Correção automática de palavras em textos

Ademir de Alvarenga Oliveira[1], Vilar Fiuza da Camara Neto[1]

[1]Departmento de Ciência da Computação – Universidade Federal de Minas Gerais
Belo Horizonte – Minas Gerais – Brasil
{ademirao,neto}@dcc.ufmg.br

Abstract. *Atualmente, os computadores se tornaram equipamentos extremamente populares para a edição de texto. Aproveitando o poder de processamento desses equipamentos, nos últimos anos foram desenvolvidas várias ferramentas para o auxílio à redação, no sentido de detectar diversos erros de edição e sugerir correções adequadas. Neste artigo, apresentaremos o problema da verificação ortográfica e as principais técnicas utilizadas para a sugestão de alternativas em caso de detecção de erros, bem como alguns exemplos de sistemas de correção disponíveis atualmente.*

1 Introdução

Durante a produção e edição de textos, erros de grafia e sintáticos são cometidos com muita freqüência [6]. Erros ortográficos podem ser detectados pela comparação de cada palavra do texto com os registros de um banco de palavras corretas (um dicionário, por exemplo), técnica que é utilizada pela maioria dos sistemas de verificação ortográfica [5].

No entanto, segundo Pacheco [6], "verificar se uma palavra está presente ou não em um vocabulário não é suficiente para garantir que um texto foi editado corretamente. (...) Alguns erros de ortografia só podem ser verificados após a análise da estrutura da frase.". As dificuldades inerentes ao tratamento destes tipos de erros dependem principalmente das regras gramaticais da linguagem analisada. Em linguagens formais, como as de programação, esse conjunto de regras é rígido, o que permite o desenvolvimento de um *parser* capaz de analisar suas sentenças. Entretanto, as linguagens naturais geralmente apresentam flexibilidades e regras tão complexas que ainda não foi possível desenvolver algoritmos 100% eficazes para a detecção de erros ou para assegurar a corretude da sentença analisada [13].

> "Muitas técnicas de correção se baseiam em palavras isoladas, sem levar em consideração qualquer informação sobre o contexto em que ela aparece. Estas técnicas não são capazes de detectar porções significantes de erros, incluindo erros tipográficos, fonéticos, cognitivos e gramaticais[1], que resultam em outras palavras válidas." [11]

Apesar disso, uma quantidade considerável de erros pode ser encontrada apenas pela verificação das palavras [18]. Este processo não é tão simples quanto parece, pois as linguagens naturais apresentam uma quantidade muito grande de palavras, o que gera a necessidade de criar meios eficientes de armazenamento e busca [16].

O objetivo deste artigo é apresentar as técnicas mais comuns de verificação de *erros ortográficos* em palavras isoladas, bem como alguns aspectos sobre aconselhamento de palavras e análise sintática. Na seção 2 apresentaremos técnicas de representação do vocabulário, verificação e aconselhamento de palavras, bem como uma breve apresentação dos principais verificadores ortográficos existentes. Na seção 3 discutiremos aspectos relacionados com a análise sintática e, finalmente, na seção 4 apresentaremos as conclusões deste trabalho.

2 Verificação ortográfica

Segundo Almeida *et al.* [7], "um verificador ortográfico é uma ferramenta capaz de verficar se palavras estão presentes ou não em um banco de dados de palavras de determinada língua". Antes de apresentarmos as principais técnicas para *verificação ortográfica*, é necessário fazermos uma distinção clara entre as tarefas de detecção e correção

[1]Como por exemplo em "um conserto de Mozart".

de erros. Existem técnicas eficientes para detectar se uma determinada palavra aparece em um dicionário ou conjunto de palavras válidas [11]. Um problema nesse processo é que o número de palavras utilizadas por um verificador ortográfico é muito grande (de dezenas a centenas de milhares de palavras) [10].

Por outro lado, corrigir uma palavra grafada incorretamente é uma tarefa muito mais complicada [11]. O ideal seria modelar um analisador ortográfico que corrigisse cada palavra do texto sem a interação do usuário. Mas isso é um objetivo muito difícil de ser alcançado, que requer um elevado grau de entendimento da linguagem natural por parte do programa [18]. Por causa disso, os atuais verificadores ortográficos apenas oferecem recursos para aconselhamento de palavras, deixando a decisão sobre a correção do texto nas mãos do usuário.

> "O desafio real para um sistema de correção ortográfica é o que fazer quando uma palavra não se encontra no dicionário. Muitos sistemas assumem que uma palavra não encontrada é um erro tipográfico, e então sugerem uma série de candidatos ordenados por alguma noção de proximidade (da palavra grafada incorretamente)." [5]

É necessária uma técnica eficiente de detecção dos possíveis candidatos para substituir a palavra em questão que leve em consideração as peculiaridades da língua [11].

Nesta seção apresentaremos os três pontos principais de um verificador ortográfico: a representação interna do vocabulário, o método de verificação e o aconselhamento de palavras.

2.1 Representação do vocabulário e métodos de verificação

> "Proporcionar uma forma compacta de armazenamento do dicionário, o acesso ao mesmo em tempo hábil e a pesquisa por palavras de maneira eficiente é tarefa de extrema importância, visto que corresponde à maior parte da computação de um sistema de verificação ortográfica" [6].

Há várias abordagens possíveis para este problema [11]. Podemos, por exemplo, construir dicionários como uma seqüência de palavras armazenadas em um arquivo de texto e separadas por um caractere especial. A busca pela palavra a ser verificada poderia ser feita por algoritmos de casamento de padrões, pela utilização de índices ou até mesmo por pesquisa binária. Apesar de bastante simples e intuitivas, essas abordagens não são adequadas para grandes vocabulários, sendo aplicáveis apenas para dicionários de tamanho reduzido ou em implementações que as utilizam combinadas com outro método de acesso [18].

Os tópicos seguintes abordam duas das maneiras mais comuns para tratar este problema: o uso da técnica de *stripping* juntamente com tabelas *hash* e a representação utilizando autômatos.

2.1.1 *Stripping* e *hashing*

A técnica mais comum para o acesso rápido a um dicionário é o uso de tabelas *hash* [11].

> "A estrutura de uma tabela *hash* é baseada no fato de que, pela utilização das características de uma palavra, pode ser definido um mapeamento de palavras em um conjunto restrito de números inteiros que podem ser utilizados como índices para localizá-las."[18]

A tabela pode ser definida utilizando-se *hash* parcial ou total. No parcial, o índice das palavras não é único, permitindo que mais de uma delas seja armazenada em uma mesma posição. Neste caso, cada entrada da tabela é uma lista encadeada de palavras que possuem o mesmo código *hash*. Uma palavra é acessada através da determinação de seu código, acesso à posição correspondente na tabela *hash* e pela busca seqüencial da mesma na lista armazenada nesta posição. A figura 1(a) mostra um exemplo de *hash* parcial.

Por outro lado, se for utilizado *hash* total, não haverá conflitos entre as palavras do vocabulário, e conseqüentemente não haverá necessidade de armazená-las na tabela. Neste caso, cada posição conterá apenas um bit indicando se a palavra correspondente a ela pertence ou não ao vocabulário. Esta técnica permite reduzir bastante o tamanho do dicionário. A figura 1(b) mostra um exemplo de representação do vocabulário utilizando *hash* total.

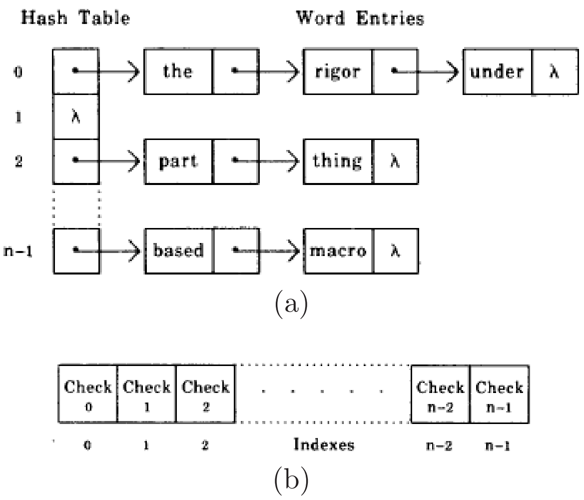

Figura 1: *Hashing* parcial (a) e total (b) [18]

Para diminuir ainda mais o tamanho do dicionário, verificadores ortográficos evitam armazenar todas as variações morfológicas de uma palavra (ex: móvel, imóvel, automóvel, etc.) como entradas diferentes na tabela. Ao invés disso, o radical da mesma é armazenado separadamente dos afixos (sufixos e prefixos). Cada radical possui *flags* indicando seus afixos válidos [6]. A verificação inicia-se pela busca da palavra no vocabulário e, se a busca é mal sucedida, a rotina de *stripping* é invocada. Esta rotina procura pelos afixos que podem ser removidos da palavra. Em seguida, uma nova busca é realizada sobre o radical da mesma. [11] [15]

Segundo Lucchesi & Kowaltowski [12], a utilização de *stripping* com tabelas *hash* reduziu um vocabulário de cerca de 250.000 palavras em inglês para algo em torno de 30.000. Um problema desta técnica é que ela permite que palavras não existentes (como *inmature*, *beginer*, *queueing*, etc., no caso do inglês) sejam aceitas como corretas [18]. Muitos do casos práticos são eliminados pela construção de uma *stop list* que contém alguns erros comuns. Entretanto, isso não resolve completamente o problema, pois esta técnica produz um vocabulário infinito que permite combinações arbitrárias de afixos [12]. Este problema se torna ainda mais sério quando estamos trabalhando com línguas derivadas do latim: no português, por exemplo, cada verbo regular possui cerca de 78 formas, das quais 51 são distintas; um adjetivo possui 4 formas; e substantivos podem ter as mesmas terminações que adjetivos quando existem as formas masculinas e femininas da palavra [6, 18]. Apesar disso, esta técnica é eficiente (pois o acesso à tabela *hash* possui custo constante) e muito utilizada (ver seção 2.3.3).

2.1.2 Representação utilizando autômatos

O uso de autômatos finitos para representar conjuntos de palavras é uma técnica bem estabelecida [12]. Segundo Pacheco [6], esta é uma maneira eficiente para a representação de léxicos, por oferecer uma maneira compacta e eficiente de armazenar e consultar palavras. Utilizando esta representação é possível armazenar mais de 200.000 palavras do português utilizando apenas 124 KB e, além disso, realizar consultas de maneira eficiente (poucas instruções por letra) [9].

A construção de um autômato não-determinístico para o reconhecimento de um vocabulário é muito simples. Considerando-se que a transição de um estado para outro corresponde à ocorrência de uma determinada letra, pode-se, a partir de um único estado inicial, montar vários caminhos independentes, cada um correspondendo a uma palavra. O inconveniente desta representação está primariamente no fato de que é muito ineficiente percorrer um autômato não-determinístico [19]. Além disso, pela grande quantidade de estados equivalentes, é certo que o armazenamento será muito dispendioso.

> "Um diagrama de estados simplificado do tipo árvore pode ser construído facilmente para o reconhecimento de um conjunto de palavras (...) aproveitando-se a mesma seção do diagrama para cada prefixo. Um autômato finito deterministico mais conciso pode ser construído, considerando-se que algumas palavras têm sufixos idênticos (...)."[19]

A construção de um autômato finito determinístico (AFD), onde os prefixos comuns compartilham os mesmos estados, faz com que a representação fique mais eficiente. Um AFD mínimo (conciso), onde não só os prefixos, mas também os sufixos comuns compartilham estados, garante um armazenamento muito mais compacto do vocabulário e um baixo custo na verificação da existência ou não de uma palavra: segundo Lucchesi & Kowaltowski [12], é possível representar o autômato de forma a alcançar uma representação compacta sem sacrificar a velocidade de acesso, que depende apenas do tamanho da palavra procurada. Para verificar a existência de uma palavra p deve-se, a partir do estado inicial do autômato, percorrê-lo seguindo as transições que correspondem a cada letra de p na ordem em que elas aparecem. Esta palavra existe no vocabulário se, e somente se, um estado um estado final é alcançado após a última letra ser consumida.

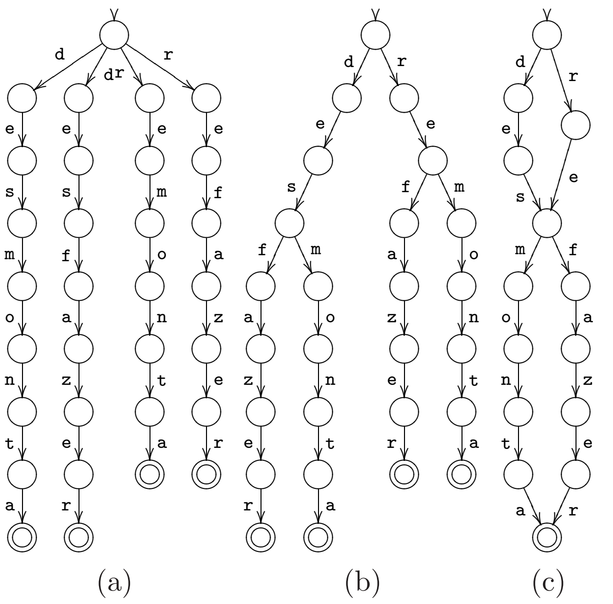

Figura 2: AFND, AFD e AFD mínimo representando as palavras *desmonta*, *remonta*, *refazer* e *desfazer* [6]

A figura 2(a) mostra a representação não determinística das palavras *desmonta*, *remonta*, *desfazer* e *refazer*. Neste, nenhum estado, além do inicial, é compartilhado. Na letra (b) mostramos a representação destas mesmas palavras em um AFD em que os prefixos comuns compartilham os mesmos estados. Finalmente, apresentamos na letra (c) o AFD mínimo. O algoritmo de minimização de autômatos pode ser encontrado em Vieira [19].

Outra vantagem desta representação é que ela permite adicionar classificações sintáticas para as palavras sem grandes impactos no tamanho do autômato: informações gramaticais de uma palavra podem ser armazenadas pela codificação destas informações como uma outra *string* concatenada a palavra original [9].

Esta representação apresenta uma vantagem adicional: é possível obter uma função de *hashing* mínimo e perfeito das palavras do vocabulário, a partir da execução do algoritmo apresentado em Lucchesi & Kowaltowski [12] sobre a versão numerada do autômato (nessa versão, cada estado possui um inteiro indicando o número de palavras que podem ser formadas iniciando-se neste estado).

2.2 Aconselhamento de palavras

O processo de aconselhamento de palavras consiste na escolha de candidatos para substituir uma palavra não encontrada no vocabulário do verificador ortográfico. Geralmente, para selecioná-los os seguintes critérios são utilizados:

- **distância de edição**: Corresponde ao número de operações de inserção, substituição, transposição e de retirada de caracteres necessário para transformar uma cadeia em outra [5]. Neste critério, os candidatos são aqueles que estão a uma menor distância de edição da palavra procurada [5, 6, 12]:
  - **substituição**: assume que o usuário substituiu algum caractere indevidamente na palavra;
  - **exclusão**: assume que o usuário deixou de digitar algum caractere da palavra;
  - **inserção**: assume que o usuário digitou algum caractere a mais na palavra;
  - **transposição de caracteres**: uma palavra obtida pela troca de dois caracteres da palavra incorreta é considerado um bom candidato para sua substituição.
- **distância fonética**: procura candidatos baseando-se na substituição de fonemas por outros de mesmo som como por exemplo: *ch* por *x*, *sc* por *ç* e *qu* por *c* [5, 6].

- ***keyboard distance*** [5]: leva em consideração a disposição dos caracteres no teclado do computador para construir candidatos.

Segundo Pacheco [6], em programas que utilizam a representação do vocabulário por tabelas *hash*, como o *ispell* (ver seção 2.3.3), supondo uma palavra grafada incorretamente por omissão, inserção, substituição ou transposição de caracteres, possíveis candidatos podem ser gerados utilizando um algoritmo de força bruta. Neste, a partir de uma palavra não encontrada no vocabulário, são realizadas transformações de forma a gerar *possíveis* candidatos que são pesquisados no vocabulário. Os encontrados são considerados candidatos reais. Esse algoritmo, apesar de ser força bruta, não é demorado devido a eficiência da pesquisa em tabelas *hash* ($O(1)$). Tomando m o tamanho da palavra incorreta e c o tamanho do alfabeto, são gerados, para cada palavra incorreta, $2m(c+1)+c-1$ possíveis candidatos (conseqüentemente, esta mesma quantidade de consultas na tabela *hash*) [6].

No caso da representação por autômatos, em Lucchesi & Kowaltowski [12] é apresentada uma forma de enumerar os candidatos de uma palavra grafada incorretamente, levando-se em consideração a distância fonética ou mesmo distância de edição:

> "Palavras para as quais desejamos gerar novas alternativas baseada na proximidade fonética são transformadas (...) em uma representação conveniente: *extender* se transforma em $eS_1tender$, *essessão* em eS_2eS_3ao, onde S_1, S_2 e S_3 representam o som usual da letra s em diferentes contextos. Substituições possíveis de S_1 são $\{s, x\}$, para S_2, ss, sc, xc, cc e para S_3, $\{ss, ç, cç\}$. Depois disso, um algoritmo de *backtracking* é utilizado para enumerar todas as possibilidades e checá-las no autômato. (...) Deve ser também destacado que a velocidade em que o autômato pode ser percorrido provavelmente torna praticável outras abordagens, como testes de substituição, transposição, omissão e inserção de letras." [12]

2.3 Sistemas de correção ortográfica

Nesta seção apresentaremos alguns sistemas de correção ortográfica. Para cada um deles, destacaremos a forma de representação do seu vocabulário, seu método de busca e também detalhes relacionados ao seu processo de aconselhamento ortográfico.

2.3.1 Verifica

Desenvolvido no Departamento de Ciência da Computação da Universidade Federal de Minas Gerais, este sistema é capaz de fazer verificações ortográficas em documentos LaTeX. Ele possui um analisador léxico utilizado para tratar as ocorrências de comandos LaTeX no texto e utiliza a representação do vocabulário por autômatos. O sistema possui um vocabulário de cerca de 242 mil palavras e oferece aconselhamento de palavras baseado na distância de edição e distância fonética.

Segundo Pacheco [6], a pesquisa por uma palavra é bastante eficiente devido à forma em que o banco de palavras é armazenado na memória. Além disso, a estrutura utilizada para armazenar o vocabulário permite um bom índice de compactação (76%) e, além disso, garante uma pesquisa eficiente (com custo linear em relação ao tamanho da palavra que está sendo verificada).

O sistema oferece versões em modo texto, com interface gráfica e uma versão disponível para utilização *on-line* no endereço `http://www.dcc.ufmg.br/verifica`[2], onde a versão *stand-alone* também pode ser encontrada para *download*.

2.3.2 Ortho

Desenvolvido pelos professores Tomasz Kowaltowski, Cláudio L. Lucchesi e Jorge Stolfi do Instituto de Computação da Universidade de Campinas, o *Ortho* é um programa que faz verificação ortográfica de textos em português, e opcionalmente sugere correções para eles. Assim como o *Verifica*, este sistema também utiliza a representação por autômatos do seu vocabulário (que é composto de mais de 200.000 palavras).

[2] Apesar da página estar acessível, o sistema, em sua versão web, não estava operacional durante o desenvolvimento deste documento.

"O sistema oferece maior rigor na verificação da validade das palavras, e a geração de sugestões mais 'inteligentes' para as palavras erradas – de maneira mais eficiente" [3]. A verificação e o aconselhamento são puramente ortográficos, sem levar em conta a gramática. Este sistema permite a codificação dos acentos e cedilhas de três maneiras: através da codificação ISO Latin-1 (8 bits), em formato TEX e TT (pares acento-letra).

O sistema pode ser utlizado via *www*, no endereço `http://www.dcc.unicamp.br/ortho`[3], ou via correio eletrônico pelo e-mail `ortho@dcc.unicamp.br`.

2.3.3 Ispell, Jspell e Aspell

O *ispell* é um verificador ortográfico interativo capaz de sugerir e aplicar correções no texto. Armazena o dicionário utilizando as técnicas de *stripping* e tabelas *hash*. O aconselhamento de palavras é feito utilizando o algoritmo de *força bruta*, citado na Seção 2.2, e os candidatos são classificados pela distância de edição [6]. Algumas das vantagens desse sistema são: o suporte multilingual, o trabalho com arquivos no formato LATEX e [nt]roff, baixo uso de memória e suporte a compressão de afixos [1].

O *jspell* é um analisador léxico-morfológico construído com base no *ispell*, estendido no sentido de associar a cada entrada do dicionário um conjunto de atributos como, por exemplo, categoria gramatical, gênero e número [4]. Isso serve para aplicações em linguagem natural que nessecitam de um mecanismo de classificação léxica. É o primeiro passo para a implementação de um analisador sintático [6].

O *aspell* foi desenvolvido com os mesmos objetivos do *ispell*. Permite a checagem de arquivos no formato LATEX e HTML. Entretanto, a sugestão de palavras é feita também pelo critério de proximidade fonética. Essa característica permite a sugestão mais rica e adequada de palavras; não obstante, o algoritmo de casamento fonético o torna adequado apenas para o uso com a língua inglesa [2].

[3]Apesar de sua página estar acessível, o sistema não se encontrou disponível durante o desenvolvimento deste documento.

2.3.4 ReGra

O *ReGra* (*Re*visor *Gra*matical) é um amplo projeto em desenvolvimento por pesquisadores da USP-São Carlos, UFSCar e UNESP-Araraquara. Todo o sistema de verificação ortográfica está baseado na representação através de autômato finito, que inclui informações de categorias gramaticais possíveis para as palavras representadas.

Além de oferecer o aconselhamento ortográfico, recurso comum em todos os sistemas apresentados, ele também inclui um verificador e aconselhador mecânico (que indica problemas de digitação, como excesso de espaços, pontuação mal empregada e correspondência de delimitadores), um verificador gramatical e o acesso direto através do editor de textos *Microsoft Word*[17] . O problema da análise sintática, que é tratado com particular atenção no desenvolvimento desse projeto, será citado na seção seguinte.

3 Análise Sintática

Como já foi citado, alguns erros de ortografia só podem ser detectados após a análise do contexto da frase.

> "A estrutura sintática da frase é um componente importante na maioria dos processos associados com a utilização das línguas naturais, desde a verificação da ortografia (...) até a estruturação de unidades frasais de informação (...)." [6]

O primeiro passo da análise sintática é a identificação das categorias sintáticas de cada palavra. Um *etiquetador gramatical* é um sistema capaz de atribuir, para cada palavra da senteça analisada, uma categoria. Estas categorias costumam incluir: substantivo (nome), verbo, pronome, preposição, advérbio, conjunção, particípio e artigo. É também tarefa do etiquetador detectar casos de ambigüidades, como no caso de *amanhecer* (substantivo/verbo) ou *a* (artigo/preposição), que podem possuir mais de uma categoria, e resolvê-los sem necessariamente passar por outras etapas de processamento. O desafio do processo de etiquetagem reside exatamente em resolver estas ambigüidades [20].

> "Os algoritmos para etiquetagem fundamentam-se em dois modelos mais conhecidos: os baseados em regras e os estocásticos. Os algoritmos baseados em regras, como o nome o diz, fazem uso de bases de regras para identificar a categoria de um certo item lexical. Neste caso, novas regras vão sendo integradas à base à medida que novas situações de uso do item vão sendo encontradas. Os algoritmos baseados em métodos estocásticos costumam resolver as ambigüidades através de um *corpus* de treino, marcado corretamente (muitas vezes através de esforço manual), calculando a probabilidade que uma certa palavra ou item lexical terá de receber uma certa etiqueta em um certo contexto".[20]

Em Pacheco [6] é feito um estudo sobre a categorização sintática, onde foi apresentado um algoritmo capaz de atribuir categorias sintáticas a cada palavra de um vocabulário. Para isso, foram determinadas as categorias de um número finito e pequeno de palavras (artigos, proposições, pronomes, conjunções, parte dos advérbios, numerais, sufixos formadores de nomes, verbos e também as conjugações dos verbos auxiliares *ser*, *estar*, *poder*, *dever*, *ter*, *ir* e *haver*) que são utilizados com muita freqüência em um texto. O problema apresentado pelo algoritmo é que ele gera ambigüidades; ou, melhor dizendo, uma mesma palavra é classificada em mais de uma categoria sintática. Isso ocorre principalmente entre artigos/preposições/pronomes e entre nomes/verbos. O problema da ambigüidade na classificação de palavras também foi verificado no desenvolvimento do *ReGra* [17].

O processo de análise sintática incia-se após a "etiquetagem" das palavras. O analisador sintático irá reconhecer se uma sentença constitui ou não uma frase correta na linguagem proposta. O problema com a fase de análise sintática é que as gramáticas de linguagens naturais são sensíveis ao contexto, e decidir se uma sentença pertence ou não a uma gramática sensível ao contexto possui custo exponencial sobre o tamanho da sentença [20].

Existem vários formalismos para expressar gramáticas (ver Vieira & Lima [20]), mas não existe um consenso sobre o melhor e mais eficiente formalismo para representar uma linguagem natural. A proposta é trabalhar mais ou menos como funcionam os compiladores de linguagens de programação, utilizar um nível intermediário entre gramáticas livres e sensíveis ao contexto. Em McAllester & Givan [14] é apresentado um estudo que faz uma analogia entre o problema da verificação sintática de linguagens naturais e a verificação de tipos em linguagens de programação. Segundo ele, é possível atribuir tipos, ou categorias, a cada estrutura sintática da sentença e, a partir destas informações, verificar se ela é válida.

4 Conclusões

Este artigo apresentou os dois modelos clássicos para a verificação e aconselhamento ortográficos: o uso de autômatos finitos e as técnicas de *stripping* e *hashing*. Embora sejam propostas relativamente antigas, observa-se que são utilizadas nas ferramentas atualmente em uso.

Embora as duas técnicas sejam eficientes para verificar se palavras constam em um determinado dicionário, a correção ortográfica não poderá ser totalmente realizada sem que sejam feitas análises que levam em consideração o contexto lingüístico. Dada a extrema riqueza e flexibilidade de uma língua como a portuguesa, torna-se particularmente complicado criar verificadores automáticos de sintaxe e semântica.

Não obstante, essas limitações não impedem que sejam construídos verificadores capazes de detectar a grande maioria dos erros cometidos durante a edição de textos. A utilidade destes sistemas pode ser verificada pelo estudo desenvolvido por Dickinson *et al.* [8], onde foi observada uma redução considerável nos erros cometidos por pessoas disléxicas durante a redação eletrônica de documentos quando auxiliadas por verificadores ortográficos.

Referências

[1] `http://fmg-www.cs.ucla.edu/geoff/ispell.html`.

[2] `http://savannah.gnu.org/download/aspell/manual/user/1_Introduction.html`.

[3] `http://www.dcc.unicamp.br/ortho/info.html`.

[4] J. J. Almeida and U. Pinto. Jspell – um módulo para análise léxica genérica de linguagem natural. *Actas do Congresso da Associação Portuguesa de Lingüística*, 1995.

[5] K. W. Church and L. F. Rau. Commercial applications of natural language processing. *Communications of the ACM*, 38(11):71–79, 1995.

[6] H. C. da Fonseca Pacheco. Uma ferramenta de auxílio à redação. Dissertação de mestrado, Universidade Federal de Minas Gerais, Outubro 1996.

[7] A. C. R. de Almeida, H. C. da Fonseca Pacheco, and M. L. B. de Carvalho. Verifica: Um verificador ortográfico para a língua portuguesa. Technical Report RT-017, Departamento de Ciência da Computação - UFMG, 1996.

[8] A. Dickinson, P. Gragos, and A. F. Newell. Ongoing investigation of the ways in which some of the problems encountered by some dyslexics can be alleviated using computer techniques. *Proceedings of the fifth international ACM conference on Assistive technologies table of contents*.

[9] T. Kowaltowski, C. L. Lucchesi, and J. Stolfi. Finite automata and efficient lexicon implementation.

[10] T. Kowaltowski, C. L. Lucchesi, and J. Stolfi. Application of finite automata in debugging natural language vocabularies. Technical Report 93-21, Computer Science Dept., University of Campinas, Brazil, 1993.

[11] K. Kukich. Technique for automatically correcting words in text. *ACM Computing Surveys (CSUR)*, 24(4):377–439, 1992.

[12] C. L. Lucchesi and T. Kowaltowski. Applications of finite automata representing large vocabularies. *Software - Practice and Experience*, 23(1):15–30, 1993.

[13] F. Mazocco. Escrita digital: Muito além do óbvio. *Univerciência*, pages 50–58, Dezembro 2002.

[14] D. A. McAllester and R. Givan. Natural language syntax and first-order inference. *Artificial Intelligence*, 56(1):1–20, 1992.

[15] M. McIlroy. Development of a spelling list. *IEEE Trans. on Communications*, COM-30(1):91–99, Janeiro 1982.

[16] L. H. M. Rino, A. di Fellipo, G. M. Pinheiro, R. T. Martins, V. M. Fillié, R. Hasegawa, M. das Graças Volp, and Nunes. Aspectos da construção de um revisor gramatical automático para o português. *Estudos Linguísticos*, 31, Maio 2002.

[17] L. H. M. Rino, A. D. Felippo, G. M. Pinheiro, R. T. Martins, V. M. Fillié, R. Hasegaw, and M. G. V. Nunes. Aspectos da construção de um revisor gramatical automático para o português. *Estudos Lingüísticos*, 31, Maio 2002.

[18] T. N. Turba. Checking for spelling and typographical errors in computer-based text. In *Proceedings of the ACM SIGPLAN SIGOA symposium on Text manipulation*, pages 51–60, 1981.

[19] J. N. Vieira. Fundamentos teóricos da computação. Material utilizado na disciplina de Teoria de Linguagens/Fundamentos Teóricos da Computação, oferecida pelo DCC/Universidade Federal de Minas Gerais, 2002.

[20] R. Vieira and V. L. S. Lima. Lingüística computacional: Princípios e aplicações. *As Tecnologias da Informação e a questão social: anais*, 2:47–88, 2001.