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.

2 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*.

2.1 *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: $S \rightarrow IP\ CP\ FRAG$, 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 → γ) in GRAMMAR-RULES-FOR(B, grammar) do
        ENQUEUE((B → • γ, [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 → γ •, [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.

2.2 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 (SIPSER, 2007).

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.

2.3 O Problema

Apesar de ter uma boa lógica em sua estrutura, o algoritmo de Earley possui um desempenho cada vez menos satisfatório a medida que o tamanho da sentença aumenta, uma vez que as entradas do *chart* crescem e os estados a serem percorridos se multiplicam. A quantidade de regras gramaticais processadas também impacta diretamente neste comportamento. O córpus discutido neste trabalho, em específico, possui 2093 regras gramaticais e 3660 regras no léxico.

Há também um problema relacionado a forma como o algoritmo é executado: sua implementação original é sequencial. Esta característica quando executada em uma sentença com mais de 8 palavras, por exemplo, fica ainda mais evidente durante o percurso.

Assim, foram adotadas medidas para otimizar o algoritmo que serão discutidas na seção 2.4 seguinte.

2.4 Arquitetura e Estratégias de Implementação

Além da implementação básica e pura do algoritmo, algumas adaptações e melhorias foram necessárias com o intuito de facilitar o teste contínuo e os futuros experimentos, conforme mencionado na seção anterior.

Os primeiros testes de execução com o algoritmo puro levaram 40 minutos, em média, para processar uma sentença de 8 palavras e 4 minutos, em média, para processar uma sentença de apenas 3 palavras. Manter esta estratégia dificultaria muito a execução ideal dos experimentos, podendo levar dias para concluir um único experimento. Para executar um experimento, por exemplo, cuja parte de treinamento é composta de 20% do córpus num total de 418 sentenças, poderia levar um tempo muito acima do necessário. No caso de testes curtos e rápidos realizados durante o desenvolvimento, estes problemas tornavam inviáveis a implementação, correção e melhoria do código.

Além destas questões mais simples de serem detectadas, foi notado um consumo de memória acima do esperado com picos de mais de 1GB alocado para o *parsing* de uma sentença com 10 palavras levando a interrupção da aplicação.

Os primeiros passos, portanto, foi possibilitar que o algoritmo pudesse ser concluído, mesmo que ainda em um tempo acima do esperado. Inúmeras análises de performance foram realizadas para caracterizar os gargalos de alocação de memória e identificar os objetos responsáveis pelo problema. A causa esteve sempre relacionada à alguma estrutura de dados do tipo lista, ou duplicada de alguma estrutura já existente ou uma estrutura desnecessária no atual ponto de desenvolvimento.

Entretanto, apesar da remoção destes gargalos, ainda existia um problema de memória ocasionado pela grande quantidade de estados gerados e armazenados em memória no *chart*, tanto quanto o esforço despendido para percorrê-lo por completo.

Outras análises foram realizadas e fez-se necessária a reestruturação dos laços de repetição que percorrem as regras gramáticas nos estados do *chart*. Simultaneamente ao *chart*, foi criada uma lista de índices de cada estado representado pela atual regra a ser

processada. Assim, ao invés de percorrer um *loop* com todos os possíveis estados contendo determinada categoria a ser processada, as posições desta categoria eram identificadas no índice e imediatamente capturadas para a associação de sua regra completa. Ou seja, apenas as ocorrências da atual categoria eram processadas.

Esta mesma lógica se aplicou tanto no *PREDICTOR* quanto no *COMPLETER*. Foram realizados testes para o *SCANNER*, porém não houve melhoria significativa de performance devido ao tamanho da lista de léxicos ser muito inferior às demais listas mencionadas.

Em uma nova etapa, foi analisado o comportamento sequencial do *Earley* e avaliada a hipótese de paralelizá-lo, algo que exigiu inúmeros pequenos testes para confirmar o impacto destas alterações no comportamento do algoritmo. Uma paralização completa do algoritmo tende a não só influenciá-lo, mas a impedir seu correto funcionamento.

Além de avaliar possíveis impactos negativos, também foi levado em consideração a real necessidade de utilizar *Threads*. Apesar de serem muito utilizadas, nem sempre são necessárias quando se tratam de atividades triviais, ou muito curtas e rápidas ou atividades de baixa performance não relacionadas ao consumo de CPU, mas consumo de disco, I/O, entre outros.

Levando estes pontos em consideração, foram identificadas tarefas mais específicas dentro do algoritmo que pudessem ser paralelizadas e trouxessem um ganho significativo de performance: *PREDICTOR* e *SCANNER*.

Cada atividade é executada com uma quantidade específica de *Threads* que demonstrou um desempenho satisfatório: 3 *Threads* para cada execução do *PREDICTOR* e 5 *Threads* para execução do *COMPLETER*, uma vez que este percorre todos os estados do *chart*. A somatória destes números equivale a quantidade de *cores* disponíveis na CPU utilizadas para os testes, definida como o número ideal de *Threads* de acordo com Goetz et al. (2006). Após a execução de cada *PREDICTOR* e *COMPLETER*, foi realizada uma chamada ao *Garbage Collector* que, mesmo não garantindo sua execução de acordo com a API Java, diminui significativamente o consumo de memória.

Por último foi realizada a melhoria com maior ganho de performance, pois corrige o maior gargalo do Earley: redução dos estados a serem processados. Quando o *PREDICTOR* é chamado, é passado o estado atual do *chart* para identificação da próxima categoria a ser processada e então prever todas as suas regras gramaticais possíveis. Porém, se determinada regra não possui nenhum *POS tag* relacionado às palavras da sentença em questão, não há necessidade de processá-la. Portanto, são previstas apenas as regras que contém categorias onde *existe a possibilidade* de processar alguma palavra da sentença. Isso reduz significativamente a quantidade de estados no *chart* para serem processados e terem seus progressos atualizados. Sentenças de 3 palavras caíram seu tempo de execução de uma média de 3 minutos para em média 26 segundos. Sentenças de 11 palavras foram processadas em 1 minuto e 13 segundos, em média, enquanto anteriormente exigia 34 minutos e 21 segundos em média. Sem esta alteração não seria possível executar muitas sentenças com mais de 20 palavras.

3 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, e em momentos distintos, as partes de treinamento e teste. 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 pela percentagem de sentenças que admitem a árvore desta sentença no córpus.

$$Cobertura = \frac{sentenças\ reconhecidas}{total\ de\ sentenças}$$

$$Precisão = \frac{árvores\ admitidas\ pelo\ córpus}{sentenças\ reconhecidas}$$

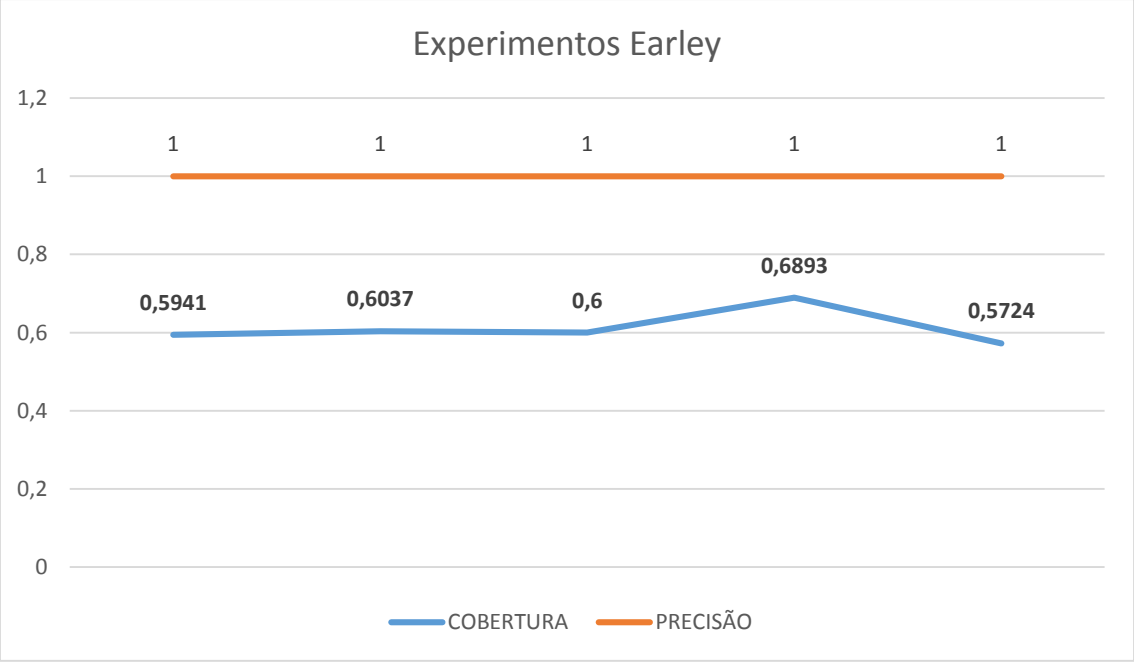
Caso a sentença não fosse reconhecida, a precisão foi então utilizada para descrever a quantidade de *brackets* encontrados pelo *parser* que estão presentes na árvore desta sentença no córpus (e.g., árvore do córpus definida por "IP-> NP NP VB", o algoritmo *reconheceu* a sentença, mas com a árvore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

$$Precisão^{*} = \frac{brackets\ corretos}{total\ de\ brackets}$$

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. Não foram utilizadas medidas para mensurar este cenário, apenas o reconhecimento do comportamento esperado do algoritmo ou não.

4 Resultados

A medida de precisão aplicada às sentenças não reconhecidas, como mencionado na seção anterior, não se mostrou uma boa estratégia ou agregou algum tipo de informação útil aos resultados, uma vez em todos os experimentos executados não haviam *brackets* em comum em caso de sentenças não reconhecidas. Pode ser necessária uma nova estratégia para aplicar este conceito futuramente.



Em cada experimento são relatadas as medidas de *cobertura* e *precisão*:

4.1 Experimento 1

COBERTURA	PRECISÃO
0,5941	1

4.2 Experimento 2

COBERTURA	PRECISÃO
0,6037	1

4.3 Experimento 3

COBERTURA	PRECISÃO
0,6	1

4.4 Experimento 4

COBERTURA	PRECISÃO
0,6893	1

4.5 Experimento 5

COBERTURA	PRECISÃO
0,5724	1

5 Conclusão

O cenário de Processamento de Linguagem Natural é muito mais complexo do que o imaginado. Existem inúmeras outras atividades para serem realizadas após um tratamento básico como o executado pelo Earley já neste ponto inicial existem dificuldades.

Os valores de cobertura não foram equivalentes a expectativa mesmo com a implementação de um algoritmo de Earley específico para o córpus dado. Podem até fazer algum sentido quando se imagina o processamento de sentenças diferentes daquelas presentes no treinamento, porém, inúmeras regras possuem semelhança e descartam este pensamento, podendo influenciar apenas no quesito de precisão. Já a medida de precisão demonstrou um comportamento inesperado, onde todas as sentenças reconhecidas tinham sua árvore idêntica ao treinado no córpus. Será necessário avaliar este cenário.

Certamente o maior desafio foi compreender que o processamento de linguagens não é uma mágica instantânea, ou ao menos em um tempo esperado, resultado que foi buscado incessantemente durante semanas, levando à imensurável perda de tempo. A

única vantagem foi ter estudado, analisado e testado todas as possibilidades de melhorias de performance, desde redução de memória, utilização de disco, estrutura de dados mais performáticas, paralelização, índices, entre outros, o que resultou em um ganho de performance que talvez não seria obtido se o objetivo idealizado fosse diferente.

6 Trabalhos futuros (OPT)

Considerando os resultados de cobertura e precisão, é compreensível a necessidade do segundo exercício relacionar a aprimoração do Earley para aceitar uma PCFG de entrada e tratar toda a sua gramática de maneira probabilística. Através desta abordagem, seria possível dividir a árvore sintática final gerada com todas as possibilidades envolvidas em árvore menores e distintas de acordo com as probabilidades.

Após estas duas etapas dos dois exercícios estarem completas, certamente será de muita valia aplicar técnicas mais refinadas de Processamento Natural de Linguagem, Análise Sentimental, ou qualquer tipo de mineração no texto.

Também será avaliada a aplicação destas técnicas em linguagens mais desestruturadas como as redes sociais, por exemplo.

Referências

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

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.

SIPSER, Michael. **Introdução À Teoria Da Computação**. 2. ed. Cengage Learning, 2007.