Procesamiento de Lenguaje Natural, Revista nº 48 marzo de 2012, pp 43-50 recibido 21-12-11 revisado 13-03-12 aceptado 18-03-12

Dependencias no dirigidas para el análisis basado en transiciones*

Undirected Dependency Structures for Transition-Based Parsing

Carlos Gómez-Rodríguez
Departamento de Computación
Univ. da Coruña, Campus de Elviña
15071, A Coruña, Spain
carlos.gomez@udc.es

Daniel Fernández-González
Departamento de Informática
Univ. de Vigo, Campus As Lagoas
32004, Ourense, Spain
danifg@uvigo.es

Resumen: En este artículo se presenta un nuevo enfoque para abordar el análisis de dependencias basado en transiciones. Se propone que el analizador construya un grafo no dirigido durante el proceso de análisis, en lugar de la estructura de dependencias dirigida clásica. *A posteriori*, la estructura no dirigida es transformada en un árbol de dependencias. Con ello se consigue reducir la propagación de errores propia de estos sistemas. Aplicando este enfoque se obtuvieron variantes no dirigidas de los analizadores Planar, 2-Planar y Covington no proyectivo. Se han llevado a cabo experimentos sobre varios bancos de árboles del *CoNLL-X shared task*, obteniendo resultados para las nuevas variantes que superan a los algoritmos originales en la mayoría de los casos.
Palabras clave: Análisis sintáctico, sintaxis, análisis sintáctico de dependencias, gramáticas de dependencias.

Abstract: In this paper we introduce a new approach to transition-based dependency parsing. We propose that the parser construct an undirected graph during the parsing process, instead of a standard directed dependency structure. *A posteriori*, the output undirected structure is converted into a dependency tree. This alleviates error propagation, a characteristic problem of these systems. We apply this approach to obtain undirected variants of the Planar and 2-Planar parsers and of Covington's non-projective parser. We perform experiments on several treebanks from the CoNLL-X shared task, showing that these variants outperform the original directed algorithms in most of the cases.
Keywords: Parsing, syntax, dependency parsing, dependency grammar

1. Introducción

En los últimos años, el análisis de dependencias sintácticas ha demostrado ser de gran utilidad en multitud de tareas de procesamiento del lenguaje natural. En concreto, los analizadores de dependencias guiados por datos (*data-driven parsers*) tales como los de Nivre, Hall, y Nilsson (2004), McDonald et al. (2005), Titov y Henderson (2007), Martins, Smith, y Xing (2009) o Huang y Sagae (2010) son altamente precisos y eficientes. Estos pueden ser entrenados a partir de datos previamente anotados sin necesidad de disponer de una gramática explícita.

Concretamente, los analizadores basados en transiciones (Nivre, 2008) son un tipo de algoritmos *data-driven* de análisis de dependencias. Estos usan un modelo que asigna puntuaciones a las transiciones entre los posibles estados del analizador. De este modo, utilizando una estrategia de búsqueda determinista, seleccionan en cada momento la transición que se debe aplicar sobre el estado actual del analizador para alcanzar el siguiente estado. Con ello se consigue una complejidad temporal lineal o cuadrática.

Ha quedado demostrado en el trabajo de McDonald y Nivre (2007) que este tipo de analizadores es sensible a la propagación de errores: una elección errónea en un momento

* Esta investigación ha sido parcialmente financiada por el Ministerio de Economía y Competitividad y el FEDER (proyectos TIN2010-18552-C03-01 y TIN2010-18552-C03-02), el Ministerio de Educación (Programa de becas FPU) y la Xunta de Galicia (Rede Galega de Recursos Lingüísticos para unha Sociedade de Coñecemento). Para los experimentos se han empleado los recursos del Centro de Supercomputación de Galicia (CESGA).

ISSN 1135-5948

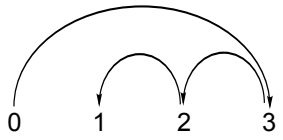

Figura 1: Ejemplo de estructura de dependencias donde el analizador basado en transiciones, manteniendo la restricción de padre único, provoca propagación de errores si erróneamente construye una dependencia $1 \rightarrow 2$ en lugar de $2 \rightarrow 1$ (las dependencias se representan como flechas dirigidas del padre al hijo).

temprano del análisis puede llevar al analizador a un estado incorrecto, lo que desencadenará más errores. Por ejemplo, supongamos que una oración, cuyo análisis correcto aparece representado en la Figura 1, es analizada por un analizador basado en transiciones ascendente o con procesado de izquierda a derecha que ofrezca como salida árboles de dependencias. Este analizador está obligado a mantener la **restricción de padre único** propia de las estructuras árboreas, donde cada nodo únicamente puede recibir un arco de entrada. Si el analizador elige una transición errónea que provoque la construcción de una dependencia de 1 a 2, en lugar de establecer la dependencia correcta de 2 a 1, esto llevará al analizador a un estado donde la restricción de padre único no permitirá crear el enlace correcto de 3 a 2. Por lo tanto, una simple elección errónea puede acarrear dos errores en el árbol resultante.

Con el fin de minimizar esta fuente de errores, se han obtenido nuevas variantes *no dirigidas* de varios analizadores de dependencias existentes. Concretamente, se ha trabajado con los analizadores Planar y 2-Planar de Gómez-Rodríguez y Nivre (2010), así como con el analizador no proyectivo descrito por Nivre (2008) y basado en el algoritmo de Covington (Covington, 2001). Para implementar estas variantes no dirigidas ha sido necesario sustituir las transiciones LEFT-ARC y RIGHT-ARC presentes en los analizadores originales (que se encargaban de crear enlaces de dependencia de derecha a izquierda y de izquierda a derecha, respectivamente) por una simple transición ARC cuyo cometido es el de crear un enlace no dirigido.

La principal ventaja de este enfoque es que el analizador no se ve obligado a mantener la restricción de padre único durante el proceso de análisis. Esto se debe a que, al trabajar con grafos no dirigidos, la dirección de padre a hijo representada en los enlaces de dependencia se pierde. Como consecuencia, el analizador dispone de una mayor libertad, permitiéndole prevenir situaciones donde la rigidez de la estructura arbórea favorecería la propagación de errores, como se reflejaba en la Figura 1.

Por otro lado, los nuevos algoritmos presentan una desventaja importante: obtienen como resultado un grafo no dirigido que debe ser transformado en un árbol de dependencias clásico. Por ello, es necesario llevar a cabo un post-procesado que recupere la dirección de los enlaces de dependencia y genere un árbol de dependencias válido. De esta forma, parte de la complejidad del proceso de análisis se traslada a esta etapa de post-procesado. Ello significa que los nuevos analizadores no dirigidos únicamente mejorarán los resultados ofrecidos por las versiones dirigidas si la simplificación aplicada en la fase de análisis es capaz de evitar más errores que los generados en la etapa de post-procesado. Como se verá más adelante, los resultados experimentales indican que efectivamente esto se cumple.

Este artículo se ha organizado de la siguiente manera. En la Sección 2 se presentan las notaciones y conceptos utilizados en este artículo. En la Sección 3, se exponen las versiones no dirigidas de los analizadores de Gómez-Rodríguez y Nivre (2010) y Nivre (2008). En la Sección 4, se proponen las posibles técnicas de post-procesado que pueden ser utilizadas para recuperar los árboles de dependencias a partir de un grafo no dirigido. Finalmente, la Sección 5 presenta un estudio empírico del rendimiento obtenido por los analizadores, y la Sección 6 contiene las conclusiones finales.

2. Conceptos básicos

2.1. Grafos de dependencias

Sea $w = w_1 \dots w_n$ una cadena de entrada. Un **grafo de dependencias** para w es un grafo dirigido $G = (V_w, E)$, donde $V_w = \{0, \dots, n\}$ es el conjunto de nodos, y $E \subseteq V_w \times V_w$ es el conjunto de arcos dirigidos. Cada nodo en V_w codifica la posición de un *token* en w, y cada arco en E representa una relación de dependencia entre dos *tokens*. Se escribe $i \rightarrow j$ para denotar un arco dirigido (i, j), denominado también como un **enlace de dependencia** de i a j. Se dice que i es el **padre** de j y, a la inversa, que j es el **hijo** o **dependiente** de i. En la práctica, los enlaces de dependencia

están **etiquetados**, pero para simplificar la representación, en este artículo se ignorarán las etiquetas.

Dado un grafo de dependencias $G = (V_w, E)$, se escribe $i \rightarrow^\star j \in E$ si existe un camino dirigido (posiblemente vacío) de i a j; y $i \leftrightarrow^\star j \in E$ si existe un camino (posiblemente vacío) entre i y j en el grafo no dirigido subyacente a G.

La mayoría de representaciones sintácticas basadas en dependencias exigen grafos acíclicos, y que cada nodo del mismo tenga, como mucho, un padre. Los grafos de dependencias que cumplen ambas restricciones se denominan **bosques de dependencias**.

Definición 1 *Un grafo de dependencias G se dice que es un **bosque** si y solo si cumple:*

1. *La restricción de aciclicidad: si existe $i \rightarrow^\star j$, entonces no existe $j \rightarrow i$.*
2. *La restricción de padre único: si existe $j \rightarrow i$, entonces no existe $k \neq j$ tal que $k \rightarrow i$.*

Un nodo que no tiene padre en un bosque de dependencias se denomina **raíz**. Algunas representaciones de dependencias añaden una restricción adicional, según la cual un bosque sólo puede tener un nodo raíz (o, equivalentemente, el bosque debe estar completamente conectado). Un bosque con estas características se denomina un **árbol de dependencias**. Un árbol de dependencias puede obtenerse a partir de cualquier bosque de dependencias enlazando todos los nodos raíz como dependientes de un **nodo raíz artificial**, que convencionalmente se ubica en la posición 0 de la cadena de entrada.

2.2. Sistemas de transiciones

En el marco de trabajo establecido en Nivre (2008), los analizadores basados en transiciones se definen mediante una máquina de estados no determinista denominada **sistema de transiciones**.

Definición 2 *Un **sistema de transiciones** para análisis de dependencias sintácticas es una tupla $S = (C, T, c_s, C_t)$, donde*

1. *C es el conjunto de posibles **configuraciones** del analizador,*
2. *T es el conjunto finito de **transiciones**, que es una función parcial $t : C \rightarrow C$,*
3. *c_s es la función de inicialización que asigna a cada cadena de entrada una única **configuración inicial**, y*
4. *$C_t \subseteq C$ es el conjunto de **configuraciones terminales**.*

Con el fin de obtener un analizador determinista a partir de un sistema de transiciones no determinista, se utiliza un **oráculo** para seleccionar una transición concreta en cada configuración. Un oráculo para un sistema de transiciones $S = (C, T, c_s, C_t)$ es una función $o : C \rightarrow T$. En la práctica, un oráculo se obtiene a partir del entrenamiento de clasificadores sobre un banco de árboles (Nivre, Hall, y Nilsson, 2004).

2.3. Sistemas de transiciones Planar, 2-Planar y Covington

Nuestros analizadores de dependencias no dirigidos se basan en los sistemas de transiciones Planar y 2-Planar de Gómez-Rodríguez y Nivre (2010) y en la versión no proyectiva del analizador de Covington (2001) definida por Nivre (2008). Por motivos de espacio, únicamente se esbozan muy brevemente estos analizadores, pudiéndose consultar una descripción más detallada en las referencias citadas.

Los tres sistemas de transiciones se describen en la Figura 2a. Estos leen la oración de izquierda a derecha. Inicialmente, todas las palabras de la oración se almacenan en una lista B denominada **buffer**. Adicionalmente, el analizador Planar utiliza una **pila** como estructura de datos intermedia; mientras que el algoritmo 2-Planar usa dos pilas, una de las cuales se encuentra **activa** en cada momento. Por su parte, el analizador Covington emplea dos **listas** que contienen las palabras parcialmente procesadas. Las estructuras adicionales de los algoritmos 2-Planar y Covington les permiten reconocer estructuras de dependencias 2-planares y no proyectivas, respectivamente, que representan un conjunto más amplio que las estructuras planares manejadas por el analizador Planar.

Las transiciones de los analizadores son las siguientes: una transición SHIFT se utiliza en cualquiera de los tres algoritmos para leer una palabra de entrada, eliminándola del *buffer* e introduciéndola en las estructuras de datos intermedias de cada analizador (pila, pilas o listas). Una transición REDUCE de los analizadores Planar y 2-Planar, elimina el elemento en la cabeza de la pila activa. Una transición SWITCH cambia la pila que se considera activa, y sólo es aplicable en el analizador 2-Planar. Una transición NO-ARC, propia del analizador Covington, desplaza el nodo ubicado en la cabeza de la primera lista a la cabeza de la segunda lista sin crear ningún ar-

a) Planar: Config. inicial/terminal:		$c_s(w_1 \dots w_n) = \langle [], [1 \dots n], \emptyset \rangle$, $C_f = \{\langle \Sigma, [], A \rangle \in C\}$
Transiciones:	SHIFT	$\langle \Sigma, i\|B, A \rangle \Rightarrow \langle \Sigma\|i, B, A \rangle$
	REDUCE	$\langle \Sigma\|i, B, A \rangle \Rightarrow \langle \Sigma, B, A \rangle$
	LEFT-ARC	$\langle \Sigma\|i, j\|B, A \rangle \Rightarrow \langle \Sigma\|i, j\|B, A \cup \{(j, i)\} \rangle$ sólo si $\nexists k \mid (k, i) \in A$ (padre único) y $i \leftrightarrow^* j \notin A$ (aciclicidad).
	RIGHT-ARC	$\langle \Sigma\|i, j\|B, A \rangle \Rightarrow \langle \Sigma\|i, j\|B, A \cup \{(i, j)\} \rangle$ sólo si $\nexists k \mid (k, j) \in A$ (padre único) y $i \leftrightarrow^* j \notin A$ (aciclicidad).
2-Planar: Config. inicial/terminal:		$c_s(w_1 \dots w_n) = \langle [], [], [1 \dots n], \emptyset \rangle$, $C_f = \{\langle \Sigma_0, \Sigma_1, [], A \rangle \in C\}$
Transiciones:	SHIFT	$\langle \Sigma_0, \Sigma_1, i\|B, A \rangle \Rightarrow \langle \Sigma_0\|i, \Sigma_1\|i, B, A \rangle$
	REDUCE	$\langle \Sigma_0\|i, \Sigma_1, B, A \rangle \Rightarrow \langle \Sigma_0, \Sigma_1, B, A \rangle$
	LEFT-ARC	$\langle \Sigma_0\|i, \Sigma_1, j\|B, A \rangle \Rightarrow \langle \Sigma_0\|i, \Sigma_1, j\|B, A \cup \{j, i)\} \rangle$ sólo si $\nexists k \mid (k, i) \in A$ (padre único) y $i \leftrightarrow^* j \notin A$ (aciclicidad).
	RIGHT-ARC	$\langle \Sigma_0\|i, \Sigma_1, j\|B, A \rangle \Rightarrow \langle \Sigma_0\|i, \Sigma_1, j\|B, A \cup \{(i, j)\} \rangle$ sólo si $\nexists k \mid (k, j) \in A$ (padre único) y $i \leftrightarrow^* j \notin A$ (aciclicidad).
	SWITCH	$\langle \Sigma_0, \Sigma_1, B, A \rangle \Rightarrow \langle \Sigma_1, \Sigma_0, B, A \rangle$
Covington: Config. inicial/term.:		$c_s(w_1 \dots w_n) = \langle [], [], [1 \dots n], \emptyset \rangle$, $C_f = \{\langle \lambda_1, \lambda_2, [], A \rangle \in C\}$
Transiciones:	SHIFT	$\langle \lambda_1, \lambda_2, i\|B, A \rangle \Rightarrow \langle \lambda_1 \cdot \lambda_2\|i, [], B, A \rangle$
	NO-ARC	$\langle \lambda_1\|i, \lambda_2, B, A \rangle \Rightarrow \langle \lambda_1, i\|\lambda_2, B, A \rangle$
	LEFT-ARC	$\langle \lambda_1\|i, \lambda_2, j\|B, A \rangle \Rightarrow \langle \lambda_1, i\|\lambda_2, j\|B, A \cup \{(j, i)\} \rangle$ sólo si $\nexists k \mid (k, i) \in A$ (padre único) y $i \leftrightarrow^* j \notin A$ (aciclicidad).
	RIGHT-ARC	$\langle \lambda_1\|i, \lambda_2, j\|B, A \rangle \Rightarrow \langle \lambda_1, i\|\lambda_2, j\|B, A \cup \{(i, j)\} \rangle$ sólo si $\nexists k \mid (k, j) \in A$ (padre único) y $i \leftrightarrow^* j \notin A$ (aciclicidad).
b) Undirected Planar: Config. i/t:		$c_s(w_1 \dots w_n) = \langle [], [1 \dots n], \emptyset \rangle$, $C_f = \{\langle \Sigma, [], A \rangle \in C\}$
Transiciones:	SHIFT	$\langle \Sigma, i\|B, A \rangle \Rightarrow \langle \Sigma\|i, B, A \rangle$
	REDUCE	$\langle \Sigma\|i, B, A \rangle \Rightarrow \langle \Sigma, B, A \rangle$
	ARC	$\langle \Sigma\|i, j\|B, A \rangle \Rightarrow \langle \Sigma\|i, j\|B, A \cup \{\{i, j\}\} \rangle$ sólo si $i \leftrightarrow^* j \notin A$ (aciclicidad).
Undirected 2-Planar: Config. i/t:		$c_s(w_1 \dots w_n) = \langle [], [], [1 \dots n], \emptyset \rangle$, $C_f = \{\langle \Sigma_0, \Sigma_1, [], A \rangle \in C\}$
Transiciones:	SHIFT	$\langle \Sigma_0, \Sigma_1, i\|B, A \rangle \Rightarrow \langle \Sigma_0\|i, \Sigma_1\|i, B, A \rangle$
	REDUCE	$\langle \Sigma_0\|i, \Sigma_1, B, A \rangle \Rightarrow \langle \Sigma_0, \Sigma_1, B, A \rangle$
	ARC	$\langle \Sigma_0\|i, \Sigma_1, j\|B, A \rangle \Rightarrow \langle \Sigma_0\|i, \Sigma_1, j\|B, A \cup \{\{i, j\}\} \rangle$ sólo si $i \leftrightarrow^* j \notin A$ (aciclicidad).
	SWITCH	$\langle \Sigma_0, \Sigma_1, B, A \rangle \Rightarrow \langle \Sigma_1, \Sigma_0, B, A \rangle$
Undirected Covington: Config. i/t:		$c_s(w_1 \dots w_n) = \langle [], [], [1 \dots n], \emptyset \rangle$, $C_f = \{\langle \lambda_1, \lambda_2, [], A \rangle \in C\}$
Transiciones:	SHIFT	$\langle \lambda_1, \lambda_2, i\|B, A \rangle \Rightarrow \langle \lambda_1 \cdot \lambda_2\|i, [], B, A \rangle$
	NO-ARC	$\langle \lambda_1\|i, \lambda_2, B, A \rangle \Rightarrow \langle \lambda_1, i\|\lambda_2, B, A \rangle$
	ARC	$\langle \lambda_1\|i, \lambda_2, j\|B, A \rangle \Rightarrow \langle \lambda_1, i\|\lambda_2, j\|B, A \cup \{\{i, j\}\} \rangle$ sólo si $i \leftrightarrow^* j \notin A$ (aciclicidad).

Figura 2: Sistemas de transiciones: a) Planar, 2-Planar y Covington; b) Undirected Planar, Undirected 2-Planar y Undirected Covington. Cada configuración es de la forma $\langle \Sigma, B, A \rangle$ para analizadores planares, $\langle \Sigma_0, \Sigma_1, B, A \rangle$ para analizadores 2-planares y $\langle \lambda_1, \lambda_2, B, A \rangle$ para analizadores Covington, donde las estructuras Σ y λ son pilas y listas, respectivamente, que recogen las palabras parcialmente procesadas; B es el buffer que contiene la cadena de entrada y A recoge la parte del grafo construida por el analizador. Para analizar la cadena $w_1 \dots w_n$, el analizador empieza en la configuración inicial $c_s(w_1 \dots w_n)$ y ejecuta transiciones hasta alcanzar una configuración terminal.

co. Finalmente, las transiciones LEFT-ARC y RIGHT-ARC crean un enlace de dependencia hacia la izquierda o hacia la derecha, respectivamente, entre el nodo en la cabeza de la pila activa y el primer nodo del *buffer* para los analizadores Planar y 2-Planar, y entre el primer nodo de la primera lista y el primer nodo del *buffer* para el algoritmo Covington.

3. *Analizadores sintácticos no dirigidos*

A partir de los sistemas de transiciones definidos en la Sección 2.3 se han definido las versiones no dirigidas de la siguiente manera:

- Las configuraciones son modificadas de forma que el conjunto A se convierte en un conjunto de arcos no dirigidos, en lugar de arcos dirigidos.
- Las transiciones LEFT-ARC y RIGHT-ARC de cada sistema se sustituyen por una única transición ARC encargada de la creación de arcos no dirigidos.
- Las precondiciones de las transiciones se relajan puesto que ya no existe la noción de padre e hijo en los grafos no dirigidos, eliminando la restricción que garantiza un padre único para cada nodo.

Aplicando estas transformaciones obtenemos las variantes no dirigidas **Undirected Planar**, **Undirected 2-Planar** y **Undirected Covington** de los algoritmos Planar, 2-Planar y Covington como se muestra en la Figura 2b.

4. Técnicas de reconstrucción del bosque de dependencias

Los nuevos sistemas de transiciones presentados en la sección anterior generan grafos no dirigidos. Con el fin de obtener analizadores de dependencias completos que sean capaces de producir bosques de dependencias dirigidos, debemos aplicar un paso de reconstrucción que asigne la dirección a los arcos, de tal forma que se mantenga la restricción de padre único. Para ello se han considerado dos técnicas diferentes que pueden ser utilizadas para la reconstrucción de árboles de dependencia a partir de grafos no dirigidos: una técnica basada en la raíz (**root-based**) y otra basada en las etiquetas (**label-based**).

4.1. Reconstrucción root-based

La técnica *root-based* utiliza el nodo raíz artificial para asignar la dirección de los arcos. Para conseguir esto, los analizadores deben entrenarse para construir explícitamente arcos no dirigidos desde el nodo raíz artificial al nodo raíz propio de cada oración utilizando transiciones ARC. De este modo, tras la ejecución del analizador, se recorre el grafo no dirigido resultante partiendo del nodo raíz artificial y finalizando en los nodos hoja. De este modo, a medida que se recorre el grafo, se va asignando la dirección de los arcos de acuerdo al sentido del camino seguido desde el nodo raíz.

Aunque este método tiene la ventaja de no añadir ninguna complicación al proceso de análisis, tiene el inconveniente de que, puesto que la dirección de todos los arcos en el grafo de salida dependen de cuál sea el nodo elegido como raíz, una elección errónea de éste puede provocar que muchos arcos de dependencia sean erróneos.

4.2. Reconstrucción label-based

La técnica *label-based* consiste en utilizar las etiquetas para codificar la dirección de los arcos de dependencia. Para cada etiqueta existente X en el conjunto de entrenamiento, se crean dos etiquetas X_l y X_r, que significan "un arco hacia la izquierda con etiqueta X" y "un arco hacia la derecha con etiqueta X", respectivamente. Esto permite codificar los grafos de dependencias con n posibles etiquetas como grafos no dirigidos con $2n$ posibles etiquetas. Así, trabajando con el analizador entrenado con estas nuevas etiquetas, podrán ser utilizadas para recuperar la dirección de los arcos del grafo no dirigido resultante.

Aunque esta técnica soluciona los inconvenientes que presentaba la reconstrucción *root-based*, el problema de este enfoque es que no garantiza que los grafos generados sean bosques, ya que los errores de etiquetado pueden dar lugar a nodos con más de un padre. Por lo tanto, esta técnica requiere un paso posterior adicional que solucione estos conflictos. Éste puede ser implementado de diferentes formas dependiendo del criterio que utilicemos para solucionar los conflictos de múltiples padres. Por ejemplo, si el analizador no dirigido genera la salida presente en la Figura 3a, ésta será decodificada obteniendo el grafo dirigido de la Figura 3b; el cual no es un bosque puesto que el nodo 1 tiene dos padres. Dependiendo de la heurística utilizada, se puede elegir 0 como padre de 1 (y entonces revertir la dirección del arco de 2 a 1, suponiendo que se ha producido un error de etiquetado, o simplemente enlazar 2 al nodo raíz artificial 0, ya que no hemos encontrado un padre para este nodo), o se puede elegir 2 como el padre de 1 (enlazando 2 al nodo raíz artificial, obteniendo el bosque de la Figura 3c).

En la práctica, se ha encontrado que la siguiente heurística produce buenos resultados:

1. Enlazar al nodo raíz artificial todo aquel nodo sin padre presente en el grafo de dependencias. Por ejemplo, aplicando esto al grafo de la Figura 3b, se creará en enlace de 0 a 2.
2. Elegir un nodo k con más de un padre. Sean $h_1, \ldots, h_m$ los padres asignados al nodo k. Considérese el camino desde la raíz a cada

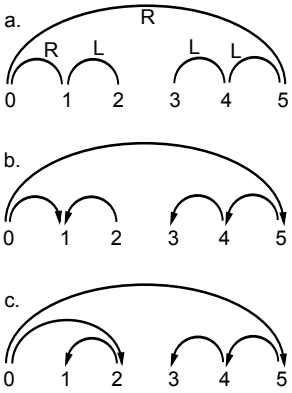

Figura 3: a) Un grafo no dirigido obtenido por un analizador con reconstrucción *label-based*, b) El grafo de dependencias obtenido tras la decodificación de las etiquetas, c) El grafo resultante tras aplicar la heurística que soluciona los conflictos de múltiples padres como se detalla en la Sección 4.2.

uno de los nodos $h_1, \ldots, h_m$. Elegir el camino más corto (si hay un empate, este se rompe eligiendo el camino asociado al padre situado más hacia la izquierda). Borrar el primer arco del camino seleccionado (el arco que parte del nodo raíz) y revertir la dirección del resto de arcos pertenecientes al camino elegido. En el ejemplo, se eliminaría el enlace de 0 a 1, obteniendo el bosque detallado en la Figura 3c.

3. Si el grafo todavía no es un bosque, volver al paso 2.

5. Experimentos y resultados

En esta sección, se evalúa el rendimiento de los analizadores desarrollados sobre ocho bancos de árboles utilizados en el *CoNLL-X shared task* (Buchholz y Marsi, 2006).

Las tablas 1, 2 y 3 comparan la precisión de las versiones no dirigidas con las técnicas de reconstrucción *root-based* y *label-based* con las versiones dirigidas de los analizadores Planar, 2-Planar y Covington, respectivamente. A mayores, se incluye una comparación con dos algoritmos del estado del arte en este tipo de sistemas: el analizador Planar se compara con el analizador proyectivo Arc-eager de Nivre (2003); y el analizador 2-Planar es comparado con el analizador Arc-eager pseudo-proyectivo de Nivre y Nilsson (2005), capaz de manejar dependencias no planares.

En los experimentos se han utilizado los clasificadores SVM proporcionados por el paquete LIBSVM (Chang y Lin, 2001) para todos los idiomas excepto el chino, checo y alemán, donde se usó el paquete LIBLINEAR (Fan et al., 2008), que reduce el tiempo necesario para el entrenamiento con estos grandes bancos de árboles.

Los resultados muestran que el uso del análisis no dirigido con reconstrucción *label-based* claramente mejora el rendimiento en la mayoría de bancos de árboles para los algoritmos Undirected Planar y Undirected Covington, donde en muchos casos incluso superaran a los analizadores Arc-eager proyectivo y pseudo-proyectivo. En el caso del analizador Undirected 2-Planar los resultados son menos concluyentes, con mejoras sobre las versiones dirigidas en cuatro de ocho idiomas.

Las mejoras obtenidas en LAS con reconstrucción *label-based* sobre las versiones dirigidas son estadísticamente significativas al nivel .05 (test de McNemar) para danés, alemán y portugués en el caso del analizador Undirected Planar; y checo, danés y turco para el analizador Undirected Covington. Además, no fue detectada una reducción estadísticamente significativa de precisión en ningún caso.

En los experimentos realizados se puede apreciar que, en todos los casos, los analizadores con reconstrucción *root-based* tienen un rendimiento inferior a aquellos que usan la técnica *label-based*.

6. Conclusiones

En este artículo se han presentado nuevas variantes de los analizadores Planar y 2-Planar de Gómez-Rodríguez y Nivre (2010) y Covington no proyectivo (Covington, 2001; Nivre, 2008) que ignoran la dirección de los arcos de dependencia, y técnicas de reconstrucción que pueden ser utilizadas para recuperar la dirección de los arcos producidos. Los resultados obtenidos muestran que la idea de un análisis no dirigido, junto con la reconstrucción *label-based* de la Sección 4.2, produce mejoras en la precisión del análisis en la mayoría de las combinaciones idioma/algoritmo probadas, y, además, es capaz de superar el rendimiento de analizadores basados en transiciones del estado del arte.

Las mejoras de precisión conseguidas indican que, al relajar la restricción de padre único, se mitiga la propagación de errores presentes en la etapa de análisis. Además, se ha observado empíricamente que las diferencias entre el LAS no dirigido obtenido a partir del grafo no dirigido previo a la reconstrucción y el LAS dirigido final es habitualmente inferior

	Planar		UPlanarR		UPlanarL		MaltP	
Idi.	LAS(p)	UAS(p)	LAS(p)	UAS(p)	LAS(p)	UAS(p)	LAS(p)	UAS(p)
AR	66.93(67.34)	77.56(77.22)	65.91(66.33)	77.03(76.75)	**67.01**(**67.50**)	**77.74**(**77.57**)	66.43(66.74)	77.19(76.83)
CN	84.23(84.20)	88.37(88.33)	83.14(83.10)	87.00(86.95)	84.51(84.50)	88.37(88.35)	**86.42**(**86.39**)	**90.06**(**90.02**)
CZ	77.24(77.70)	83.46(83.24)	75.08(75.60)	81.14(81.14)	**77.60**(**77.93**)	**83.56**(**83.41**)	77.24(77.57)	83.40(83.19)
DA	83.31(82.60)	88.02(86.64)	82.65(82.45)	87.58(86.67)	**83.87**(**83.83**)	**88.94**(**88.17**)	83.31(82.64)	88.30(86.91)
GE	84.66(83.60)	87.02(85.67)	83.33(82.77)	85.78(84.93)	**86.32**(**85.67**)	**88.62**(**87.69**)	86.12(85.48)	88.52(87.58)
PO	86.22(83.82)	89.80(86.88)	85.89(83.82)	89.68(87.06)	86.52(**84.83**)	**90.28**(**88.03**)	**86.60**(84.66)	90.20(87.73)
SW	**83.01**(82.44)	88.53(87.36)	81.20(81.10)	86.50(85.86)	82.95(**82.66**)	88.29(87.45)	82.89(82.44)	**88.61**(**87.55**)
TU	62.70(71.27)	73.67(78.57)	59.83(68.31)	70.15(75.17)	**63.27**(**71.63**)	**73.93**(**78.72**)	62.58(70.96)	73.09(77.95)

Cuadro 1: Precisión del analizador Undirected Planar con reconstrucción *root-based* (UPlanarR) y *label-based* (UPlanarL) en comparación con los algoritmos dirigidos Planar (Planar) y Arc-eager proyectivo implementado en el MaltParser (MaltP), sobre ocho bancos de árboles del *CoNLL-X shared task* (Buchholz y Marsi, 2006): Árabe (AR) (Hajič et al., 2004), chino (CN) (Chen et al., 2003), checo (CZ) (Hajič et al., 2006), danés (DA) (Kromann, 2003), alemán (GE) (Brants et al., 2002), portugués (PO) (Afonso et al., 2002), sueco (SW) (Nilsson, Hall, y Nivre, 2005) y turco (TU) (Oflazer et al., 2003; Atalay, Oflazer, y Say, 2003). Se muestran las medidas de precisión *labelled attachment score* (LAS) y *unlabelled attachment score* (UAS) excluyendo e incluyendo los *tokens* de puntuación en los resultados (este último entre paréntesis). Se marcan en negrita los mejores resultados para cada idioma.

	2Planar		U2PlanarR		U2PlanarL		MaltPP	
Idi.	LAS(p)	UAS(p)	LAS(p)	UAS(p)	LAS(p)	UAS(p)	LAS(p)	UAS(p)
AR	**66.73**(**67.19**)	**77.33**(**77.11**)	66.37(66.93)	77.15(77.09)	66.13(66.52)	76.97(76.70)	65.93(66.02)	76.79(76.14)
CN	84.35(84.32)	88.31(88.27)	83.02(82.98)	86.86(86.81)	84.00(83.94)	87.83(87.75)	**86.42**(**86.39**)	**90.06**(**90.02**)
CZ	77.72(77.91)	83.76(83.32)	74.44(75.19)	80.68(80.80)	78.00(**78.59**)	84.22(**84.21**)	**78.86**(78.47)	**84.54**(83.89)
DA	**83.81**(83.61)	88.50(87.63)	82.00(81.63)	86.87(85.80)	83.75(**83.65**)	**88.62**(**87.82**)	83.67(83.54)	88.52(87.70)
GE	86.28(85.76)	88.68(87.86)	82.93(82.53)	85.52(84.81)	86.52(85.99)	88.72(87.92)	**86.94**(**86.62**)	**89.30**(**88.69**)
PO	87.04(**84.92**)	**90.82**(**88.14**)	85.61(83.45)	89.36(86.65)	86.70(84.75)	90.38(87.88)	**87.08**(84.90)	90.66(87.95)
SW	83.13(**82.71**)	88.57(**87.59**)	81.00(80.71)	86.54(85.68)	82.59(82.25)	88.19(87.29)	**83.39**(82.67)	**88.59**(87.38)
TU	61.80(70.09)	72.75(77.39)	58.10(67.44)	68.03(74.06)	61.92(70.64)	72.18(77.46)	**62.80**(**71.33**)	**73.49**(**78.44**)

Cuadro 2: Precisión del analizador Undirected 2-Planar con reconstrucción *root-based* (U2PlanarR) y *label-based* (U2PlanarL) en comparación con los algoritmos dirigidos 2-Planar (2Planar) y Arc-eager pseudo-proyectivo implementado en el MaltParser (MaltPP). El significado de los resultados es el mismo que en los presentados en el Cuadro 1.

a 0,20 %. Esto se cumple para ambas reconstrucciones, indicando que las diferencias de precisión entre ellas vienen principalmente de las diferencias en el entrenamiento más que de los técnicas en sí.

Las simplificaciones no dirigidas podrían aplicarse a cualquier otro analizador basado en transiciones que disponga de transiciones Left-Arc y Right-Arc simétricas. Por otra parte, las técnicas de reconstrucción podrían aplicarse individualmente a cualquier analizador de dependencias (basado en transiciones o no).

La idea de un análisis sintáctico con relaciones no dirigidas entre palabras ha sido aplicada antes en el trabajo sobre *Link Grammar* (Sleator y Temperley, 1991), pero en ese caso el mismo formalismo trabajaba con grafos no dirigidos, los cuales eran la salida final del analizador. Por lo que sabemos, la idea de usar grafos no dirigidos como un paso intermedio hacia la obtención de estructuras de dependencias no ha sido explorada antes.

Bibliografía

Afonso, Susana, Eckhard Bick, Renato Haber, y Diana Santos. 2002. "Floresta sintá(c)tica": a treebank for Portuguese. En *Actas de LREC 2002*, páginas 1968–1703, Paris, France. ELRA.

Atalay, Nart B., Kemal Oflazer, y Bilge Say. 2003. The annotation process in the Turkish treebank. En *Actas de LINC 2003*, páginas 243–246. ACL.

Brants, Sabine, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, y George Smith. 2002. The tiger treebank. En *Actas de TLT 2002, September 20-21*, Sozopol, Bulgaria.

Buchholz, Sabine y Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. En *Actas de CoNLL 2006*, páginas 149–164.

	Covington		UCovingtonR		UCovingtonL	
Idi.	LAS(p)	UAS(p)	LAS(p)	UAS(p)	LAS(p)	UAS(p)
AR	65.17(65.49)	75.99(**75.69**)	63.49(63.93)	74.41(74.20)	**65.61**(**65.81**)	**76.11**(75.66)
CN	85.61(85.61)	89.64(89.62)	84.12(84.02)	87.85(87.73)	**86.28**(**86.17**)	**90.16**(**90.04**)
CZ	78.26(77.43)	84.04(83.15)	74.02(74.78)	79.80(79.92)	**78.42**(**78.69**)	**84.50**(**84.16**)
DA	83.63(82.89)	88.50(87.06)	82.00(81.61)	86.55(85.51)	**84.27** (**83.85**)	**88.82** (**87.75**)
GE	**86.70**(85.69)	**89.08**(87.78)	84.03(83.51)	86.16(85.39)	86.50(**85.90**)	88.84(**87.95**)
PO.	84.73(82.56)	89.10(86.30)	83.83(81.71)	87.88(85.17)	**84.95**(**82.70**)	**89.18**(**86.31**)
SW	**83.53**(**82.76**)	**88.91**(**87.61**)	81.78(81.47)	86.78(85.96)	83.09(82.73)	88.11(87.23)
TU	64.25(72.70)	74.85(79.75)	63.51(72.08)	74.07(79.10)	**64.91**(**73.38**)	**75.46**(**80.40**)

Cuadro 3: Precisión del analizador Undirected Covington con reconstrucción *root-based* (UCovingtonR) y *label-based* (UCovingtonL) en comparación con el algoritmo dirigido (Covington). El significado de los resultados es el mismo que en los presentados en el Cuadro 1.

Chang, Chih-Chung y Chih-Jen Lin, 2001. *LIBSVM: A Library for Support Vector Machines*. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

Chen, K., C. Luo, M. Chang, F. Chen, C. Chen, C. Huang, y Z. Gao. 2003. Sinica treebank: Design criteria, representational issues and implementation. En Anne Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*. Kluwer, capítulo 13, páginas 231–248.

Covington, Michael A. 2001. A fundamental algorithm for dependency parsing. En *Actas de 39th Annual ACM Southeast Conference*, páginas 95–102.

Fan, R.-E., K.-W. Chang, C.-J. Hsieh, X.-R. Wang, y C.-J. Lin. 2008. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

Gómez-Rodríguez, Carlos y Joakim Nivre. 2010. A transition-based parser for 2-planar dependency structures. En *Actas de ACL 2010*, páginas 1492–1501. ACL.

Hajič, Jan, Jarmila Panevová, Eva Hajičová, Jarmila Panevová, Petr Sgall, Petr Pajas, Jan Štěpánek, Jiří Havelka, y Marie Mikulová. 2006. Prague Dependency Treebank 2.0. CDROM CAT: LDC2006T01, ISBN 1-58563-370-4. Linguistic Data Consortium.

Hajič, Jan, Otakar Smrž, Petr Zemánek, Jan Šnaidauf, y Emanuel Beška. 2004. Prague Arabic Dependency Treebank: Development in data and tools. En *Actas de NEMLAR*.

Huang, Liang y Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. En *Actas de ACL 2010*, páginas 1077–1086. ACL.

Kromann, Matthias T. 2003. The Danish dependency treebank and the underlying linguistic theory. En *TLT 2003*, páginas 217–220, Växjö, Sweden. Växjö University Press.

Martins, Andre, Noah Smith, y Eric Xing. 2009. Concise integer linear programming formulations for dependency parsing. En *Actas de ACL 2009*, páginas 342–350.

McDonald, Ryan y Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. En *Actas de EMNLP-CoNLL 2007*, páginas 122–131.

McDonald, Ryan, Fernando Pereira, Kiril Ribarov, y Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. En *Actas de HLT/EMNLP 2005*, páginas 523–530.

Nilsson, Jens, Johan Hall, y Joakim Nivre. 2005. MAMBA meets TIGER: Reconstructing a Swedish treebank from Antiquity. En Peter Juel Henrichsen, editor, *Proceedings of the NODALIDA Special Session on Treebanks*.

Nivre, Joakim. 2003. An efficient algorithm for projective dependency parsing. En *Actas de IWPT 2003*, páginas 149–160.

Nivre, Joakim. 2008. Algorithms for Deterministic Incremental Dependency Parsing. *Computational Linguistics*, 34(4):513–553.

Nivre, Joakim, Johan Hall, y Jens Nilsson. 2004. Memory-based dependency parsing. En *Actas de CoNLL 2004*, páginas 49–56. ACL.

Nivre, Joakim y Jens Nilsson. 2005. Pseudo-projective dependency parsing. En *Actas de ACL 2005*, páginas 99–106.

Oflazer, Kemal, Bilge Say, Dilek Zeynep Hakkani-Tür, y Gökhan Tür. 2003. Building a Turkish treebank. En Anne Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*. Kluwer, páginas 261–277.

Sleator, Daniel y Davy Temperley. 1991. Parsing English with a link grammar. Informe Técnico CMU-CS-91-196, Carnegie Mellon University, Computer Science.

Titov, Ivan y James Henderson. 2007. A latent variable model for generative dependency parsing. En *Actas de IWPT 2007*, páginas 144–155.