O Reconhecimento e *Parsing Top-down* de Gramáticas Livres de Contexto

Rayssa Küllian Martins

Instituto de Matemática e Estatística, Universidade de São Paulo

19 de outubro de 2014

**Resumo**

Neste trabalho foi desenvolvida uma implementação do algoritmo de *Earley* para o reconhecimento e *parsing* de Gramáticas Livres de Contexto para sentenças em Português dado um córpus.

Foram realizadas análises e comparações de resultados em diferentes casos para encontrar possíveis necessidades de melhoria em termos de manipulação dos dados, tempo de execução e cobertura do algoritmo.

1. Introdução

Processar uma linguagem natural é uma ideia antiga que evoluiu de muitas maneiras. Entre as primeiras etapas de processamento de qualquer sistema que requeira esta ideia envolvem análises sintática e semântica, um conhecimento linguístico e não apenas computacional.

A maioria das atividades que envolvem conhecimento linguístico com processamento natural de linguagem pode ser vista como uma tentativa de resolver problemas de ambiguidade. Identificar se a palavra *“canto”* é um verbo ou substantivo, por exemplo, pode ser resolvido através da classisificação morfossintática (*part-of-speech tagging*), identificar diferentes sentidos para uma mesma palavra pode se resolver através da desambiguação *word sense*, entre outros. Tarefas que podem parecer simples quando realizamos mentalmente, mas envolvem inúmeras técnicas e suas combinações para habilitar uma interpretação adequada.

Neste artigo discutiremos uma abordagem para implementação de um algoritmo de *parsing* de Gramáticas Livres de Contexto e apresentaremos os resultados obtidos, bem como uma descrição dos passos realizados até o final de seu desenvolvimento.

1. Fundamentação Teórica

A estrutura matemática mais comum para modelagem de estruturas de constituintes são as Gramáticas Livres de Contexto (JURAFSKY; MARTIN, 2006). Estas gramáticas representam todas as maneiras como os símbolos de uma sentença podem ser organizados e ordenados e são constituídas de regras e terminais (este formando um léxico). Uma regra NP (*Noun Phrase)*, por exemplo, pode ser formada tanto por “NP-> NPR” quanto por “NP-> PRO$ NPR”.

GLC’s podem ser utilizadas para gerar sentenças ou determinar a estrutura de uma sentença, porém não especificam uma árvore de uma sentença deve ser computada, tarefa chamada de *parsing* sintático. Esta necessidade é suprida através da implementação de algoritmos que usam estas gramáticas para produzir suas árvores.

Os algoritmos de *parsing* estão divididos em duas abordagens: *bottom-up* e *top-down*, onde uma inicia sua busca pelos terminais até identificar a conclusão da sentença e outra faz o caminho inverso e identifica os nós da árvore a partir do nó raiz. Cada abordagem possui suas vantagens e desvantagens, porém discutiremos em detalhes a implementação do algoritmo de Earley *top-down*.

1. *Top-down* Parser: Earley

Através de uma busca *top-down,* o algoritmo de Earley tem como sua estratégia principal percorrer um *array* (*chart*) *com* N+1 entradas, sendo N o número de terminais na sentença de entrada. Cada terminal tem suas possibilidades previstas em uma entrada do *chart*, tendo sua percussão iniciada em um estado inicial *dummy* contendo as possíveis cabeças de árvores. Em nosso caso específico: *S → IP CP FRAG*, obtida através da análise das árvores do córpus dado.

Conforme o caminho é percorrido no *chart*, são gerados estados representando as regras gramaticais previstas, o progresso realizado até aquele ponto e as posições de início e término para aquela entrada no *chart*. O progresso é anotado como *dotted rules*, onde um ● é utilizado a esquerda da regra para identificar que ela ainda não foi processada e a direita da regra para indicar que ela foi processada.

Todo o *chart* é percorrido e suas regras são processadas. Caso a próxima categoria a ser processada seja um *part-of-*speech, ou seja, faça parte do léxico, é então identificada a palavra da sentença correspondente àquela categoria naquela posição específica e sua regra assinalada como concluída (progresso realizado) para avançar o processamento do algoritmo.

**function** EARLEY-PARSE(*words*, *gramar*) **returns** *chart*

ENQUEUE((ɣ → S ●, [0,0]), *chart*[])

**for** *i*<- **from** 0 **to** LENGTH(*words*) **do**

**for each** *state* **in** *chart*[*i*] **do**

**if** INCOMPLETE?(*state*) **and**

NEXT-CATEG(*state*) is not part-of-speech **then**

PREDICTOR(*state*)

**else if** INCOMPLETE?(*state*) **and**

NEXT\_CATEG(*state*) is a part-of-speech **then**

SCANNER(*state*)

**else**

COMPLETER(*state*)

**end**

**end**

**return**(*chart*)

Figura 2.1: Estrutura do algoritmo de Earley

Para realizar todos os processos descritos, o algoritmo de Earley é subdividido em três tarefas principais: *PREDICTOR*, *SCANNER* e *COMPLETER*. O *PREDICTOR* realiza a tarefa de prever as possíveis regras para cada categoria dos estados, enquanto o *COMPLETER* realiza a atividade de assinalar o progresso de processamento para os estados. A última tarefa de identificação de terminais fica a cargo do *SCANNER*.

**procedure** PREDICTOR((*A* → α ● B *β*, [*i,j*]))

**for each**(*B* → ɣ) in GRAMMAR-RULES-FOR(*B*, *grammar*) **do**

ENQUEUE((B → ● ɣ, [*i,j*]), *chart*[*j*])

**end**

**procedure** SCANNER((*A* → α ● B *β*, [*i,j*]))

**if** B ⊂ PARTS-OF-SPEECH(*word*[*j*]) **then**

ENQUEUE((B → *word*[*j*], [*j,j*+1]), chart[*j*+1])

**end**

**procedure** COMPLETER((B → ɣ ●, [*j,k*]))

**for each**(A → α ● B *β*, [*i,j*]) in *chart*[*j*] **do**

ENQUEUE((A → α B ● *β*, [*i,k*]), *chart*[*k*])

**end**

Figura 2.2: Implementação dos métodos que constituem o *Earley*

Foram aplicadas melhorias neste algoritmo sem impactar em seu comportamento adequado. Estes detalhes serão abordados na seção 2.3 de estratégias de implementação.

1. Manipulação dos Dados

Os dados foram apresentados estruturados em um arquivo de córpus, isto é, diversas sentenças com cada uma de suas palavras previamente classificadas (*POS tags*) representadas através de suas árvores gramaticais (*treebanks*).

A leitura do arquivo foi realizada linha por linha e caracter por caracter, utilizando como base estrutural a abertura e fechamento de parênteses, tal como sua contagem para determinar qual regra estava sendo concluída ou inicializada.

Enquanto a regra não havia sido lida por completo, uma estrutura adicional era manuseada para determinar seu conteúdo e organização. A cada regra lida, um valor inteiro era então associado a regra, determinando seu nível na árvore. Todas as regras seguintes eram formadas por seus ramos anteriores mais seu próprio nível e valor. Assim que o fim da regra era identificado, uma leitura era realizada na lista para capturar todos os elementos pertencentes a regra concluída naquele momento.

Caso a linha em leitura contivesse uma sequência de caracteres que não fossem maiúsculos, descaracterizando a leitura de uma categoria POS, era então realizada a leitura de um léxico. Não foi realizado nenhuma exclusão, todos os casos, incluindo pontuações, são terminais e integram a estrutura de léxico construída, de acordo com a especificação de Gramáticas Livres de Contexto (XXXXXXXX ALAIR).

Apesar de seguir uma estrutura, os dados do córpus não são de fácil manipulação. Foi utilizada uma estrutura de Mapa para salvar as regras como chaves e as sentenças como uma lista associada de valores não duplicados.

Além do tratamento inicial para geração da gramática e suas respectivas árvores, foi necessário realizar algumas atividades de pós-processamento nas estruturas coletadas a partir do córpus para obter o comportamento esperado do Earley:

* Regras recursivas (e.g., IP-> IP) foram removidas.
* Ciclos recursivos (e.g., IP-> NP, NP->PP, PP-> IP) foram removidos.

Houve ainda um caso especial de tratamento. Existem regras que estão presentes no córpus tanto como regras gramaticais quanto no léxico (e.g., NP-> PRO$ NPR e NP-> elliptical). Isto causava um comportamento inapropriado no algoritmo, uma vez que ao invés de entrar no *loop* do PREDICTOR, descrito na Figura 2.2, o estado era encontrado no léxico e então identificado como *part-of-speech*, entrando inesperadamente no loop do *SCANNER*. Desta forma, o algoritmo deixava de reconhecer sentenças que possuíssem estes casos já que muitos estados deixavam de ser previstos. Fez-se necessário a análise de todas as regras gramaticais e todo o léxico para identificação destes casos:

|  |  |  |
| --- | --- | --- |
| **POS TAG** | **REGRA** | **LÉXICO** |
| NP | 328 | 2 |
| WPP | 8 | 1 |
| VB | 4 | 1430 |

O *POS tag* NP estava presente em 328 regras gramaticais e em apenas outras 2 regras de léxico. O WPP estava presente em 8 regras gramaticais e 1 léxico. O VB estava presente em 1430 regras de léxico e apenas 4 regras gramaticais. Para solucionar este problema, levou em conta a proporção de cada *POS Tag* presente em cada estrutura e foram excluídas as menores contagens para menor impacto e evitar efeitos colaterais no algoritmo (e.g., se mantiveram as 328 regras gramaticais de NP, excluindo suas duas ocorrências no léxico). Nos experimentos testados não houve nenhum impacto negativo no comportamento correto do algoritmo após estas mudanças e o *bug* foi corrigido conforme esperado.

1. O Problema

Asfafasfas

Melhor colocar antes pq daí na seção seguinte descrevo como resolvi....

1. Arquitetura e Estratégias de Implementação

- comparar primeiros resultados com últimos

- threads

Numero ideal de threads (qtde): citar referencia do livro e argumentar q testes com maior qtde não surtiam resultados melhores;

Aumento de consumo de memória ;

Tarefas triviais não precisam de thread;

- remoção de predição de estados inúteis

- redução do consumo de memória

chamada ao garbage collector;

remoção de estruturas não utilizadas;

- melhorias de performance

utilização de estruturas adicionais como índice para habilitar .contains() pegar o get() id dessa regra e busca-la diretamente na lista completa de gramar;

completer e predictor não percorrem mais as listas inteiras.. consultam o índice, pegam apenas as ocorrências da X regra em questão e as processam;

1. O problema

*Acho que já vou deixar abordado na introdução....*

1. Metodologia

O córpus *“aires-treino.parsed”,* utilizado nas duas distribuições para testes, está em Português (Brasil) e é constituído de regras de Gramáticas Livres de Contexto, ou seja, cada sentença é sintaticamente anotada com sua respectiva árvore de *parse*, formando um córpus *treebank* com seu POS *tagset* específico (JURAFSKY; MARTIN, 2006). A imagem 3.1 mostra a árvore para a sentença “alguma vez se havia de ver a vaidade sem lugar .”.

(IP

(NP (Q Alguma)

(N vez))

(NP (SE se))

(HV havia)

(PP (P de)

(IP (VB ver)

(NP (D a)

(N vaidade))

(IP

(PP (P sem)

(NP (N lugar))))))

(. .))

Figura 3.1: Exemplo de *parser tree* no córpus “*aires-treino.parsed”*

Para análise e avaliação de desempenho do algoritmo de *Earley* foram realizados cinco experimentos em que se extraiu aleatoriamente as partes de treinamento. A distribuição dada consiste em:

* Gramática extraída de 80% do córpus de treinamento
* Gramática verificada em 20% do córpus de teste

As regras gramaticais foram extraídas do córpus de treinamento e as sentenças foram extraídas do córpus de teste.

As medidas utilizadas para avaliação foram *cobertura* e *precisão*, tal que cobertura é definida pela proporção de sentenças do córpus de treinamento que foram reconhecidas pela gramática e precisão definida como pela percentagem de sentenças que admitem a árvore desta sentença no córpus.

Caso a sentença não fosse reconhecida, a precisão foi então utilizada para descrever a quantidade de *brackets* encontrados pelo *parser* que estão presentes na árvore desta sentença no córpus (e.g., árvore do córpus definida por “IP-> NP NP VB”, o algoritmo *reconheceu* a sentença, mas com a árvore de parse “IP-> NP VB ADV“) mesmo quando não reconhecida.

Isto não se mostrou uma boa estratégia ou não agregou nenhum tipo de informação útil aos resultados, uma vez em todos os experimentos executados não haviam *brackets* em comum em caso de sentenças não reconhecidas.

sentença extraída - reconhecida - precisão - tempo

Como método de correção para avaliar se o algoritmo se mantinha com o comportamento esperado durante o desenvolvimento das melhorias citadas na seção 2.3, utilizou-se o exemplo contido no livro do Jurafsky.

1. Resultados

Asfasfas

1. Experimento 1 (gráficos + breves explc.)

Asfasfasfas

1. Experimento 2

Asfasfasfsa

1. Trabalhos futuros (OPT)

Considerando os resultados de cobertura e precisão, faz ainda mais sentido o segundo exercício relacionar a aprimoração do Earley para considerar uma PCFG.

1. Conclusão

~ BEM breve!

"Apesar do problema SAT ser um problema NP-Completo, é possível resolver eficientemente várias instâncias e a razão M/N é um bom indicador se uma certa distância aleatória pode ser resolvida eficientemente."

Referências

JURAFSKY, Daniel; MARTIN, James H.. **Speech and Language Processing**: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. 2. ed., Pearson Prentice Hall, 2009.

Citação com autor incluído no texto: Jurafsky e Martin (2006)

Citação com autor não incluído no texto: (JURAFSKY; MARTIN, 2006)

GOETZ, Brian et al. **Java Concurrency In Practice**. Addison Wesley, 2006.

Citação com autor incluído no texto: Goetz et al. (2006)

Citação com autor não incluído no texto: (GOETZ et al., 2006)

// Removes recursive rules:

// ,-> ,

// IP-> IP

// .-> .

// Handles "(, OPEN)" and "(, CLOSE)" cases (4 rules instances):

// - ",-> O"

// - ",-> C"

// Can not remove ALL cases that has only one POS tag (e.g.: NP-> N)!

// The best way of overcoming the special cases is removing specifically

// each one of them (IP-> NP, NP->PP and PP-> IP, a recursive cycle).

grammarRules.remove("IP-> NP");

grammarRules.remove("NP-> PP");

grammarRules.remove("PP-> IP");

grammarRules.remove("IP-> CP");

grammarRules.remove("CP-> IP");

grammarRules.remove("NP-> CP");

grammarRules.remove("PP-> CP");

grammarRules.remove("NP-> IP");

grammarRules.remove("IP-");

lexicon.remove("NP-> elliptical");

lexicon.remove("NP-> nos");

lexicon.remove("WPP-> 0");

lexicon.remove("VB");