Introducción al Procesamiento del Lenguaje Natural

Segundo Cuatrimestre del 2014

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Parsing de dependencias

Grupo:

Integrante	LU	Correo electrónico
Ramiro Camino	264/06	ramirocamino@gmail.com
Guillermo Alejandro Gallardo Diez	032/10	gagdiez@hotmail.com
Martín Langberg	086/10	martinlangberg@gmail.com
Fabrizio Borghini	406/10	fabriborghini@gmail.com

Índice

1. Introducción

La meta del parsing de dependencias es, dado un string de entrada, generar un grafo acíclico dirigido que indique de qué manera dependen las palabras entre sí. En este trabajo estudiamos los aportes al estado del arte que han realizado tres trabajos de Joakim Nivre, poniendo especial énfasis en el uso que se le da a la pila y no en el uso de aprendizaje automático.

El trabajo está estructurado de la siguiente manera: en primer lugar se presentan las definiciones generales necesarias para entender los papers estudiados, luego se explicará los parsers propuestos por Nivre. Luego de esto se hará una comparación de los parsers teniendo en cuenta el rol que juegan las pilas en cada uno y finalmente la última sección contendrá las conclusiones del grupo.

2. Definiciones

2.1. Grafo de dependencias

Sea $w = w_1 \dots w_n$ un string de entrada. Un *intervalo* $[i, j]$ es un conjunto de la forma $[i, j] = \{w_k : i \leq k \leq j\}$.

Un grafo de dependencias es un grafo dirigido $G = (V_w, E)$ donde $V_w = [1, n]$ y $E \subseteq V_w \text{x} V_w$. Llamamos a $(w_i, w_j) \in E$ un *link de dependencia* de w_i a w_j, y decimos que w_i es el padre (o cabeza) de w_j, o que w_j es hijo de w_i. Por conveniencia escribimos $(w_i, w_j) \in E$ si el link (w_i, w_j) existe; escribimos $w_i \leftrightarrow w_j \in E$ si existe alguno de los links entre ambos tokens; $w_i \rightarrow *w_j \in E$ si existe un camino dirigido (puede ser vacío) de w_i a w_j; por último, $w_i \leftrightarrow *w_j \in E$ si existe alguno de los caminos dirigidos entre ambos tokens.

La proyección de un nodo w_i es el conjunto de dependencias reflexivo transitivas de w_i, es decir: $\lfloor w_i \rfloor = \{w_j \in V : w_i \rightarrow *w_j\}$. En otras palabras, la proyección de un nodo w_i es el conjunto de nodos a los que puede llegar w_i.

2.2. Bosque de dependencias

Un grafo de dependencias es un bosque si y sólo si es acíclico y todo nodo posee un solo padre (ergo, es conexo). Si el bosque posee un solo nodo raíz, entonces se dice que es un *árbol de dependencias*.

2.3. Bosque proyectivo

Un bosque de dependencias es proyectivo si y sólo si $\lfloor w_i \rfloor$ es un intervalo para cada palabra $w_i \in [1, n]$. Dicho de otra manera, $(w_i \leftrightarrow w_k \wedge w_i < w_j < w_k) \Rightarrow (w_i \rightarrow *w_j \vee w_k \rightarrow *w_j)$.

2.4. Grafo de dependencias bien formado

Un grafo de dependencias esta bien formado si y sólo si es un árbol de dependencias proyectivo.

2.5. Planaridad

Un grafo se dice planar si no posee *links cruzados*. Sean (w_i, w_k), (w_j, w_l) links de dependencia, se asume sin pérdida de generalidad que $min(i, k) \leq min(j, l)$. Decimos que estos links se cruzan si y sólo si $min(i, k) \leq min(j, l) \leq max(i, k) \leq max(j, l)$.

2.6. Multiplanaridad

Un grafo de dependencias $G = (V, E)$ se dice $m - planar$ si y sólo si existen grafos de dependencias planares $G_1 = (V, E_1) \dots G_m = (V, E_m)$ tales que $E = \bigcup E_i$. En [3] se demuestra que el problema de determinar si un grafo es k-planar es análogo a determinar si es k-coloreable.

2.7. Sistema de transiciones

Un sistema de transiciones representa un parser como un grafo donde los nodos son estados y las aristas transiciones. Las transiciones son un conjunto de operaciones elementales que pueden ser aplicadas durante el parsing, haciendo cambiar el estado del mismo hasta que finaliza.

Más formalmente un sistema de transiciones es una tupla $S =< C, T, c_s, C_t >$ donde:

- C es un conjunto de posibles configuraciones.
- T es un conjunto de transiciones tales que $t \in T | t : C \to C$.
- c_s es una función que mapea cada posible entrada a una configuración. Es decir $c_s(w) \in C$.
- $C_t \subseteq C$ es un conjunto de configuraciones terminales.

2.8. Oráculo

Un oráculo es una función $o : C \to T$. Esto quiere decir que para cada configuración el oráculo determina que transición realizar.

3. Parsers

3.1. Inductive Dependency Parsing (Malt Parser) [1][2]

En [1],[2] Nivre presenta un parser *data-driven* de dependencias determinístico con complejidad de orden lineal; para conseguir que el parser sea determinístico se utiliza un oráculo que decide las transiciones a realizar; *data-driven* quiere decir que el oráculo es entrenado mediante *machine-learning* utilizando un corpus de datos anotado, o sea, no se crean gramáticas ni lexicones [1].

Empezaremos hablando del sistema transicional de dicho parser. El mismo se puede representar usando una tripla $< \Sigma, B, A >$, donde Σ es la pila del parser, B es el buffer de lectura y A es el conjunto de dependencias que se deriva.

Las transiciones entre configuraciones para este parser son las siguientes:

- **Inicial**: $c_s(w_1 \dots w_n) =< [], [w_1 \dots w_n], \emptyset >$
- **Terminal**: $C_t = \{< \Sigma, [], A >\in C\}$
- **Transiciones**:
  - SHIFT: $< \Sigma, w_i|B, A > \rightarrow < \Sigma|w_i, B, A >$
  - REDUCE: $< \Sigma|w_i, B, A > \rightarrow < \Sigma, B, A >$
    solo si $\exists k|(w_k, w_j) \in A$
  - LEFT-ARC: $< \Sigma|w_i, w_j|B, A > \rightarrow < \Sigma, w_j|B, A \cup \{(w_j, w_i)\} >$
    solo si $\nexists k|(w_k, w_i) \in A$
  - RIGHT-ARC: $< \Sigma|w_i, w_j|B, A > \rightarrow < \Sigma|w_i|w_j, B, A \cup \{(w_i, w_j)\} >$
    solo si $\nexists k|(w_k, w_j) \in A$

Como bien ya se dijo, la transición entre estados es dirigida por un oráculo entrenado previamente con un corpus.

3.1.1. Algoritmo de Covington [4]

Una alternativa que propone Covington es usar una lista en vez de una pila, es decir, poder enlazar cualquier token pasado con el siguiente token del input. Esto permite admitir dependencias no proyectivas, pero tiene un costo cuadrático en función del tamaño del input.

3.1.2. Dependencias pseudo-proyectivas

Este parser viene con una herramienta para tratar con dependencias no proyectivas realizando transformaciones sobre los grafos, y las llama dependencias pseudo-proyectivas:

1. Cuando entrena, si encuentra un grafo no proyectivo, lo transforma moviendo hacia arriba los ejes cruzados hasta que no estén cruzados, y agrega anotaciones en los ejes para dejar constancia de esta transformación.

2. Cuando parsea trabaja como siempre.

3. Obtiene grafos de dependencia proyectivos pero con etiquetas enriquecidas.

4. Al final si el grafo tiene anotaciones trata de invertir la transformación para obtener un grafo no proyectivo.

3.2. Parser for 2-planar dependency structures [3]

En [3] Nivre demuestra que se pueden cubrir un 99 % de las frases en los treebanks más importantes utilizando grafos 2-planares al momento de generar las dependencias y por ello presenta un parser que aprovecha esta característica. Dicho parser se puede representar utilizando una tupla $< \Sigma_0, \Sigma_1, B, A >$ donde Σ_0 es la pila activa, Σ_1 la pila inactiva, B es el buffer de lectura y A el conjunto de dependencias que se derivan.

Las transiciones entre configuraciones para este parser son las siguientes:

- **Inicial**: $c_s(w_1 \dots w_n) =< [], [], [w_1 \dots w_n], \emptyset >$
- **Terminal**: $C_t = \{< \Sigma_0, \Sigma_1, [], A >\in C\}$
- **Transiciones**:
  - SHIFT: $< \Sigma_0, \Sigma_1, w_i|B, A > \rightarrow < \Sigma_0|w_i, \Sigma_1|w_i, B, A >$
  - REDUCE: $< \Sigma_0|w_i, \Sigma_1, B, A > \rightarrow < \Sigma_0, \Sigma_1, B, A >$
  - LEFT-ARC: $< \Sigma_0|w_i, \Sigma_1, w_j|B, A > \rightarrow < \Sigma_0|w_i, \Sigma_1, w_j|B, A \cup \{(w_j, w_i)\} >$ solo si $\nexists k|(w_k, w_i) \in A$ y no exista $w_i \leftrightarrow *w_j \in A$
  - RIGHT-ARC: $< \Sigma_0|w_i, \Sigma_1, w_j|B, A > \rightarrow < \Sigma_0|w_i, \Sigma_1, w_j|B, A \cup \{(w_i, w_j)\} >$ solo si $\nexists k|(w_k, w_j) \in A$ y no exista $w_i \leftrightarrow *w_j \in A$
  - SWITCH: $< \Sigma_0, \Sigma_1, B, A > \rightarrow < \Sigma_1, \Sigma_0, B, A >$

Nuevamente se utiliza un oráculo al momento de elegir qué transición utilizar.

4. Comparando Parsers

4.1. Comparación Teórica

Si bien los parsers parecen similares (pues comparten el nombre de casi todas las transiciones y los cambios de configuraciones son parecidos), el agregar una pila produce varios cambios. Para empezar es importante notar que en el parser para estructuras de dependencias 2-planar las transiciones ARC y REDUCE se realizan solo en la pila activa; SHIFT saca el elemento del buffer al igual que antes pero ahora lo guarda en ambas pilas; REDUCE ya no posee restricciones; ARC ya no saca la palabra de la pila al crear los arcos, y, a causa de los cambios descritos, se agrega a estas últimas transiciones una condición en la guarda que permite comprobar que no se produzcan ciclos antes de crear los arcos. Es importante destacar que esta condición conserva la complejidad lineal [3]. El agregado de una nueva pila también hace que el 2-planar parser posea una nueva transición llamada SWITCH que se encarga de intercambiar las pilas, para indicar cuál es la activa.

En resumen, mientras que el Malt parser va generando un grafo de dependencias basándose fuertemente en sus transiciones, 2-planar parser va generando dos bosques de dependencias en cada pila evitando que existan pares de links cruzados dentro de las mismas, a la vez que permite que se crucen links pertenecientes a palabras en distintas pilas [3]. Esto lo realiza de tal manera de tomar en cuenta las guardas y propiedades de multiplanaridad.

4.2. Comparación Empírica

En [3] Nivre hace una comparación empirica de estos parsers, lo cual nos ahorra el tiempo a nosotros de hacer las pruebas. Los resultados del 2-planar parser fueron mejores o muy cercanos a la implementación de Malt parser con transformaciones pseudo-proyectivas [2]. A continuación se muestran los valores que se obtuvo para distintos idiomas usando los parsers explicados:

	Czech				Danish				German				Portuguese			
	LAS	UAS	NPP	NPR	LAS	UAS	NPP	NPR	LAS	UAS	NPP	NPR	LAS	UAS	NPP	NPR
2-P	79.2	85.3	68.9	60.7	83.8	88.5	66.7	20.0	86.5	88.8	57.1	45.8	87.0	90.8	82.8	33.8
M	79.8	85.7	76.7	56.1	83.6	88.5	41.7	25.0	85.7	88.6	58.1	40.7	87.0	90.6	83.3	46.2

Cuadro 1: Accuracy del parsing para el 2-planar parser (2P) comparado con Malt parser con transformaciones pseudo-proyectivas (M). LAS = Labeled Attachment Score; UAS = Unlabeled Attachment Score; NPP = Precision on Non-Projective arcs; NPR = Recall on Non-Projective arcs.

5. Conclusiones

Utilizar dos pilas al momento de parsear dependencias genera un aporte significativo al estado del arte, ya que, como bien explica Nivre en [3] el parser 2-planar es más simple que la implementación pseudo-proyectiva de Malt parser, así como también define una clase concreta y bien conocida de estructuras parseables, mientras que no se sabe qué tipos de estructuras puede parsear el Malt parser. Si a esto le sumamos el hecho de que demuestran que las frases utilizadas en los corpus más importantes son 2-planares, podemos asegurar que el agregar una pila al parser genera otro parser más simple y robusto.

Finalmente podemos agregar que si se quisiera poder cubrir más estructuras, o frases dentro de los corpus, simplemente basta con agregar otra pila y hacer al parser 3-planar, que nuevamente, como ya se demostró en [3], cubre el 100 % de las casos.

6. Bibliografía

1. Nivre, J. (2006), *Inductive Dependency Parsing*, Springer.

2. Nivre, J., J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kübler, S. Marinov and E. Marsi (2007), *MaltParser: A language-independent system for data-driven dependency parsing*, Natural Language Engineering, 13(2), 95-135.

3. Gómez-Rodríguez, C. and Nivre, J. (2010), *A Transition-Based Parser for 2-Planar Dependency Structures*, In Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, 1492-1501.

4. Covington, M. A. 2001. *A fundamental algorithm for dependency parsing*, In Proceedings of the 39th Annual ACM Southeast Conference, pp. 95–102.