * Método que cria um vetor de strings a partir de uma coleção de valores
   * @param colecaoDeValores Coleção inicial
   * @return Vetor de strings
   */public static String[] criaArrayDeString(Collection colecaoDeValores) {
    String[] array;
    if (colecaoDeValores.size() == 0) {
      array = new String[1];
    } else {
      array = new String[colecaoDeValores.size()];
    }
    Iterator iter = colecaoDeValores.iterator();
    for (int i = 0; i < colecaoDeValores.size(); i++) {
      array[i] = (String) iter.next();
    }
    return array;
  }

   /**
    * Método que transforma uma string em um vetor de substrings
    * @param texto String inicial
    * @param separador Caracter que será marcado como separador de strings
    * @return Vetor de substrings
    */
  public static String[] criaArrayDeString(String texto, String separador) {
    StringTokenizer token = new StringTokenizer(texto, separador);
    String[] array = new String[token.countTokens()];
    for (int i = 0; i < array.length; i++) {
      array[i] = token.nextToken();
    }
    return array;
  }

  /**
   * Método que remove os acentos de um texto
   * @param palavra String inicial
   * @return String alterada sem os acentos
   */
  public static String retiraAcentos(String palavra) {
    String palavraAlterada = palavra;
    for (int i = 0; i < VOGAIS_ACENTUADAS.length; i++) {
      palavraAlterada = palavraAlterada.replaceAll(VOGAIS_ACENTUADAS[i],
          VOGAIS_NAO_ACENTUADAS[i]);
    }
    return palavraAlterada;
  }

  /**
```

```
 * Método que transforma uma coleção em uma string separada
 * @param col Coleção inicial
 * @param separador Caracter que será marcado como separador de strings
 * @return String separada
 */
public static String criaStringComPieces(Collection col, String separador) {
  return criaStringComPieces(col.iterator(), separador);
}

/**
 * Método que transforma um iterador de uma coleção em uma string separada
 * @param iter Iterador inicial
 * @param separador Caracter que será marcado como separador de strings
 * @return String separada
 */
public static String criaStringComPieces(Iterator iter, String separador) {
  String html = "";
  while (iter.hasNext()) {
    if (html.length() > 0) {
      html = html + separador;
    }
    html = (String) iter.next();
  }
  return html;
}

/**
 * Método que retorna o nome de uma classe sem o pacote
 * @param classe Classe que será analisada
 * @return Nome da classe sem o pacote
 */
public static String extraiNomeDaClasse(Class classe) {
  return extraiNomeDaClasse(classe.getName());
}

/**
 * Método que retorna o nome de uma classe sem o pacote
 * @param nomeDaClasse Nome da classe
 * @return Nome da classe sem o pacote
 */
public static String extraiNomeDaClasse(String nomeDaClasse) {
  int posPonto = nomeDaClasse.lastIndexOf(".");
  if (posPonto >= 0) {
    return nomeDaClasse.substring(posPonto + 1);
  }
  return nomeDaClasse;
}

/**
 * Método que retorna o nome do pacote de uma classe
 * @param nomeDaClasse Nome da classe
 * @return Nome do pacote
 */
public static String extraiNomeDoPacote(String nomeDaClasse) {
  int posPonto = nomeDaClasse.lastIndexOf(".");
  if (posPonto >= 0) {
    return nomeDaClasse.substring(0, posPonto);
  }
  return nomeDaClasse;
}

/**
 * Método que retorna o nome do pacote de uma classe
 * @param classe Classe que será analisada
 * @return Nome do pacote
 */
public static String extraiNomeDoPacote(Class classe) {
  return extraiNomeDoPacote(classe.getName());
```

```
  }

  /**
   * Método que avalia se dois textos são iguais independente de maiúsculas e
minúsculas
   * @param palavra1 Primeiro texto
   * @param palavra2 Segundo texto
   * @return Verdadeiro ou falso para indicar se os textos são iguais ou não
respectivamente
   */
  public static boolean equalsCanonico(String palavra1, String palavra2) {
    return palavra1.trim().equalsIgnoreCase(palavra2.trim());
  }

  /**
   * Método que retorna uma palavra com a inicial em maiúscula
   * @param palavra String inicial
   * @return String com inicial em maiúscula
   */
  public static String capitula(String palavra) {
    return Character.toUpperCase(palavra.charAt(0)) + palavra.substring(1);
  }

  /**
   * Método que retorna todas as palavras de uma frase com a inicial em maiúscula
   * @param frase String inicial
   * @return String com todas as iniciais das palavras em maiúsculas
   */
  public static String capitulaFrase(String frase) {
    StringBuilder texto = new StringBuilder();
    StringTokenizer token = new StringTokenizer(frase.toLowerCase(), " ");
    while (token.hasMoreTokens()) {
      texto.append(capitula(token.nextToken()));
      if (token.hasMoreTokens()) {
        texto.append(" ");
      }
    }
    return texto.toString();
  }

  /**
   * Método que transforma uma coleção em uma string separada por vírgulas
   * @param lista Coleção inicial
   * @return String separada
   */
  public static String criaStringComOsValoresDaColecao(Collection lista) {
    return criaStringComOsValoresDaColecao(lista, ",");
  }

  /**
   * Método que transforma uma coleção em uma string separada
   * @param lista Coleção inicial
   * @param separador Caracter que será marcado como separador de strings
   * @return String separada
   */
  public static String criaStringComOsValoresDaColecao(Collection lista,
      String separador) {
    Iterator ite = lista.iterator();
    if (!ite.hasNext()) {
      return "";
    }
    StringBuilder retorno = new StringBuilder();
    while (ite.hasNext()) {
      retorno.append(ite.next() + separador);
    }
    return retorno.substring(0, retorno.length() - 1);
  }
```

```
    /**
     * Método que remove determinados caracteres especiais de um texto
     * @param palavra String inicial
     * @return Substring com caracteres removidos
     */
    public static String removeCaracteresEspeciais(String palavra) {
        palavra = palavra.replace("*", "");
        palavra = palavra.replace("@", "");
        palavra = palavra.replace("!", "");
        palavra = palavra.replace("?", "");
        palavra = palavra.replace(".", "");
        palavra = palavra.replace(",", "");
        palavra = palavra.replace("'", "");
        palavra = palavra.replace("´", "");
        palavra = palavra.replace("`", "");
        palavra = palavra.replace(";", "");
        palavra = palavra.replace(":", "");
        palavra = palavra.replace("=", "");
        palavra = palavra.replace("_", "");
        palavra = palavra.replace("\"", "");
        palavra = palavra.replace("(", "");
        palavra = palavra.replace(")", "");
        palavra = palavra.replace("[", "");
        palavra = palavra.replace("]", "");
        palavra = palavra.replace("#", "");
        palavra = palavra.replace("~", "");
        palavra = palavra.replace("^", "");
        palavra = palavra.replace("¨", "");
        palavra = palavra.replace("+", "");
        palavra = palavra.replace("<", "");
        palavra = palavra.replace(">", "");
        palavra = palavra.replace("|", "");
        palavra = palavra.replace("\\", "");
        palavra = palavra.replace("\r", "");
        palavra = palavra.replace("\n", "");
        palavra = palavra.replace("\t", "");

        return palavra;
    }
}
```

ANEXO G – CÓDIGO FONTE CLASSE EXAMPLE FILE FILTER

```
package logica;

/*
 * @(#)ExampleFileFilter.java    1.16 04/07/26
 *
 * Copyright (c) 2004 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * -Redistribution of source code must retain the above copyright notice, this
 *  list of conditions and the following disclaimer.
 *
 * -Redistribution in binary form must reproduce the above copyright notice,
 *  this list of conditions and the following disclaimer in the documentation
 *  and/or other materials provided with the distribution.
 *
 * Neither the name of Sun Microsystems, Inc. or the names of contributors may
 * be used to endorse or promote products derived from this software without
 * specific prior written permission.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING
 * ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
 * OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MIDROSYSTEMS, INC. ("SUN")
 * AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE
 * AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS
 * DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST
 * REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL,
 * INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY
 * OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS SOFTWARE,
 * EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 *
 * You acknowledge that this software is not designed, licensed or intended
 * for use in the design, construction, operation or maintenance of any
 * nuclear facility.
 */

/*
 * @(#)ExampleFileFilter.java    1.16 04/07/26
 */


import java.io.File;
import java.util.Enumeration;
import java.util.Hashtable;

import javax.swing.filechooser.FileFilter;

/**
 * A convenience implementation of FileFilter that filters out
 * all files except for those type extensions that it knows about.
 *
 * Extensions are of the type ".foo", which is typically found on
 * Windows and Unix boxes, but not on Macinthosh. Case is ignored.
 *
 * Example - create a new filter that filerts out all files
 * but gif and jpg image files:
 *
 *     JFileChooser chooser = new JFileChooser();
 *     ExampleFileFilter filter = new ExampleFileFilter(
 *                     new String{"gif", "jpg"}, "JPEG & GIF Images")
```

```
 *     chooser.addChoosableFileFilter(filter);
 *     chooser.showOpenDialog(this);
 *
 * @version 1.16 07/26/04
 * @author Jeff Dinkins
 */
public class ExampleFileFilter extends FileFilter {

    private Hashtable<String, ExampleFileFilter> filters = null;
    private String description = null;
    private String fullDescription = null;
    private boolean useExtensionsInDescription = true;

    /**
     * Creates a file filter. If no filters are added, then all
     * files are accepted.
     *
     * @see #addExtension
     */
    public ExampleFileFilter() {
       this.filters = new Hashtable<String, ExampleFileFilter>();
    }

    /**
     * Creates a file filter that accepts files with the given extension.
     * Example: new ExampleFileFilter("jpg");
     *
     * @see #addExtension
     */
    public ExampleFileFilter(String extension) {
       this(extension,null);
    }

    /**
     * Creates a file filter that accepts the given file type.
     * Example: new ExampleFileFilter("jpg", "JPEG Image Images");
     *
     * Note that the "." before the extension is not needed. If
     * provided, it will be ignored.
     *
     * @see #addExtension
     */
    public ExampleFileFilter(String extension, String description) {
       this();
       if(extension!=null) addExtension(extension);
       if(description!=null) setDescription(description);
    }

    /**
     * Creates a file filter from the given string array.
     * Example: new ExampleFileFilter(String {"gif", "jpg"});
     *
     * Note that the "." before the extension is not needed adn
     * will be ignored.
     *
     * @see #addExtension
     */
    public ExampleFileFilter(String[] filters) {
       this(filters, null);
    }

    /**
     * Creates a file filter from the given string array and description.
     * Example: new ExampleFileFilter(String {"gif", "jpg"}, "Gif and JPG Images");
     *
     * Note that the "." before the extension is not needed and will be ignored.
     *
     * @see #addExtension
```

```
 */
public ExampleFileFilter(String[] filters, String description) {
   this();
   for (int i = 0; i < filters.length; i++) {
       // add filters one by one
       addExtension(filters[i]);
   }
   if(description!=null) setDescription(description);
}

/**
 * Return true if this file should be shown in the directory pane,
 * false if it shouldn't.
 *
 * Files that begin with "." are ignored.
 *
 * @see #getExtension
 * @see FileFilter#accepts
 */
public boolean accept(File f) {
   if(f != null) {
       if(f.isDirectory()) {
          return true;
       }
       String extension = getExtension(f);
       if(extension != null && filters.get(getExtension(f)) != null) {
          return true;
       };
   }
   return false;
}

/**
 * Return the extension portion of the file's name .
 *
 * @see #getExtension
 * @see FileFilter#accept
 */
 public String getExtension(File f) {
   if(f != null) {
       String filename = f.getName();
       int i = filename.lastIndexOf('.');
       if(i>0 && i<filename.length()-1) {
          return filename.substring(i+1).toLowerCase();
       };
   }
   return null;
}

/**
 * Adds a filetype "dot" extension to filter against.
 *
 * For example: the following code will create a filter that filters
 * out all files except those that end in ".jpg" and ".tif":
 *
 *   ExampleFileFilter filter = new ExampleFileFilter();
 *   filter.addExtension("jpg");
 *   filter.addExtension("tif");
 *
 * Note that the "." before the extension is not needed and will be ignored.
 */
public void addExtension(String extension) {
   if(filters == null) {
       filters = new Hashtable<String, ExampleFileFilter>(5);
   }
   filters.put(extension.toLowerCase(), this);
   fullDescription = null;
}
```

```
/**
 * Returns the human readable description of this filter. For
 * example: "JPEG and GIF Image Files (*.jpg, *.gif)"
 *
 * @see setDescription
 * @see setExtensionListInDescription
 * @see isExtensionListInDescription
 * @see FileFilter#getDescription
 */
public String getDescription() {
   if(fullDescription == null) {
       if(description == null || isExtensionListInDescription()) {
          fullDescription = description==null ? "(" : description + " (";
          // build the description from the extension list
          Enumeration extensions = filters.keys();
          if(extensions != null) {
              fullDescription += "." + (String) extensions.nextElement();
              while (extensions.hasMoreElements()) {
                 fullDescription += ", ." + (String) extensions.nextElement();
              }
          }
          fullDescription += ")";
       } else {
          fullDescription = description;
       }
   }
   return fullDescription;
}

/**
 * Sets the human readable description of this filter. For
 * example: filter.setDescription("Gif and JPG Images");
 *
 * @see setDescription
 * @see setExtensionListInDescription
 * @see isExtensionListInDescription
 */
public void setDescription(String description) {
   this.description = description;
   fullDescription = null;
}

/**
 * Determines whether the extension list (.jpg, .gif, etc) should
 * show up in the human readable description.
 *
 * Only relevent if a description was provided in the constructor
 * or using setDescription();
 *
 * @see getDescription
 * @see setDescription
 * @see isExtensionListInDescription
 */
public void setExtensionListInDescription(boolean b) {
   useExtensionsInDescription = b;
   fullDescription = null;
}

/**
 * Returns whether the extension list (.jpg, .gif, etc) should
 * show up in the human readable description.
 *
 * Only relevent if a description was provided in the constructor
 * or using setDescription();
 *
 * @see getDescription
```

```
     * @see setDescription
     * @see setExtensionListInDescription
     */
    public boolean isExtensionListInDescription() {
      return useExtensionsInDescription;
    }
}
```