UNIVERSIDADE DA CORUÑA
Departamento de Computación

TESIS DOCTORAL

RECONOCIMIENTO DE PATRONES EN BOSQUES COMPARTIDOS

AUTOR: Francisco José Ribadas Pena
DIRECTOR: Dr. Manuel Vilares Ferro

A Coruña, Noviembre de 2.002

Tesis Doctoral: Reconocimiento de Patrones en Bosques Compartidos
Autor: Francisco José Ribadas Pena
Director: Manuel Vilares Ferro
Fecha: 29 de noviembre de 2.002

Tribunal

Presidente : Dr. José Luis Freire Nistal

Vocal 1º : Dr. Antonio Bahamonde Rionda

Vocal 2º : Dr. Richard F.E. Sutcliffe

Vocal 3º : Dr. Jean-Cédric Chappelier

Secretario : Dr. Alejandro Sobrino Cerdeiriña

Calificación : _____

Agradecimientos

Deseo expresar mi más sincero agradecimiento a las todas las personas que me han ayudado y apoyado durante la realización de esta tesis. En primer lugar, deseo agradecer a mi director de tesis, Manuel Vilares Ferro, que me ha guiado y aconsejado durante el desarrollo de este trabajo. También, agradezco la ayuda prestada por todos mis compañeros del grupo CoLe[a], Víctor Darriba, David Cabrero, Jesús Vilares, Miguel Alonso, Jorge Graña y Leandro Rodríguez.

Y por último, aunque no menos importante, deseo recordar y agradecer su apoyo a mis padres y hermanos.

Este trabajo ha sido parcialmente financiado por el proyecto 1FD97-0047-C04-02 del programa FEDER de la Unión Europea, los proyectos PGIDT99XI10502B y PGIDT01PXI10506PN de la Xunta de Galicia, y por el Ministerio de Ciencia y Tecnología a través del proyecto TIC2000-0370-C02-01.

[a] Compiladores y Lenguajes (`http://www.grupocole.org`).

Resumen

"Reconocimiento de Patrones en Bosques Compartidos"

El uso de árboles como formalismo de representación es una técnica empleada comúnmente en multitud de campos de la informática. Podemos encontrar ejemplos de su uso como estructuras de datos básicas en los compiladores y optimizadores de código, tanto para lenguajes procedimentales, como para lenguaje lógicos o funcionales. También se emplean árboles en la representación de estructuras moleculares, en los sistemas de gestión de documentos estructurados o en la representación de sentencias de lenguajes naturales. Por esta razón, un problema de especial relevancia, es el del reconocimiento de patrones en estructuras arborescentes. El objetivo de este tipo de técnicas es el de localizar una estructura de interés, expresada en forma de patrón, dentro de un conjunto de estructuras mayores. De este modo, el reconocimiento de patrones en árboles ofrece un formalismo descriptivo extremadamente útil para la interrogación y el acceso a datos estructurados.

El trabajo que presentamos en esta tesis pretende extender las técnicas clásicas de reconocimiento de patrones sobre árboles, basadas en el concepto de *distancia de edición*, al caso de los bosques compartidos. Nuestro trabajo se enmarca en el campo de la aplicación de técnicas avanzadas de reconocimiento de patrones en sistemas de recuperación y extracción de información. En concreto, en el estudio de las posibilidades del uso de estructuras sintácticas como elementos clave para la descripción y el acceso a los documentos relevantes para las consultas de los usuarios.

En el caso del análisis y el procesamiento del lenguaje natural, debemos enfrentarnos al problema de la ambigüedad inherente a este tipo de lenguajes. Dicha ambigüedad origina que una misma sentencia pueda tener más de una estructura sintáctica asociada. En nuestro trabajo hemos empleado el generador de analizadores ICE, que proporciona una representación compacta de los múltiples análisis de una sentencia en forma de bosque de análisis compartido. Por lo tanto, nos hemos centrado en la adaptación de los algoritmo clásicos de reconocimiento de patrones sobre árboles al caso de los bosques de análisis sintáctico compartidos generados por ICE.

Para llevar a cabo esta adaptación hemos realizado un estudio detallado de los algoritmos clásicos, centrándonos, tanto en sus aspectos computacionales, como en su capacidad expresiva. Como resultado de esta evaluación, hemos elegido las propuestas de Zhang y Shasha como base para nuestro trabajo, por ser las más adecuadas para nuestros propósitos. A la hora de efectuar la integración del algoritmo de Zhang y Shasha en ICE, se ha llevado a cabo un estudio del tipo de representación sintáctica utilizada, identificado los factores que determinan la compartición de estructuras y cómo afectan al mecanismo de reconocimiento de patrones. Esto nos ha permitido modificar la propuesta original de Zhang y Shasha para adaptarla a las peculiaridades de las estructuras sintácticas generadas por ICE, consiguiendo un aprovechamiento máximo de la compartición de estructuras, que evita la realización de cálculos redundantes.

Finalmente, hemos evaluado las ventajas aportadas por nuestra aproximación en cuanto a la mejora del rendimiento. Para ello hemos realizado una serie de experimentos empleando un conjunto de gramáticas no deterministas. Estos resultados experimentales muestran una importante reducción del coste computacional y confirman la adecuación de nuestras propuestas al tratamiento de sentencias altamente ambiguas. Ello nos ofrece un marco prometedor de cara a la posible utilización práctica de estas técnicas de reconocimiento de patrones en cualquier tipo de aplicación donde se precise el procesamiento eficaz de árboles con compartición. En concreto, creemos que nuestra propuesta puede servir como mecanismo básico para la mejora de las tareas de indexación y consulta en sistemas de recuperación y extracción de información. Aunque, eso sí, dentro de entornos acotados, puesto que el coste computacional sigue siendo excesivamente alto, lo que penaliza la utilización práctica de estas aproximaciones en aplicaciones no restringidas.

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Parte I

Preliminares

Capítulo 1
Introducción

Esta tesis se centra en el estudio de las técnicas de reconocimiento de patrones en árboles y de las posibilidades de extender dichas técnicas para manejar eficientemente bosques de análisis sintáctico compartidos. Los árboles son uno de los formalismos más potentes y más utilizados para estructurar datos en multitud de entornos, siendo especialmente útiles a la hora de manejar cualquier tipo de información que posea una estructura jerárquica. Como formalismo de representación, los árboles y otras estructuras similares como los grafos acíclicos, han venido siendo empleados para representar sentencias en lenguajes naturales como el español o el inglés, para representar programas escritos en lenguajes de programación, en aplicaciones de procesamiento de imágenes o para representar estructuras moleculares, por citar sólo algunas de sus muchas aplicaciones.

Por otra parte, el reconocimiento de patrones se puede definir como el proceso de encontrar una subestructura que está incluida dentro de otra mayor, mediante la comparación de ésta con una representación de aquella en forma de patrón. El reconocimiento de patrones ofrece un modo natural para interrogar de forma descriptiva un conjunto de estructuras complejas, que en el caso que nos ocupa tendrán la forma de árboles o bosques. El uso de este tipo de técnicas hace posible que en la interrogación el usuario se limite a describir como deben ser los resultados de la búsqueda sin tener que expresar el modo en el que éstos deberían ser encontrados. Esto hace de las técnicas de reconocimiento de patrones un marco prometedor para muchos sistemas de gestión y localización de información en donde la estructura de los elementos almacenados sea relevante.

De este modo, el reconocimiento de patrones en estructuras arbóreas constituye un problema con multitud de aplicaciones prácticas interesantes [39]. Se han empleado con éxito técnicas de este tipo para la optimización de programas informáticos [2] y para la comparación y tratamiento de imágenes [21, 22]. En cuanto al uso del reconocimiento de patrones en árboles como mecanismo de interrogación en bases de datos estructuradas, cabe destacar los trabajos en el campo de localización y procesamiento de estructuras moleculares, como proteínas o secuencias de ácido ribonucléico (ARN) [38]. También se ha empleado este tipo de técnicas para la interrogación de bases de datos documentales basada en la estructura de dichos documentos [20, 8, 62]. En general dichas bases de datos estarán compuestas por colecciones de documentos estructurados, o bien podrán ser interpretadas como grupos de sentencias en un determinado lenguaje natural con una determinada estructura sintáctica.

Desde el punto de vista formal, el reconocimiento de patrones en árboles [13, 24] es una evolución de su homónimo en cadenas de caracteres [30, 34, 32]. Este es un problema que ha sido ampliamente estudiado, principalmente motivado por aplicaciones en el campo de las técnicas de *recuperación de información* (RI) [3] y, más recientemente, en el campo del procesamiento de secuencias de ácido desoxirribonucléico (ADN) [26].

Sin embargo, la cantidad de trabajos en el campo de los algoritmos de reconocimiento de árboles es bastante más reducida que en el caso del de cadenas, debido principalmente a la propia naturaleza del problema. Como resultado, la aparición de propuestas novedosas y de avances en relación a la mejora de sus eficiencia han sido limitados. Como veremos en esta memoria, la comparación de árboles es un problema mucho más costoso computacionalmente que la comparación de cadenas, como, por otra parte,

parece obvio. Esto tiene como consecuencia que el uso del reconocimiento de patrones sobre árboles vea frenada su utilización como mecanismo básico en algunas aplicaciones prácticas, debido a que el coste computacional que implicaría sería inadmisible en el actual estado de la técnica. En esos casos se suelen emplear estrategias de reconocimiento de patrones sobre cadenas [32], que si bien son menos poderosas semánticamente, resultan más eficientes computacionalmente.

Nuestro trabajo se centra en extender los algoritmos clásicos de reconocimiento de patrones en árboles [37, 43, 65, 66] para que permitan el procesamiento de bosques compartidos [56, 6, 28]. Estos no son más que colecciones de árboles en las cuales las subestructuras comunes, compartidas entre dos o más de esos árboles, están representadas una única vez. Típicamente, este tipo de bosques se obtendrán como resultado del análisis sintáctico de cadenas ambiguas, de modo que los distintos análisis de una misma cadena puedan tener partes comunes.

Presentaremos a continuación, más detalladamente, las motivaciones de nuestro trabajo y los principales objetivos de la tesis. Terminaremos este capítulo introductorio describiendo la estructura de la memoria y el contenido de los distintos capítulos que la componen.

1.1. Motivación y objetivos

Como hemos adelantado, el reconocimiento de patrones en árboles es un problema de interés para diversas áreas, como las ya citadas de la optimización de programas y la consulta de bases de datos estructuradas. Pretendemos extender las técnicas clásicas de reconocimiento de patrones sobre árboles [37, 43, 65] al caso de bosques de árboles compartidos. En concreto, al tipo de bosques construidos por los analizadores sintácticos generados por el sistema ICE[1] [56]. Este es un sistema de generación de analizadores sintácticos incrementales para gramáticas independientes del contexto sin restricciones. La salida de dichos analizadores tiene la forma de un grafo Y-O que representa de forma compacta un bosque compartido [6, 28] que contiene todos los posibles árboles de análisis que se pueden construir para una cadena de entrada dada. Nuestra propuesta es, por otro lado, generalizable a cualquier tipo de bosque generado por un analizador del tipo salto-reducción.

Nuestro trabajo se ha desarrollado en el contexto de la investigación que viene realizando el grupo COLE[2] sobre el uso de técnicas de *procesamiento del lenguaje natural* (PLN) en el campo de los sistemas de recuperación y extracción de información. En concreto, esta tesis está relacionada con el estudio de las posibilidades de aplicación de estructuras sintácticas para mejorar el rendimiento de determinados aspectos de estos sistemas, como pueden ser la consulta [42] o la indexación [11, 16]. Los sistemas de RI clásicos [3] suelen estar basados en el concepto de palabra clave. En estos entornos, una palabra clave, también denominada término de indexación, está formada por una o más palabras extraídas a partir de los documentos que componen la base de datos documental. La idea básica de esta aproximación es que el conjunto de palabras clave será capaz de representar de forma adecuada la semántica del documento del cual fueron extraídas.

Sin embargo, en muchos casos, para encontrar la información de interés, es el modo en que se combinan dichas palabras junto con sus implicaciones semánticas lo que realmente determina la auténtica relevancia de un documento respecto a una consulta. Esto lleva a plantear la necesidad de utilizar representaciones más sofisticadas de los contenidos referidos. Diferentes autores, como Fagan [11], Smeaton [41, 42], Jacquemin [16] y otros [18, 31], han planteado la posibilidad de utilizar las estructuras sintácticas extraídas mediante un proceso de análisis sintáctico de las frases que componen un documento para mejorar el proceso de indexación y/o consulta de los sistemas de RI.

Dentro del marco descrito, nuestra aportación está orientada al desarrollo de técnicas eficientes para el reconocimiento de patrones sobre bosques de árboles altamente ambiguos [51, 49]. En efecto, uno de los problemas típicos que se presentan al emplear técnicas de PLN, en concreto el análisis sintáctico, es la ambigüedad en el proceso de análisis de las sentencias escritas en los lenguajes naturales. Dicha

[1] Por *Incremental Context-Free Environment*.
[2] `http://www.grupocole.org`.

ambigüedad es, por un lado, inherente a la mayoría de los lenguajes de comunicación humana y, por otro; derivada de la falta de cobertura gramatical del lenguaje analizado. Es por ello necesario disponer de un analizador sintáctico que sea capaz de identificar de forma eficiente todos los análisis posibles[3] y representarlos de modo compacto [6] para fases posteriores de análisis semántico. En nuestro caso hemos utilizado el sistema ICE [56], que genera grafos Y-O para representar los bosques de salida. Nos hemos centrado en estudiar los mecanismos que originan la compartición de estructuras en ICE y en como sacar provecho de esa compartición para evitar realizar operaciones de reconocimiento redundantes.

La aproximación clásica al problema del reconocimiento de patrones en árboles se basa en el concepto de *distancia de edición* [37, 43]. La idea básica es cuantificar la similaridad entre dos estructuras tomando como medida el coste de la transformación de una de ellas en la otra. Para ello se define un conjunto de operaciones de edición, que permiten modificar los elementos de una estructura, bien una cadena o bien un árbol, y se asocia a cada una de ellas un coste numérico. En el caso que nos ocupa, utilizaremos esta misma aproximación, si bien la aplicaremos sobre un bosque compartido resultado del análisis de una sentencia en lenguaje natural, generalmente ambigua, generado por el sistema ICE. Por lo tanto, nuestro objetivo será trasladar la aproximación clásica del cálculo de distancias de edición a este tipo de bosques, sacando el mayor partido posible de la compartición de estructuras ofrecida por ICE para evitar repetir cálculos innecesarios.

De cara a la aplicación práctica de estas técnicas, es también interesante poder manejar un mecanismo de reconocimiento de patrones lo más flexible posible. En muchos casos, el usuario, o bien desconoce detalles sobre la representación arbórea utilizada, o bien, existen ciertos detalles que no le son relevantes. Por ello, es conveniente manejar mecanismos de reconocimiento aproximado, que permitan una especificación vaga de las consultas. De ese modo se libera al usuario de tener que indicar todos los detalles. Será el sistema el que deba decidir cuales son las respuestas más próximas a esa descripción aproximada de la necesidad de información del usuario. En nuestro caso hemos optado por dar soporte a la inclusión de símbolos VLDC[4] [66, 63] que permiten especificar de forma aproximada los árboles patrón que deseamos buscar. De este modo, nuestro trabajo ha estado dirigido a la consecución de los objetivos que se enumeran a continuación:

- *Estudio y evaluación de los algoritmos clásicos de reconocimiento de patrones en árboles.* Hemos revisado las distintas aproximaciones al problema del reconocimiento de patrones en árboles [37, 43, 65], fijándonos tanto en la eficiencia computacional de las soluciones propuestas como en sus capacidades en cuanto a potencia expresiva.

- *Adaptación e integración de dichos algoritmos en el entorno* ICE. Revisados los algoritmos clásicos, nos hemos centrado en adaptar las propuestas de Zhang y Shasha [65, 66] al entorno del analizador ICE [56]. Esto ha supuesto estudiar el mecanismo de generación de representaciones sintácticas [6] de ICE e identificar los factores que originan la compartición de estructuras sintácticas.

- *Aprovechamiento de la compartición de estructuras para evitar redundancias de cálculos.* Una vez aislados los aspectos que afectan a la compartición de estructuras, siguiendo la aproximación presentada en [51, 49], hemos adaptado las propuestas de Zhang y Shasha para que se integren de forma eficiente en el entorno ICE. Dicha adaptación tratará de aprovechar al máximo las posibilidades de compartición de estructuras sintácticas que ofrece este entorno de análisis sintáctico, para evitar la repetición de cálculos redundantes.

- *Evaluación de las posibles aplicaciones prácticas del sistema propuesto.* Por último, hemos evaluado las modificaciones del algoritmo básico propuestas, tanto desde el punto de vista de la eficiencia computacional como desde sus posibilidades de utilización por parte del usuario.

[3] En el algunos casos el número de análisis posibles puede ser exponencial o incluso infinito.
[4] Por *Variable Length Don't Care*.

1.2. Estructura de la memoria

Finalizamos el capítulo presentado como se estructuran los distintas partes de esta memoria y esbozando el contenido de los capítulos que la componen, en relación a los objetivos antes descritos. Consideramos cuatro parte, que pasamos a describir:

- La **parte I** comienza con el presente capítulo introductorio, donde se introduce la motivación y objetivos de esta tesis, junto con una descripción general de la estructura de la memoria.

  - El **capítulo 2**, incluido en esta primera parte, contiene un repaso de los conceptos y notaciones empleados a lo largo de la memoria. Se divide en dos partes, en la primera de ellas se examinan los conceptos básicos relacionados con los lenguajes, cadenas y gramáticas. En la segunda parte se repasan los conceptos relativos a grafos y árboles, prestando especial atención al uso de estas estructuras como formalismos de representación de relaciones sintácticas.

- La **parte II** está dedicada al estudio del problema del reconocimiento de patrones en árboles, y a la descripción y evaluación de las aproximaciones clásicas.

  - En el **capítulo 3** se introduce el problema del reconocimiento de patrones en cadenas que, como veremos, está íntimamente relacionado con su homónimo en árboles. Ambos se pueden formular como problemas de cálculo de distancias y, en ambos casos, se sigue la aproximación de medir esa distancia como el coste de una transformación. Nos limitaremos a presentar una breve introducción del problema y al estudio de un algoritmo clásico, el propuesto por Wagner y Fischer [34]. Dicho algoritmo proporciona la base que siguen los primeros trabajos sobre reconocimiento de patrones en árboles descritos en los capítulos posteriores. También mostraremos el problema del reconocimiento aproximado de cadenas y como se puede adaptar el algoritmo de Wagner y Fischer para resolverlo.
  - El **capítulo 4** presenta el problema de la comparación de árboles y los primeros algoritmos propuestos para resolverlo. El interés de estos trabajos estriba en que definen las propiedades y conceptos básicos que constituyen la base de los algoritmos que se presentarán en los siguientes capítulos. En concreto, se estudiará el algoritmo propuesto por Selkow [37] y el de Tai [43]. Este último es especialmente importante porque se trata del primero que propone una solución al problema de la comparación de árboles en tiempo polinomial y de forma no recursiva. Además, este autor define las tres operaciones básicas de edición sobre árboles y el concepto de distancia de edición que han sido utilizadas en los trabajos posteriores.
  - En el **capítulo 5** se presenta el algoritmo para la comparación de árboles propuesto por Zhang y Shasha[65]. Este algoritmo será el que utilicemos como punto de partida para nuestro trabajo sobre reconocimiento de patrones en bosques compartidos. Para ello revisaremos en profundidad sus fundamentos teóricos, así como su corrección y su complejidad computacional. Presentaremos también ejemplos detallados de su funcionamiento.
  - La segunda parte de la memoria termina con el **capítulo 6**. Este capítulo está dedicado a describir las extensiones del algoritmo de Zhang y Shasha necesarias para dar soporte al reconocimiento aproximado de patrones en árboles. Este será una extensión del problema del reconocimiento aproximado sobre cadenas. El interés de este tipo de técnicas es que permiten trabajar con árboles patrón más generales, sobre los cuales poder omitir detalles estructurales no relevantes. Se expondrán, además, las distintas posibilidades a la hora de definir el reconocimiento aproximado de árboles.

- La **parte III** está dedicada a describir nuestra aportación al problema del reconocimiento de patrones en bosques compartidos.

- Comenzaremos describiendo, en el **capítulo 7**, como son y como se generan las estructuras sintácticas sobre las que trabajarán nuestras modificaciones del algoritmo de Zhang y Shasha. El capítulo arranca presentando el problema del análisis sintáctico y una descripción del sistema ICE [56], basado en la interpretación tabular de *autómatas con pila* (AP) [28]. La segunda parte de este capítulo se dedica a revisar los formalismos de representación de estructuras sintácticas, y explica como se lleva a cabo la generación de estas representaciones en ICE.
- Después de presentar el sistema ICE y el formalismo de representación de estructuras sintácticas que emplea, el **capítulo 8** contiene la parte fundamental de nuestra aportación. Dedicaremos este a estudiar la integración del algoritmo de reconocimiento de patrones en árboles de Zhang y Shasha [65] con los analizadores generados por ICE. Dicha integración se centra en dos aspectos. En primer lugar, se describirá como adaptar el algoritmo de cálculo de la distancia de edición al tipo de bosques compartidos construidos siguiendo el formalismo de representación empleado en ICE. En segundo lugar, se describirá como aprovechar la compartición de estructuras sintácticas ofrecida por ICE, para evitar repetir cálculos de distancias que afecten a subestructuras compartidas por dos o más análisis.

- Finalmente, la **parte IV** contiene una evaluación experimental de nuestras propuestas, junto con las conclusiones más importantes de nuestro trabajo y las posibles líneas de trabajo futuro.

  - En el **capítulo 9** mostramos los resultados de una serie de experimentos prácticos que hemos realizado para evaluar nuestra propuesta. Nuestro principal objetivo al realizar estos experimentos ha sido probar la eficiencia computacional de la aproximación que proponemos y las ventajas que ofrece la utilización de estructuras compartidas. El segundo objetivo ha sido el de valorar las posibilidades de uso de este tipo de técnicas de reconocimiento de patrones y la influencia que tienen las elecciones de los usuarios en la eficiencia del proceso de reconocimiento.
  - Para terminar, en el **capítulo 10** expondremos la conclusiones más importantes de nuestro trabajo y esbozaremos las posibles lineas de aplicación de la técnica de reconocimiento de patrones en bosques compartidos que proponemos.

Capítulo 2
Conceptos previos

El objeto de este capítulo es introducir los conceptos y definiciones que serán empleados en el resto de la memoria y establecer la notación utilizada. Los conceptos son bien conocidos y pueden ser encontrados, aunque con pequeñas diferencias de notación, en múltiples referencias bibliográficas que tratan el análisis sintáctico [23, 1] y los algoritmos sobre árboles [20].

En una primera sección se repasarán conceptos generales sobre cadenas, lenguajes y gramáticas. Se revisarán seguidamente, las definiciones y notaciones utilizadas para el tratamiento de grafos y árboles. Finalmente, se definirán los conceptos básicos relativos al uso de grafos y árboles como estructuras de representación de relaciones sintácticas.

2.1. Lenguajes y gramáticas

En esta primera sección se desarrollan los conceptos y definiciones básicos utilizados en el estudio de los lenguajes y las gramáticas formales. Todos estos conceptos serán necesarios cuando se revisen las técnicas de análisis sintáctico y se estudie la integración del reconocimiento de patrones en árboles.

2.1.1. Alfabetos, cadenas y lenguajes

El primer concepto que es necesario definir es el de *alfabeto*. Desde el punto de vista de los lenguajes formales, un *alfabeto* Σ será un conjunto finito de símbolos, con los cuales formar los constituyentes de los lenguajes, o *cadenas*.

Definición 2.1 (cadena de caracteres)**.**
Una cadena w *sobre un alfabeto* Σ *es una secuencia de cero o más símbolos del alfabeto. La cadena que no contiene símbolos se denomina* cadena vacía *y se representa como* ε*. Resultando:*

- ε *es una cadena sobre el alfabeto* Σ
- *Si* w *es una cadena sobre* Σ *y* $a \in \Sigma$*, entonces* wa *también es una cadena sobre* Σ

El conjunto de todas las cadenas definidas sobre Σ*, incluida* ε*, se designa como* Σ^**.*

Es común emplear los términos *frase* o *sentencia* para referirse a una cadena, y *letra*, *carácter* o *palabra*[1] en lugar de símbolo de un alfabeto. Una propiedad básica de toda cadena es su longitud. La *longitud* de una cadena $w \in \Sigma$, denotada como $|w|$, se define como el número símbolos de que consta. En el caso de la cadena vacía, ε, su longitud es 0. Usando la noción de longitud se puede definir la operación de *concatenación*.

[1] Especialmente cuando se está trabajando con lenguajes naturales.

Definición 2.2.
Sea Σ *un alfabeto y sean* u,v $\in \Sigma$. *Se define la* concatenación de u y v, *denotada por* uv, *de la forma siguiente:*

- *Si* $|v| = 0$, $uv = u$
- *Si* $|v| > 0$, *tenemos que* $v = wa$ *con* $a \in \Sigma$ *y* $|w| = n - 1$. *Entonces* $uv = (uw)a$.

Es decir, uv es la cadena formada por la adición de los símbolos de v al final de la cadena u. Por ejemplo, empleando el alfabeto $\Sigma = \{a, b\}$ y dadas las cadenas de Σ^*, $u = ababab$ que $uv = abababbbbbb$. La cadena vacía es el elemento neutro del operador de concatenación. En efecto, para cualquier cadena $u \in \Sigma^*$, tenemos que $u = u\varepsilon = \varepsilon u$. En base a este operador de concatenación se definen los conceptos de *subcadena*, *prefijo* y *sufijo*.

Definición 2.3.
Sea Σ *un alfabeto y sean* $u, v \in \Sigma^*$.

- *Se dice que* u *es una* subcadena *de* v *si y sólo si* u *está contenido en* v, *es decir si* $\exists x, y \in \Sigma^* \ / \ v = xuy$.
- *Se dice que* u *es un* prefijo *de* v *si y sólo si* $\exists w \in \Sigma^* \ / \ v = uw$.
- *Se dice que* u *es un* sufijo *de* v *si y sólo si* $\exists w \in \Sigma^* \ / \ v = wu$.

Seguiremos la convención de emplear las letras $a, b, c, \ldots$ para indicar símbolos de un alfabeto y las letras $u, v, w, \ldots$ para referirnos a cadenas. De este modo, dada una cadena $w = a_1, a_2, \ldots, a_n \in \Sigma$, de longitud n; con w_i denotaremos al carácter situado en la i-ésima posición de esa cadena, y con $w_{i..j}$ denotaremos la subcadena de w que va del carácter en la posición i al carácter en la posición j.

Una vez definidos los constituyentes básicos de un lenguaje, sus propiedades y la notación a emplear, podemos definir formalmente el concepto de lenguaje.

Definición 2.4.
Sea Σ *un alfabeto, definimos un* lenguaje sobre Σ *como un subconjunto, finito o infinito, de* Σ^*.

Desde un punto de vista intuitivo, un lenguaje no es más que un conjunto de cadenas construidas empleando un alfabeto Σ determinado. En la práctica, serán necesarios formalismos que permitan especificar el conjunto de cadenas que forman un lenguaje de un modo más manejable que la simple enumeración de las cadenas que lo constituyen. La aproximación que se suele emplear en la práctica es la utilización de una gramática que genere el lenguaje, como veremos en el próximo apartado.

2.1.2. Gramáticas y análisis sintáctico

Una vez definido formalmente el concepto de lenguaje, es necesario introducir un mecanismo de representación adecuado que permita definir cómo deben ser las cadenas que forman parte de un lenguaje. A partir de ese mecanismo de representación debe ser posible generar todas las cadenas que forman parte del mismo y poder determinar si una cadena dada pertenece o no al lenguaje. En el caso de lenguajes finitos y con un número reducido de cadenas, es posible representarlos simplemente enumerando todas y cada una de las que los componen. Sin embargo, para lenguajes que contienen un número de cadenas muy grande o son infinitos, es necesario un mecanismo finito de representación. Este tipo de representaciones toman la forma de *gramáticas formales*.

Notación	Significado
$\Sigma, \Sigma_1, \Sigma_2, \ldots$	Alfabetos
$a, b, c, \cdots \in \Sigma$	Símbolos del alfabeto Σ, también letras o caracteres
$u, v, w, \cdots \in \Sigma^*$	Cadenas sobre el alfabeto Σ
$w_i \in \Sigma$	Carácter i-ésimo de la cadena w
$w_{i..j} \in \Sigma^*$	Subcadena comprendida entre el carácter i-ésimo y el j-ésimo de la cadena w
$G, G_1, G_2, \ldots$	Gramáticas. Si no se indica lo contrario se entenderá que son GICs
$V = N \cup \Sigma$	Conjunto total de símbolos de una gramática
$a, b, c, \cdots \in \Sigma$,	Símbolos terminales
$A, B, C, \cdots \in N$,	Símbolos no terminales
$X, Y, Z, \cdots \in V$,	Símbolos arbitrarios, terminales y no terminales
$u, v, w, \cdots \in \Sigma$	Cadenas de terminales
$\alpha, \beta, \gamma, \cdots \in V^*$	Cadenas arbitrarias de símbolos terminales y no terminales
\$	Símbolo de fin de cadena

Cuadro 2.1: Notaciones y convenciones sobre cadenas y gramáticas

Definición 2.5 (gramática)**.**
Una gramática *es una 4-tupla* $\mathcal{G} = (N, \Sigma, P, S)$ *donde:*

- Σ *es el alfabeto finito de la gramática o conjunto finito de* símbolos terminales*.*
- N *es un conjunto finito de* símbolos no terminales *o* variables*,* $N \cap \Sigma = \emptyset$
- P *es un subconjunto finito de* $(N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$ *a cuyos elementos denominaremos* producciones*,* reglas*, o* reglas de producción
- $S \in N$ *es el* símbolo inicial*, o* axioma *de la gramática*

En lo sucesivo se utilizarán las convenciones relativas a los elementos de una gramática mostradas en la tabla 2.1. Frecuentemente se prefiere representar las producciones $(\alpha, \beta) \in P$ como $\alpha \longrightarrow \beta \in P$. El primer miembro de una regla de producción $\alpha \longrightarrow \beta$ suele denominarse *parte izquierda* y el segundo *parte derecha*. Las reglas cuya parte derecha esté vacía se denominan *producciones-*ε y se notan como $\alpha \longrightarrow \varepsilon$.

Las cadenas del lenguaje se construyen partiendo del símbolo inicial S, siendo las producciones las encargadas de describir cómo se lleva a cabo esa generación. Empleando las reglas de producción de la gramática, se pueden construir distintas secuencias de símbolos terminales y no terminales a partir del símbolo inicial. Se denominará *formas sentenciales* a dichas secuencias.

Definición 2.6 (forma sentencial)**.**
Sea $\mathcal{G} = (N, \Sigma, P, S)$ *una gramática, entonces:*

- S *es una forma sentencial.*
- *Si* $\alpha\beta\gamma$ *es una forma sentencial y* $\beta \to \delta \in P$*, entonces* $\alpha\delta\gamma$ *también es una forma sentencial.*

Intuitivamente, S es la forma sentencial más simple. A partir de ella se generan las demás formas sentenciales. Dada una forma sentencial y una regla de producción se generará una nueva forma sentencial sustituyendo una ocurrencia de la parte izquierda de la regla en la primera, por la parte derecha de dicha regla. Un tipo especialmente interesante de forma sentencial es aquella que está formada exclusivamente por símbolos terminales. Estás formas sentenciales se denominan *sentencias* o *frases* y son las que formarán parte del lenguaje generado por la gramática.

Definición 2.7 (frase)**.**
Dada una gramática $\mathcal{G} = (N, \Sigma, P, S)$*, denominaremos* frase generada por una gramática *a cualquier forma sentencial que únicamente contenga símbolos terminales.*

La generación de formas sentenciales y sentencias descrita anteriormente puede formalizarse empleando el concepto de *derivación*

Definición 2.8 (derivación directa)**.**
Sea $\mathcal{G} = (N, \Sigma, P, S)$ *una gramática, se define una* derivación directa *o* derivación en un solo paso, $\underset{\mathcal{G}}{\Rightarrow}$*, como sigue:*

$$Si\ \alpha\beta\gamma \in (N \cup \Sigma)^*\ y\ \beta \to \delta \in P,\ entonces\ \alpha\beta\gamma \underset{\mathcal{G}}{\Rightarrow} \alpha\delta\gamma.$$

En el caso de una cadena de derivaciones directas, se dirá que $\alpha\beta\gamma$ deriva indirectamente $\alpha\delta\gamma$ *si y sólo si:*

- $\beta \underset{\mathcal{G}}{\Rightarrow} \delta_1 \underset{\mathcal{G}}{\Rightarrow} \delta_2 \ldots \underset{\mathcal{G}}{\Rightarrow} \delta_n \Rightarrow \delta$*, que notaremos* $\alpha\beta\gamma \overset{+}{\underset{\mathcal{G}}{\Rightarrow}} \alpha\delta\gamma$*, o bien*
- $\beta = \delta$ *ó* $\alpha\beta\gamma \overset{+}{\underset{\mathcal{G}}{\Rightarrow}} \alpha\delta\gamma$*, que notaremos* $\alpha\beta\gamma \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \alpha\delta\gamma$

En caso de conocer el número exacto, k*, de derivaciones directas, se usará la notación* $\alpha\beta\gamma \overset{k}{\underset{\mathcal{G}}{\Rightarrow}} \alpha\delta\gamma$*.*

Por lo tanto, empleando el concepto de derivación, es posible partir del axioma de una gramática y, aplicando un conjunto de reglas de producción, generar nuevas formas sentenciales. Este proceso se repetirá hasta dar lugar al conjunto de sentencias sobre el alfabeto Σ, que forman el lenguaje generado por la gramática. Formalmente tenemos:

Definición 2.9 (lenguaje generado por una gramática)**.**
Sea $\mathcal{G} = (N, \Sigma, P, S)$ *una gramática, el* lenguaje generado por la gramática*, es el conjunto* $L(\mathcal{G})$ *definido del siguiente modo:*

$$L(\mathcal{G}) = \left\{ w | w \in \Sigma^*,\ S \overset{*}{\Rightarrow} w \right\}$$

Ejemplo 2.1.
Sea la gramática $\mathcal{G}_N = (N_N, \Sigma_N, P_N, S_N)$*, donde:*

$$\begin{array}{ll} N_N = & \{S\} \\ \Sigma_N = & \{+, a\} \\ P_N = & \{S \to S + S,\ S \to a\} \\ S_N = S & \end{array}$$

La secuencia de derivaciones $S \Rightarrow S + S \Rightarrow a + S$*, da lugar a la forma sentencial* $a + S$*. Del mismo modo si se le añade a esa secuencia la derivación* $a + S \Rightarrow a + a$*, obtendremos la sentencia* $a + a$*.*
Como es fácil comprobar, el lenguaje generado por $\mathcal{G}_N$ *es el siguiente:*

$$L(\mathcal{G}_N) = \{a,\ a + a,\ a + a + a,\ \ldots,\ a + a + \cdots + a\}$$

En general, en función de la forma de las reglas de producción se pueden generar lenguajes con mayor o menor complejidad. Atendiendo a la forma de las reglas de producción, Chomsky [7] propone una jerarquía de cuatro clases. En ella se clasifican, de menor a mayor complejidad, las gramáticas formales y sus lenguajes asociados, de forma que cada nivel de la jerarquía incluye a las gramáticas y lenguajes del nivel anterior.

- *Gramáticas regulares.* En este caso, las producciones son de la forma: $A \rightarrow x$ ó $A \rightarrow xB$. Este tipo de producciones nos asegura que las todas las formas sentenciales generadas contendrán a lo sumo un único símbolo no terminal. Los lenguajes que pueden ser generados por este tipo de gramáticas se denominan *lenguajes regulares.*

- *Gramáticas independientes del contexto* (GICs). Las producciones tienen un único símbolo, no terminal, en la parte izquierda: $A \rightarrow \beta$. De esta forma, a la hora de realizar un paso de derivación directo, es posible decidir qué símbolo no terminal queremos reescribir independientemente del contexto que lo rodea. Los lenguajes que pueden ser generados por este tipo de gramáticas se denominan *lenguajes independientes del contexto.*

- *Gramáticas dependientes del contexto.* La parte izquierda de las producciones pueden contener cualquier combinación de símbolos terminales y no terminales, siempre y cuando sea de longitud menor o igual que la parte derecha. De esta forma aseguramos que al aplicar una derivación sobre una forma sentencial obtendremos otra forma sentencial de igual o mayor longitud. Las producciones siguen el patrón $\alpha \rightarrow \beta,\ |\alpha| \leq |\beta|$, siendo $|\alpha|$ la longitud de α, esto es, el número de símbolos en α. Los lenguajes que pueden ser generados por este tipo de gramáticas se denominan *lenguajes sensibles al contexto.*

- *Gramáticas con estructura de frase.* No existe ninguna restricción sobre las producciones. Los lenguajes que pueden ser generados por este tipo de gramáticas se denominan *lenguajes recursivamente enumerables.*

2.1.3. Gramáticas independientes del contexto

Debido a su menor complejidad, los lenguajes regulares e independientes del contexto han sido ampliamente estudiados y empleados en la práctica. Es por ello por lo que frecuentemente son empleados como base formal para diversas tareas relacionadas con el PLN . Dado que nuestro trabajo se centra en el procesamiento de árboles obtenidos a partir de GICs, presentaremos una serie de conceptos básicos referentes a este tipo de gramáticas.

En una gramática es posible tener distintas derivaciones que dan lugar a una misma cadena, y que sean equivalentes; en el sentido de que usan las mismas reglas en los mismos lugares, pero en diferente orden. Para evitar confusiones, es necesario fijar un criterio que determine el orden en el cual se vayan a aplicar dichas derivaciones. De esta forma surge el concepto de *derivación por la derecha* (resp. *por la izquierda*).

Definición 2.10 (derivación por la derecha (resp. por la izquierda))**.**
Dada una gramática $\mathcal{G} = (N, \Sigma, P, S)$ y una derivación directa $\alpha A \beta \Rightarrow \alpha\gamma\beta$, se dirá que se trata de una derivación directa por la derecha*, (resp.* por la izquierda*) si y sólo si:*

- $A \rightarrow \gamma \in P$
- $\beta \in \Sigma^*$ *(resp.* $\alpha \in \Sigma^*$*)*

Empleándose en ese caso la notación $\Rightarrow_{rm}$ (resp. $\Rightarrow_{lm}$). Al igual que sucedía con las derivaciones directas, se definen las derivaciones indirectas por la derecha (resp. por la izquierda):

- *Derivaciones en 1 ó más pasos, $\overset{+}{\Rightarrow}_{rm}$ (resp. $\overset{+}{\Rightarrow}_{lm}$).*
- *Derivaciones en 0 ó más pasos, $\overset{*}{\Rightarrow}_{rm}$ (resp. $\overset{*}{\Rightarrow}_{lm}$).*
- *Derivaciones en k pasos $\overset{k}{\Rightarrow}_{rm}$ (resp. $\overset{k}{\Rightarrow}_{lm}$).*

De estos dos tipos de derivaciones, se escogerá como *derivación canónica* la derivación por la derecha. En adelante, siempre que no se indique lo contrario, se considerará que cualquier derivación es una derivación canónica.

Ejemplo 2.2.
Continuando con la gramática del ejemplo 2.1, la cadena "$a+a+a$" se puede obtener con las siguientes dos derivaciones por la derecha:

$$S \Rightarrow_{rm} S+S \Rightarrow_{rm} S+a \Rightarrow_{rm} S+S+a \Rightarrow_{rm} S+a+a \Rightarrow_{rm} a+a+a$$

$$S \Rightarrow_{rm} S+S \Rightarrow_{rm} S+S+S \Rightarrow_{rm} S+S+a \Rightarrow_{rm} S+a+a \Rightarrow_{rm} a+a+a$$

y mediante estas otras dos por la izquierda:

$$S \Rightarrow_{lm} S+S \Rightarrow_{lm} a+S \Rightarrow_{lm} a+S+S \Rightarrow_{lm} a+a+S \Rightarrow_{lm} a+a+a$$

$$S \Rightarrow_{lm} S+S \Rightarrow_{lm} S+S+S \Rightarrow_{lm} a+S+S \Rightarrow_{lm} a+a+S \Rightarrow_{lm} a+a+a$$

Aún en el caso de utilizar derivaciones canónicas, es posible, tal y como muestra el ejemplo anterior, que para una misma forma sentencial exista más de un conjunto de derivaciones canónicas que la generen. En estos casos surge el concepto de *ambigüedad*.

Definición 2.11 (gramática ambigua)**.**
Se dice que una gramática $\mathcal{G} = (N, \Sigma, P, S)$ es una gramática ambigua *si y sólo si $\exists x \in L(\mathcal{G})$, tal que existen al menos dos derivaciones canónicas $S \overset{*}{\Rightarrow} x$ distintas.*

Esta noción de ambigüedad se puede extender de forma natural a los lenguajes.

Definición 2.12 (lenguaje ambiguo)**.**
Diremos que un lenguaje L no es ambiguo *si y sólo si existe una gramática $\mathcal{G}$ no ambigua tal que $L(\mathcal{G}) = L$. En caso contrario diremos que L es un* lenguaje ambiguo.

Por definición, el hecho de que un lenguaje sea ambiguo implica que todas las gramáticas que lo generan lo han de ser. Sin embargo, lo contrario no es cierto. Es posible escribir gramáticas ambiguas para generar lenguajes no ambiguos.

2.2. Arboles

Los grafos y los árboles proporcionan una descripción conveniente de muchas estructuras útiles en computación. En nuestro caso prestaremos especial atención a su utilización como mecanismo de representación de la estructura sintáctica de una frase. Es por esto por lo que la segunda parte de este capítulo está dedicada a revisar las definiciones básicas y los conceptos generales utilizados en los algoritmos de procesamiento de árboles y grafos. Se presentarán también, la terminología y las notaciones utilizadas en esta memoria al abordar los temas relativos al uso de árboles y grafos como mecanismos de representación de estructuras sintácticas.

2.2.1. Grafos y árboles

En primer lugar, es conveniente recordar la definición del concepto de *relación binaria*, que servirá de soporte para la definición formal del concepto de grafo.

Definición 2.13 (relación binaria)**.**
Una relación binaria *sobre un conjunto D es un subconjunto del producto cartesiano $D \times D$. El* cierre

transitivo *de una relación R, denotado* R^+*, y el* cierre reflexivo transitivo *de R, denotado* R^**, se definen de la siguiente forma:*

$$\begin{array}{rcl} R^0 & = & \{(x,x)|\ x \in D\}, \\ R^{n+1} & = & \{(x,y)|\ \exists z \in D,\ (x,z) \in R,\ (z,y) \in R^n\}, \\ R^+ & = & \bigcup_{n>0} R^n, \\ R^* & = & \bigcup_{n\geq 0} R^n \ = \ R^0 \cup R^+. \end{array}$$

El concepto de relación se utilizará para dar soporte a los enlaces existentes entre los elementos del grafo, tal y como se aprecia en la definición formal de *grafo dirigido.*

Definición 2.14 (grafo dirigido no ordenado)**.**
Un grafo dirigido no ordenado *G, es un par* $G = (A, R)$ *donde A es un conjunto de elementos llamados* nodos *o* vértices *y R es una relación binaria sobre el conjunto A.*

Los grafos dirigidos no ordenados son el tipo de grafo más general, puesto que no incluye ninguna restricción sobre el conjunto de nodos ni sobre la relación entre esos nodos. Intuitivamente, un grafo de este tipo no es más que una colección de nodos entre los cuales se establece una serie de relaciones, representadas por arcos.

A los nodos y arcos de cualquier tipos de grafo se le pueden asociar algún tipo de información, dando lugar a los *grafos etiquetados*.

Definición 2.15.
Sea $G = (A, R)$ *un grafo y sean* Σ_1 *y* Σ_2 *dos alfabetos. Una* etiquetación del grafo *es un par de funciones f y g, donde:*

- $f : A \longrightarrow \Sigma_1$*, etiqueta a los nodos, asociando a cada nodo de A un símbolo de* Σ_1
- $g : R \longrightarrow \Sigma_2$*, etiqueta a los arcos, asociando a cada par* $(a, b) \in R$ *un símbolo de* Σ_2

Tal como se indica en la definición formal, para un grafo $G = (A, R)$, un par $(a, b) \in R$ representa un arco o enlace. En ese caso, se dice que el arco *sale* de *a* y *entra* en *b*. Además *a* será el *predecesor* de *b* y *b* será el *sucesor* de *a*. Se denomina *grado de entrada* (resp. de *salida*) de un nodo al número de arcos que entran en él (resp. salen de él). Se pueden considerar varios arcos a la vez, formando lo que se denomina un *camino.*

Definición 2.16 (camino)**.**
En un grafo $G = (A, R)$*, una secuencia de nodos* $(a_0, a_1, \ldots, a_n)$ *con* $a_i \in A\ \forall i\ 0 \leq i \leq n$ *forma un* camino *de longitud n del nodo* a_0 *al nodo* a_n *si y sólo si existe un arco en G que deja el nodo* a_{k-1} *y entra en* a_k*,* $\forall i\ 1 \leq i \leq n$.

Intuitivamente, un camino no es más que una secuencia de nodos unidos por arcos. Si existe un camino desde a_0 a a_n, se dice que a_n es *accesible* desde a_0. Dado que se trata de un grafo dirigido la accesibilidad en sentido inverso no tiene por que ser cierta en todos los casos.

Un tipo especial de camino que pasa dos o más veces por un mismo nodo se denomina *ciclo* o *circuito.* Se dirá que un grafo tiene ciclos si se puede construir un camino (a_0, a_1, a_n) en el cual $a_0 = a_n$. Dado que la presencia de ciclos suelen representar un problema en su procesamiento, los grafos sin ciclos conforman un clase de especial relevancia debido a sus aplicaciones prácticas. Formalmente, un *grafo dirigido acíclico* se define del siguiente modo:

Definición 2.17 (grafo dirigido acíclico)**.**
Un grafo dirigido acíclico*(*GDA*), es un grafo dirigido que no posee ciclos.*

En los GDAs se suele utilizar una terminología propia para algunos de los conceptos vistos hasta el momento. Así, en el caso de que (a, b) sea un arco de un GDA, se dirá que a es un *ancestro directo* de b y que b es un *descendiente directo* de a. Esta idea se extiende en caso de que exista un camino entre a y b, de forma que a se denomina *ancestro* de b y b es *descendiente* de a. Además, los nodos cuyo grado de entrada es 0 se llamarán *nodos base* y los nodos con grado de salida 0 serán *nodos hoja*.

Una vez definidos los GDAs y la terminología a emplear, se puede empezar a revisar la definición de *árbol* y todos los demás conceptos relacionados. De forma muy simple, se puede decir que un árbol no es más que un tipo especial de GDA, que posee un único nodo base, denominado *raíz*. Formalmente tenemos la siguiente definición:

Definición 2.18 (árbol)**.**
Un árbol con raíz*, o simplemente un* árbol*, es una estructura* $T = (V, E, raíz(T))$*, donde* V *es un conjunto finito de* nodos*,* $raíz(T) \in V$ *es un nodo, denominado* nodo raíz *y* E *es una relación binaria sobre el conjunto* V *que satisface una serie de condiciones que se indicarán a continuación.*
Si $(u, v) \in E$ *se dirá que* (u, v) *es un* arco *y que el nodo* u *es el* padre *del nodo* v*, denotado como* $padre(v)$*. El conjunto de arcos definido en* E *debe verificar las siguientes condiciones:*

- *El nodo raíz no tiene padre.*
- *Cada nodo del árbol, excepto la raíz, tiene exactamente un padre.*
- *Todos los nodos son accesibles desde el nodo raíz a través de los arcos de* E*. Es decir,* $(raíz(T), v) \in E^*$*,* $\forall v \in V$*.*

Tal como se puede apreciar en la definición, un árbol es un GDA con sólo un nodo base, su raíz, del cual todos los demás nodos son descendientes. Además, en un árbol hay un único camino que vaya desde la raíz hasta cualquier otro nodo. Se define la *profundidad*[2] de un árbol T, notada como $prof(T)$, como la longitud del camino más largo desde la raíz a un nodo, más uno. Para simplificar la notación, y siempre que no exista confusión, se representará un árbol $T = (V, E, raíz(T))$ como T simplemente. Para indicar el número de nodos de ese árbol, $|V|$, se usará la notación $|T|$.

Los nodos de un árbol que tengan un padre común se dirá que son los *hijos* de ese nodo padre u, y se notarán como $hijos(u) = \{v \in V |\ (u, v) \in E\}$. Asimismo, de esos nodos que comparten un mismo padre se dirá que son *hermanos* entre si. Al igual que en los GDAs, los nodos que no tienen hijos se denominan *hojas* y los que no son hojas se dice que son *nodos internos*. El conjunto de *descendientes* de un nodo u se denota como $desc(u)$ y se define como:

$$\text{desc}(u) = \{v \in V |\ (u, v) \in E^+\}$$

Análogamente, el conjunto $\text{anc}(u)$ de los ancestros de un nodo u se define como:

$$\text{anc}(u) = \{v \in V |\ (v, u) \in E^+\}$$

El conjunto de *ancestros de nivel* n se define como:

$$\text{anc}^n(u) = \{v \in V |\ (v, u) \in E^n\}$$

De esta forma, el padre de un nodo es su único ancestro de nivel 1. Además, se sigue la convención de que todo nodo es su propio ancestro de nivel 0.

Al igual que con los grafos, se pueden asignar etiquetas a los nodos y arcos de un árbol. En nuestro caso sólo es necesario considerar la etiquetación de los nodos, de forma que se definen los *árboles etiquetados* del siguiente modo:

[2] También denominada *altura*.

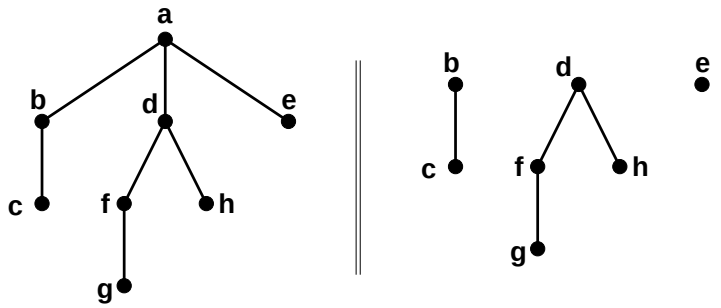


Figura 2.1: Ejemplos de árboles y subárboles.

Definición 2.19 (árbol etiquetado)**.**
Se dirá que un árbol $T = (E, V, raíz(T))$ *está* etiquetado *si, dado un alfabeto* Σ*, existe una función* $etiqueta : V \longrightarrow \Sigma$*, que asigne a cada nodo* $v \in V$ *del árbol un símbolo,* $etiqueta(v)$*, de ese alfabeto* Σ*.*

Frecuentemente es deseable establecer algún tipo de ordenación entre los hijos de los nodos de un árbol, dando lugar a los *árboles ordenados*.

Definición 2.20 (árbol ordenado)**.**
Dado un árbol T*, se dice que es un* árbol ordenado *si para cualquiera de sus nodos internos con* k *hijos, estos* k *nodos pueden ser numerados de forma única como* $i, \ldots, k$*. Se notará, entonces, al hijo* i*-ésimo de un nodo* u *como* $hijo(i, u)$*.*

Además, si un nodo u de un árbol tiene k hijos, con $k > 0$, se sigue la convención de denominar *hijo más a la izquierda* al nodo $hijo(1, u)$ y, respectivamente, *hijo más a la derecha* a $hijo(k, u)$. De forma análoga, siendo $hijo(i, u)$ uno de esos hijos, se dirá que los nodos $hijo(j, u)$, con $1 \leq j < i$, son sus hermanos izquierdos y los nodos $hijo(j, u)$, con $i < j \leq k$ sus hermanos derechos. Salvo que se indique expresamente, de ahora en adelante, todas las referencias a árboles se referirán a árboles etiquetados y ordenados.

Ejemplo 2.3.
Para ilustrar estas definiciones, en la parte izquierda de la figura 2.1 se muestra un ejemplo de un árbol ordenado y etiquetado. La raíz de este árbol está etiquetada como a*, su tamaño,* $|T|$*, es 7 y su profundidad es 4, que se corresponde con el camino* a, d, f, g*. Los descendientes directos del nodo* d *son los nodos* f *y* h*, en ese orden. Los ancestros del nodo* c *son* b*, de nivel 1, y* a*, ancestro de nivel 2.*

Cualquier nodo de un árbol mantiene con sus descendientes la misma relación, expresada en la definición formal, que la existente entre la raíz del árbol y todos los nodos del árbol. Esto significa que a partir de cualquier nodo de un árbol puede construirse otro árbol formado por él mismo y todos sus descendientes. Esto es lo que se denomina un *subárbol*.

Definición 2.21 (subárbol)**.**
Siendo u *un nodo de un árbol* $T = (V, E, raíz(T))$*. El* subárbol *con raíz en* u*, denotado como* $T[u]$*, será el árbol* (V', E', u) *construido de la forma:*

$$\begin{aligned} V' &= \{u\} \cup desc(u) \\ E' &= E \cap (V' \times V') \end{aligned}$$

En caso de que t *sea un árbol etiquetado y/o ordenado, las etiquetas de los nodos y la ordenación de los hijos se mantendrá en todos sus posibles subárboles.*

Cuando sea necesario manejar de forma conjunta un grupo de árboles se usará el concepto de *bosque*. Un *bosque ordenado*, o simplemente *bosque*, no es más que una secuencia ordenada de árboles en la cual ninguno de ellos tiene nodos en común con otro. El bosque B, formado por los árboles $T_1, T_2, \ldots, T_k$, se denotará como $B = \langle T_1, T_2, \ldots, T_k \rangle$. Dado un árbol ordenado $T = (V, E, raíz(T))$, en el cual el nodo raíz tiene k hijos, siempre se podrá extraer un bosque ordenado, B_T, a partir de T, que estará formado por el conjunto de los k subárboles con raíz en los hijos de $raíz(T)$. Es decir, obtendremos el bosque $B_T = \langle T[hijo(1, raíz(T))], T[hijo(2, raíz(T))], \ldots T[hijo(k, raíz(T))] \rangle$.

En la parte derecha de la figura 2.1 se muestra el bosque compuesto por el conjunto de subárboles con raíz en los hijos del nodo a del árbol descrito en el ejemplo 2.3.

Para facilitar el procesamiento y simplificar las notaciones, se suele asignar un índice único a cada nodo del árbol de forma que se pueda identificar unívocamente. Llevar esto a cabo supone un ordenamiento de los nodos del árbol $T = (V, E, raíz(T))$, para después asignar, de forma consecutiva, los índices desde 1 hasta $|T|$. Los dos ordenamientos usuales, y los que se emplean en esta memoria, son el *preorden* y el *postorden*.

Definición 2.22 (preorden)**.**
Dado un nodo u de un árbol $T = (V, E, raíz(T))$, se le puede asignar un índice *o* numeración en preorden*, $pre(u)$, de acuerdo a las siguientes reglas:*

- *La numeración en preorden de $raíz(T)$ es* 1*.*
- *La numeración en preorden del hijo más a la izquierda del nodo u es $pre(u) + 1$.*
- *Siendo u el siguiente hermano a la derecha de v y siendo p el mayor índice en preorden asignado a un nodo en $T[v]$[3], entonces $pre(u) = p + 1$.*

La ordenación en preorden asigna los índices a los nodos de un árbol de forma descendente. Comienza numerando la raíz y procesa a sus descendientes de izquierda a derecha, aplicando de forma recursiva esta numeración sobre los subárboles descendientes. El otro tipo de numeración, el postorden, asigna índices de forma ascendente. Tal como se muestra a continuación, comienza por la hoja más a la izquierda, numerando de forma consecutiva a sus hermanos derechos hasta terminar asignando un índice al nodo raíz.

Definición 2.23 (postorden)**.**
De igual modo, dado un nodo u de un árbol $T = (V, E, raíz(T))$, se le puede asignar un índice *o* numeración en postorden*, $post(u)$, de acuerdo a las siguientes reglas:*

- *La numeración en postorden de la hoja más a la izquierda de T es* 1*.*
- *Si u es un nodo interno, y p es el mayor índice en preorden asignado a un nodo en $T[u]$[4], entonces $post(u) = p + 1$.*
- *Siendo v la hoja más a la izquierda del subárbol con raíz en el siguiente hermano a la derecha de u, entonces $post(v) = post(u) + 1$.*

La figura 2.2 muestra un ejemplo de estos dos tipos de ordenamiento. En en el árbol de la izquierda se muestran, entre paréntesis, los índices asociados a los nodos en preorden, y en el de la derecha los índices en postorden.

[3] Ese nodo será la hoja más a la derecha descendiente de v.
[4] Ese nodo será el hijo más a la derecha de u.

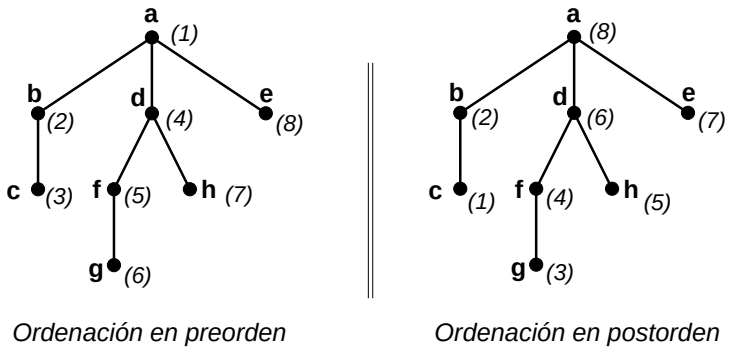


Figura 2.2: Numeración de los nodos de un árbol.

Dado un árbol T cuyos nodos estén numerados de acuerdo a uno de los órdenes anteriores, se usará la notación $t[i]$ para referirnos al nodo i-ésimo según el orden elegido. En esta línea, $T[i]$ se referirá al subárbol con raíz en el nodo $t[i]$ y la notación $T[i..j]$ se referirá al bosque formado por los nodos comprendidos entre $t[i]$ y $t[j]$, ambos incluidos.

Algunas propiedades importantes de los ordenamientos citados se enuncian en los siguientes lemas, en los cuales se relacionan los índices en preorden y postorden con las relaciones jerárquicas entre los nodos de un árbol, en concreto con el concepto de ancestro.

Lema 2.1.
Sean u y v dos nodos de un árbol T. El nodo u es un ancestro de v si y sólo si $pre(u) < pre(v)$ y $post(u) > post(v)$.

Demostración.
Se puede encontrar la demostración de este lema en [23]. □

Derivadas de la ordenación en preorden se verifican las siguientes propiedades, de demostración inmediata.

Lema 2.2.
Dado un árbol T, ordenado en preorden, siendo $t[i_1]$ y $t[i_2]$ dos de sus nodos, y siendo $t[i_1]$ un ancestro de $t[i_2]$, tenemos que:

1. *$\forall i$ tal que $i_1 < i \leq i_2$, $t[i]$ es un descendiente de $t[i_1]$.*
2. *Siendo $t[i_3]$ el padre de $t[i_2]$, $t[i_3]$ es $t[i_2 - 1]$ o un ancestro de $t[i_2 - 1]$; y además $t[i_3]$ está en el camino de $t[i_1]$ a $t[i_2 - 1]$.*

Demostración.
La demostración de estos resultados puede encontrarse en [43]. □

Para concluir este repaso, la tabla 2.2 resume las notaciones empleadas en esta memoria en lo que corresponde a los árboles.

2.2.2. Arboles de análisis sintáctico

Hasta el momento se han representado las derivaciones de las cadenas generadas por las gramáticas, tal y como han sido definidas, como una secuencia de derivaciones canónicas directas. En la práctica, se suele optar por compactar estas secuencias en forma de *árboles de derivación*.

Notación	Significado
$T, T_1, T_2, ...,\ P, D$	Arboles etiquetados y ordenados
$\|T\|$	Número de nodos en un árbol T
$raíz(T)$	Nodo raíz del árbol T
$prof(T)$	Profundidad de un árbol T
$u, v, \cdots \in T$	Nodos de un árbol T
$etiqueta(u)$	Etiqueta asociada al nodo u
$padre(u)$	Nodo padre del nodo u
$hijo(i, u)$	Hijo i-ésimo del nodo u
$hijos(u)$	Conjunto de nodos hijos del nodo u
$desc(u)$	Conjunto de nodos descendientes del nodo u
$anc(u)$	Conjunto de nodos ancestros del nodo u
$pre(u), post(u)$	Indices del nodo u en preorden y postorden, respectivamente
$t[i]$, con $1 \leq i \leq \|T\|$	Nodo i-ésimo del árbol T según un orden dado
$T[i]$, con $1 \leq i \leq \|T\|$	Subárbol con raíz en el nodo i-ésimo del árbol T
$T[i..j]$, con $1 \leq i \leq j \leq \|T\|$	Bosque formado por los nodos comprendidos entre los nodos $t[i]$ y $t[j]$

Cuadro 2.2: Notaciones y convenciones sobre árboles

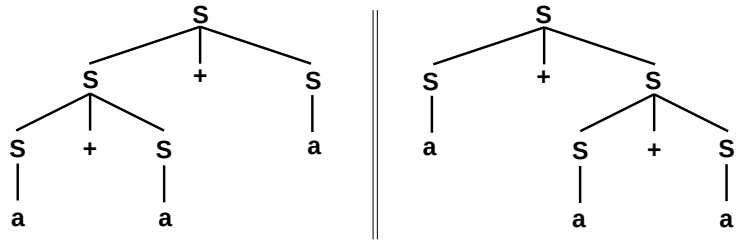


Figura 2.3: Ejemplos de árboles de derivación.

Definición 2.24 (árbol de derivación)**.**
Un árbol de derivación *para una GIC* $\mathcal{G} = (N, \Sigma, P, S)$ *es un árbol ordenado y etiquetado* $\mathcal{D}$ *verificando:*

- *Si* X *es una etiqueta de un nodo, entonces* $X \in \Sigma \cup N\ \cup \{\varepsilon\}$.
- *Si* u *es un nodo etiquetado con un símbolo no terminal* A*, y sus hijos, ordenados de izquierda a derecha, tienen como etiquetas a* $X_1, \ldots, X_n$*, entonces* $A \rightarrow X_1 \cdots X_n$ *es una producción de la gramática.*

Tal y como indica la definición, los nodos de los árboles de derivación están etiquetados con los símbolos de la gramática. Cada nodo interno estará etiquetado con un símbolo no terminal que aparezca en el lado izquierdo de una regla. Además, sus hijos estarán etiquetados con los símbolos terminales y no terminales presentes en la parte derecha de esa misma producción y ordenados, de izquierda a derecha, tal y como aparecen en la parte derecha de dicha regla. De esta forma, es posible representar de forma compacta y manejable un conjunto de derivaciones. Sin embargo, no es posible indicar explícitamente en un árbol de derivación el orden el que se tienen que aplicar las reglas utilizadas. El hecho de no indicar el orden de las derivaciones hace posible que un mismo árbol puede representar a más de una derivación indirecta que de lugar a una misma forma sentencial.

La figura 2.3 muestra los árboles correspondientes a las derivaciones mostradas en el ejemplo 2.2. Como se puede apreciar en esta figura, cada uno de los árboles se corresponde con una de las dos posibles derivaciones de la cadena "$a + a + a$" mostradas en el ejemplo y que se diferencian únicamente en el

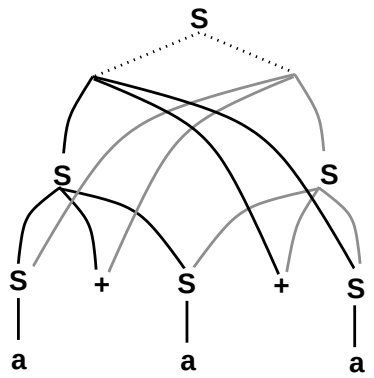


Figura 2.4: Representación de bosques de análisis como grafos Y-O.

orden de aplicación de las reglas. Como también se puede observar en esta figura, la reconstrucción de la forma sentencial representada por un árbol de derivación es directa. Simplemente basta con concatenar, en orden, los símbolos asociados a las hojas del árbol. Esa idea de concatenación de hojas se recoge en el concepto de *frontera*

Definición 2.25 (frontera de un árbol)**.**
Sea una GIC $\mathcal{G} = (N, \Sigma, P, S)$ *y sea* T *un árbol de derivación para esa gramática. La* frontera *del árbol de derivación* T *es la cadena,* $\alpha \in \{\Sigma \cup N\}^*$*, resultante de concatenar, ordenadas de izquierda a derecha, todas las etiquetas asociadas a las hojas de* T*.*

En la práctica, serán especialmente interesantes los árboles de derivación que representen la generación de las cadenas del lenguaje, esto es, aquellos cuya frontera esté formada exclusivamente por símbolos terminales. Este tipo particular de árboles de derivación se denominan *árboles de análisis*.

Definición 2.26 (árbol de análisis)**.**
Dada una GIC $\mathcal{G} = (N, \Sigma, P, S)$*, un* árbol de análisis *es un árbol de derivación donde:*

- *La etiqueta de la raíz es* S*.*
- *La frontera del árbol es* $w \in \mathcal{L}(\mathcal{G})$*.*

Como se desprende de la definición, los árboles de análisis no son más que árboles de derivación con la restricción de que su raíz debe ser el axioma de la gramática y todas sus hojas deben ser símbolos terminales. De este modo, un árbol de análisis representará como se puede generar un cadena del lenguaje asociado a una GIC, indicando que producciones se deben aplicar, aunque no el orden en el que deben realizarse las derivaciones correspondientes. Los dos árboles de la figura 2.3 son posibles árboles de análisis para la cadena "$a + a + a$" del ejemplo 2.2 empleando la gramática $\mathcal{G}_N$.

Para terminar esta sección introducimos una alternativa comúnmente empleada para representar de forma conjunta varios árboles de derivación en una única estructura de representación. En estos casos nos referiremos al conjunto así formado como *bosque de derivación*. Esta situación ocurrirá típicamente en el análisis de sentencias ambiguas. En esos casos existirá más de una derivación canónica y se deseará representarlas todas de forma adecuada. Frecuentemente, los árboles implicados en esos bosques de derivación tendrán subárboles comunes, por lo que resultará conveniente compartir dichas partes representando el bosque como un grafo Y-O al que se denomina *bosque compartido*. En estos bosques, que estudiaremos con detalle en el capítulo 7, no se cumple la exigencia de que los árboles que los componen tengan nodos diferentes. Así, es posible que un mismo conjunto de nodos pertenezca a más de

un árbol de análisis. Si bien esta situación puede complicar ligeramente el procesamiento de esos árboles, permitirá ahorrar espacio en la representación de los múltiples análisis de la cadena. La figura 2.4 muestra una representación del grafo Y-O que representa a los dos análisis posibles de la cadena "$a + a + a$" del ejemplo 2.2. En esta figura las lineas punteadas representan las diferentes alternativas de los nodos-O. Los nodos-Y se corresponden con los nodos usuales de los árboles de análisis. Y los nodos con más de un padre, representan la compartición de estructuras comunes entre dos o más posibles análisis de la cadena.

Parte II

Reconocimiento de Patrones en Arboles

Capítulo 3

Reconocimiento de patrones en cadenas

En este capítulo se realiza una introducción al problema del reconocimiento de patrones en cadenas. Como veremos en capítulos posteriores, es un problema íntimamente relacionado con el reconocimiento de patrones en árboles. De hecho, la comparación de cadenas de caracteres resulta ser un caso particular de la comparación de árboles. Esto es así, puesto que toda cadena de caracteres puede representarse mediante un árbol, en el cual cada carácter de la cadena se corresponde con una hoja. Ambos problemas se pueden formular como problemas de cálculo de distancias y, en ambos casos, se sigue la aproximación de medir esa distancia como el coste de una transformación. En un caso, se trata de la transformación de un árbol en otro distinto, y en el otro, es la transformación de una cadena en otra.

El reconocimiento de cadenas es un problema ampliamente estudiado [32], por lo que nos limitaremos a presentar una breve introducción del mismo y al estudio de un algoritmo clásico, el propuesto por Wagner y Fischer [34]. Si bien este algoritmo no es especialmente eficiente, si es de interés para nuestro trabajo porque las estrategias de reconocimiento de patrones sobre árboles que estudiaremos lo toman como punto de partida. Estos algoritmos pretenden extender la aproximación seguida en el de Wagner y Fischer al caso de los árboles.

Comenzaremos este capítulo introduciendo el reconocimiento de patrones en cadenas, que será formulado como un problema de cálculo de distancias de edición. Veremos a continuación la solución propuesta por Wagner y Fischer, y estudiaremos su funcionamiento. Terminaremos el capítulo presentando el problema del reconocimiento aproximado de patrones en cadenas. Estudiaremos distintas alternativas a la hora de definirlo y cómo se puede resolver dentro del marco del algoritmo de Wagner y Fischer.

3.1. Definición del problema

La aproximación al reconocimiento de patrones en cadenas que vamos a describir se basa en la noción de *distancia de edición*. Se definirá un conjunto de *operaciones de edición* que se podrán aplicar sobre las cadenas para transformarlas. Generalmente se consideran las operaciones de cambio, inserción y borrado de caracteres. Para determinar la proximidad entre dos cadenas se determinará la mejor forma de transformar una de ellas en la otra, empleando las operaciones definidas. Asociando un coste numérico a cada operación de edición empleada se podrá obtener un valor numérico que cuantifique la distancia de edición entre las dos cadenas.

Para formalizar el problema, comenzaremos definiendo las *operaciones de edición* sobre cadenas que utilizaremos en adelante.

Definición 3.1 (operaciones de edición sobre cadenas)**.**
Sea x$=a_1, a_2, \ldots, a_n$ *una cadena de longitud* n *sobre un alfabeto* Σ*. Se definen los siguientes tipos de operaciones de edición sobre la cadena* x*:*

- Cambio. *Siendo* x=uav*, la operación de* cambio *del carácter* a *por el carácter* b*, denotada* $a \rightarrow b$*, dará lugar a la cadena* x'=ubv.
- Inserción. *Siendo* x=uv*, la operación de* inserción *del carácter* a *, denotada* $\varepsilon \rightarrow a$*, dará lugar a la cadena* x'=uav.
- Borrado. *Siendo* x=uav*, la operación de* borrado *del carácter* a *, denotada* $a \rightarrow \varepsilon$*, dará lugar a la cadena* x'=uv.

Siendo $\varepsilon \notin \Sigma$ *un carácter especial que denotará al carácter nulo.*

Intuitivamente, una operación de cambio simplemente sustituye un carácter de una cadena por otro. La operación de inserción añade un nuevo carácter y la de borrado elimina uno de los presentes en la cadena inicial. Por ejemplo, la operación $a \rightarrow e$ sobre la cadena $x =$"`casa`"da lugar a la cadena $x'=$"`cesa`". La operación de borrado $a \rightarrow \varepsilon$ sobre la misma cadena da lugar a $x'=$"`cas`z la inserción $\varepsilon \rightarrow r$ produce $x'=$"`casar`". Si bien pueden definirse otras posibles operaciones de edición sobre cadenas, como por ejemplo el intercambio de caracteres en dos posiciones adyacentes, todas ellas podrán llevarse a cabo mediante la combinación de dos o más de las operaciones anteriores. Por ejemplo, intercambiando la "`c`z la "`a`"de la cadena $x=$"`casa`", obtenemos $x'=$"`acsa`", que es equivalente a realizar la eliminación del carácter "`c`", $c \rightarrow \varepsilon$, y su posterior inserción, $\varepsilon \rightarrow c$, a continuación del carácter "`a`".

Para transformar las operaciones de edición en un valor numérico se define una *función de coste*, γ, que asocia a cada operación un número real no negativo.

Definición 3.2 (función de coste)**.**
Se define la función de coste γ *que asocia a cada operación de edición* $a \rightarrow b$*, con* $a, b \in \Sigma \cup \{\varepsilon\}$*, un número real no negativo,* $\gamma(a \rightarrow b)$*, que verifica las siguientes propiedades:*

1. $\gamma(a \rightarrow b) \geq 0,\ \forall a, b \in \Sigma \cup \{\varepsilon\}$
2. $\gamma(a \rightarrow a) = 0,\ \forall a \in \Sigma$
3. $\gamma(a \rightarrow b) = \gamma(b \rightarrow a),\ \forall a, b \in \Sigma \cup \{\varepsilon\}$
4. $\gamma(a \rightarrow c) \leq \gamma(a \rightarrow b) + \gamma(b \rightarrow c),\ \forall a, b, c \in \Sigma \cup \{\varepsilon\}$

Si $S = op_1, op_2, \ldots op_n$ *es una secuencia de operaciones de edición , su* coste*,* $\gamma(S)$*, se calcula como:*

$$\gamma(S) = \sum_{i=1}^{n} \gamma(op_i)$$

Como se aprecia en la definición, la función γ es una métrica, puesto que debe cumplir las cuatro condiciones anteriores. Una vez definida la función de coste correspondiente se podrá transformar en un valor numérico el conjunto de operaciones de edición necesarias para transformar una cadena en otra, como se muestra en el ejemplo 3.1.

Ejemplo 3.1.
Suponiendo que el coste de cada operación de edición sea 1, la distancia entre la cadena $x=$"`casa`" *e* $y=$"`cesar`"*sería 2. Dicha distancia se corresponde con el coste de cambiar el carácter "*`a`*"por "*`e`*",* $\gamma(a \rightarrow e) = 1$*, más el coste de insertar el carácter "*`r`*",* $\gamma(\varepsilon \rightarrow r) = 1$.

Normalmente, dada dos cadenas, existirán múltiples conjuntos de operaciones de edición que permitan transformar una en otra. Cada una de ellas empleará distintas operaciones y, por lo tanto, podrán tener diferentes costes. Para calcular la distancia de edición entre dos cadenas nos interesa hallar el conjunto de operaciones que den lugar al menor coste de transformación posible. Teniendo esta última idea en cuenta se define la *distancia de edición* entre dos cadenas del siguiente modo:

Definición 3.3 (distancia de edición entre cadenas)**.**
Sean x *e* y *dos cadenas sobre un alfabeto* Σ*, se define la* distancia de edición *entre* x *y* y*, denotada* d(x,y)*, como:*

$$d(x, y) = min\{\gamma(S) |\ S \text{ es una secuencia de operaciones de edición que transforma } x \text{ en } y\}$$

Intuitivamente, la distancia entre dos cadenas es igual al coste de la secuencia de operaciones que transforma una de ellas en la otra con el menor coste posible. Dado que no todas las secuencias de operaciones de edición son legales, es necesario introducir restricciones adicionales. Para ello, se introduce el concepto *traza*.

Definición 3.4 (traza)**.**
Sean x *e* y *dos cadenas sobre un alfabeto* Σ*. Se define una* traza *entre* x *e* y *como una* $3-upla$ (TR, x, y)*, donde* TR *es un conjunto de pares de la forma:*

$$TR = \{(i, j) \mid 1 \leq i \leq |x|,\ 1 \leq j \leq |y|,\ \exists\ x_i \rightarrow y_j \text{ en la secuencia que transforma x en y}\}$$

Verificando que, dados los pares (i_1, j_1) *y* (i_2, j_2) *en* TR *:*

1. $i_1 = i_2$*, si y sólo si* $j_1 = j_2$*.*
2. $i_1 < i_2$*, si y sólo si* $j_1 < j_2$*.*

Una traza TR representa de forma abreviada, mediante un conjunto de pares, una secuencia válida de operaciones de edición que transforma una cadena x en otra cadena x. Cada par (i, j) de una traza TR indica que $x_i \rightarrow y_j$ es una de las operaciones incluidas en esa secuencia, siendo x_i e y_j, respectivamente, el i-ésimo carácter de x y el j-ésimo de y. Si ningún par de la traza contiene a i como primer elemento, entonces significa que el carácter x_i deberá ser borrado. Análogamente, todo j que no aparezca como segundo elemento en TR, supondrá realizar una operación de inserción del carácter y_j. La condición *1* impide que ningún carácter de x ó y sea utilizado en más de una operación de cambio de caracteres. La condición *2* asegura que en la cadena resultante se mantiene el orden de izquierda a derecha que existía en la cadena original.

Finalmente, el coste de una traza TR se obtendrá a partir del coste de la secuencia de operaciones de edición asociadas. Así, siendo I el conjunto de caracteres de la cadena x que no aparecen en la traza TR, y J el conjunto de caracteres de y que no aparecen en la traza, tenemos que el *coste de una traza* TR se calculará de la siguiente forma:

$$\gamma(TR) = \sum_{i \in I} \gamma(x_i \rightarrow \varepsilon) + \sum_{j \in J} \gamma(\varepsilon \rightarrow y_j) + \sum_{(i,j) \in TR} \gamma(x_i \rightarrow y_j)$$

3.2. Algoritmo de Wagner-Fischer

Uno de los algoritmos clásicos de reconocimiento de patrones en cadenas es el de Wagner y Fischer [34], que se describe brevemente en esta sección. Su importancia radica en que los primeros algoritmos de reconocimiento de patrones en árboles, que se presentarán en secciones posteriores, siguen la misma arquitectura general, tratando de extender la solución que ahora presentamos a la comparación de árboles.

Este algoritmo es un ejemplo clásico del uso de técnicas de programación dinámica [5]. El principio básico de este tipo de estrategias es el de resolver los problemas complejos mediante la combinación de las soluciones de subproblemas más sencillos. Es común el uso de una estructura de almacenamiento donde

guardar la soluciones a los subproblemas, generalmente en forma de tabla de soluciones parciales, de forma que cada subproblema se resuelva una sola vez y su resultado esté disponible para futuros cálculos.

Dado un par de cadenas x e y, de longitudes $|x| = n$ e $|y| = m$, el algoritmo de Wagner y Fischer [34] crea una matriz de distancias, D, de tamaño $(n+1)\times(m+1)$. Cada entrada $D[i,j]$, almacena la distancia entre la subcadena formada por los i primeros caracteres de x, $x_{1..i}$, y la subcadena formada por los j primeros de y, $y_{1..j}$. Las entradas de la fila 0, $D[0,j]$, $\forall j$; contienen la distancia entre ε y $y_{1..j}$. Esta distancia se corresponde con el coste de insertar cada uno de los caracteres que componen el prefijo $y_{1..j}$. Del mismo modo, las entradas de la columna 0, $D[i,0]$, $\forall i$; almacenan las distancia entre cada prefijo $x_{1..j}$ de x y la ε, que se corresponde con el coste de borrar los primeros i caracteres de x. Así, la matriz se inicializa según las siguientes fórmulas:

$$\begin{array}{lcl} D[0,0] & = & 0 \\ D[0,j] & = & D[0,j-1] + \gamma(\varepsilon \rightarrow y_j), \forall\ 1 \leq j \leq m \\ D[i,0] & = & D[i-1,0] + \gamma(x_i \rightarrow \varepsilon), \forall\ 1 \leq i \leq n \end{array}$$

El cálculo de cada celda de la matriz se realiza a partir de las celdas adyacentes ya calculadas anteriormente, hasta llenar por completo de la matriz de distancias. De esta forma, en el cálculo de $D[i,j]$ se parte de los valores de $D[i-1,j-1]$, $D[i,j-1]$ y $D[i-1.j]$, a los que se les suma el coste de la operación de edición que corresponda en cada caso, cambio, inserción y borrado, respectivamente, almacenando el mínimo de esas tres distancias. Los tres casos son:

- Sumar a $D[i-1,j]$ el coste de borrar el carácter x_i.
- Sumar a $D[i,j-1]$ el coste de insertar el carácter y_j.
- Sumar a $D[i-1,j-1]$ el coste de cambiar el carácter x_i por y_j.

El cálculo de $D[i,j]$ se resume, por lo tanto, en la siguiente fórmula:

$$D[i,j] = \mathrm{m}in \left\{ \begin{array}{l} D[i-1,j] + \gamma(x_i \rightarrow \varepsilon) \\ D[i,j-1] + \gamma(\varepsilon \rightarrow y_j) \\ D[i-1,j-1] + \gamma(x_i \rightarrow y_j) \end{array} \right\}$$

El algoritmo 3.1 muestra como se rellenan las celdas de la matriz $D[i,j]$ utilizando esas distancias parciales calculadas en pasos anteriores. La terminación del algoritmo está asegurada, dado que los valores de las nuevas distancias se obtienen a partir de distancias calculadas en pasos anteriores. También es trivial comprobar que al terminar el algoritmo, la celda $D[n,m]$ almacenará el valor de la distancia mínima entre las dos cadenas iniciales x e y. El siguiente ejemplo ilustra el funcionamiento del algoritmo.

Ejemplo 3.2.
Partimos de las cadenas x=*"`faena`", de longitud 5, e* y=*"`cadena`", de longitud 6. Supondremos una función γ que asigne costes unitarios a las operaciones de edición del siguiente modo:*

$$\gamma(a \rightarrow b) = \left\{ \begin{array}{ll} 0 & si\ a = b\ con\ a \neq \varepsilon,\ b \neq \varepsilon \\ 1 & en\ otro\ caso \end{array} \right.$$

El algoritmo 3.1 construye la siguiente matriz de distancias $D[i,j]$.

$D[i,j]$		c	a	d	e	n	a
	0	1	2	3	4	5	6
f	1	1	2	3	4	5	6
a	2	2	1	2	3	4	5
e	3	3	2	2	2	3	4
n	4	4	3	3	3	2	3
a	5	5	4	4	4	3	2

Algoritmo 3.1 Distancia de Wagner-Fischer

Entrada: X: cadena de longitud n, Y: cadena de longitud m
Salida: D: matriz de distancias de edición entre X e Y

```
/*inicialización matriz*/

D[0,0] = 0
for i = 1 to n do
   D[i,0] = D[i-1,0] + γ(x_i → ε)
end for
for j = 1 to m do
   D[0,j] = D[0,j-1] + γ(ε → y_j)
end for

/*cálculo de distancias*/

for i = 1 to n do
   for j = 1 to m do
      D[i,j] = min { D[i-1,j] + γ(x_i → ε)        /*borrado de x_i*/
                   { D[i,j-1] + γ(ε → y_j)        /*inserción de y_j*/
                   { D[i-1,j-1] + γ(x_i → y_j)    /*cambio de x_i por y_j*/
   end for
end for
```

El algoritmo comienza inicializando la matriz de distancias D, de dimensión (7×6). Primero calcula las distancias entre los prefijos de la cadena x *y ε, almacenando sus valores en las celdas de la primera columna, $D[0, j]$, $\forall j$. A continuación, hace lo mismo con las distancias entre la cadena vacía y los prefijos de la cadena* y*, almacenando esos valores en las celdas de la primera fila, $D[0, j]$, $\forall j$.*
El bucle principal aplica las fórmulas anteriores entre pares de prefijos de x *y de* y*. En cada iteración, se incrementa en una unidad la longitud de esos prefijos hasta llegar al cálculo de las cadenas completas. La distancia resultante para las cadenas* x *e* y *se almacena en la celda inferior derecha, $D[5, 6]$, siendo su valor 2. Para obtener dicho valor, el algoritmo toma el mínimo de los siguientes tres valores:*

1. *Distancia entre el prefijo "**faen**" y "**cadena**", almacenada en la celda $D[4, 6]$, más el coste de la operación $a \rightarrow \varepsilon$, resultando una distancia de 4.*
2. *Distancia entre "**faena**" y el prefijo "**caden**", almacenada en la celda $D[5, 5]$, más el coste de la operación $\varepsilon \rightarrow a$, resultando una distancia de 4.*
3. *Distancia entre los prefijos "**faen**" y "**caden**", almacenada en la celda $D[4, 5]$, más el coste de la operación $a \rightarrow a$, resultando una distancia de 2.*

La distancia resultante se corresponde con el coste de la operación de cambio $f \rightarrow c$, más el coste de la inserción $\varepsilon \rightarrow e$.

Es trivial comprobar que la complejidad temporal, del algoritmo de Wagner y Fischer es $O(n \times m)$ en el peor de los casos. En cuanto a la complejidad espacial; esta es $O(min\{n, m\})$. Si suponemos que la matriz $D[i, j]$ se rellena por columnas, al calcular las distancias de la columna j, $D[i, j]$, $\forall i$, basta con conocer únicamente las distancias de la columna anterior, $j - 1$. Tal como se puede observar en las expresiones utilizadas en el cálculo de las distancias, no es necesario disponer de los valores de todas las demás distancias $D[i, k]$, $\forall i$, con $k < j - 1$. En el caso de que se rellenase la matriz de distancias por filas el razonamiento sería análogo. Por lo tanto, siempre podrán calcularse las distancias parciales eligiendo el recorrido de la matriz que de lugar a las menores exigencias de espacio, resultando una complejidad espacial $O(min\{n, m\})$.

El algoritmo 3.1 calcula únicamente la matriz de distancias de edición. El resultado que ofrece nos indica el coste total de las operaciones de edición necesarias para transformar una cadena en la otra, pero no proporciona el conjunto de operaciones que originan esa distancia. Sin embargo, es posible extraer esa lista de operaciones a partir de la matriz de distancias parciales. Bastará con recorrer de forma inversa la matriz. Comenzando en la celda $D[n, m]$, se identifica, en cada paso, la celda a partir de la cual fue calculada la distancia almacenada en la celda actual y se guarda traza de la operación correspondiente. Se repite ese proceso con cada nueva celda hasta llegar a la distancia $D[0, 0]$. Al final, se habrá generado una lista con las operaciones de edición empleadas en una secuencia de operaciones de coste mínimo. En el caso del ejemplo 3.2, en la siguiente matriz de distancias se muestran, señaladas con un recuadro, las distancias parciales implicadas en el cálculo de la distancia final.

$D[i,j]$		c	a	d	e	n	a
	0	1	2	3	4	5	6
f	1	**1**	2	3	4	5	6
a	2	2	**1**	**2**	3	4	5
e	3	3	2	2	**2**	3	4
n	4	4	3	3	3	**2**	3
a	5	5	4	4	4	3	**2**

A partir de esta matriz, se obtiene la siguiente lista de operaciones de edición que dan lugar a la distancia calculada anteriormente:

Operación	Coste
$a \rightarrow a$	0
$n \rightarrow n$	0
$e \rightarrow e$	0
$\varepsilon \rightarrow d$	1
$a \rightarrow a$	0
$f \rightarrow c$	1

Una de las grandes ventajas del algoritmo de Wagner y Fischer es su flexibilidad. Dentro del marco que ofrece el algoritmo, se puede dar soporte a otros problemas relacionados con el procesamiento de cadenas de caracteres. En algunos casos simplemente bastará con modificar la función de coste. En otros casos, como el reconocimiento aproximado que presentaremos en el próximo apartado, será necesario modificar ciertos aspectos de algoritmo, pero sin afectar al esquema de funcionamiento general presentado anteriormente. Para finalizar esta sección describiremos distintos tipos de distancias entre cadenas [32] y veremos como el algoritmo descrito puede manejarlas simplemente modificando la definición de la función de coste de las operaciones:

Distancia de edición o distancia Levenshtein. Descrita en [30], es una versión restringida de la distancia entre cadenas que hemos venido manejando en este capítulo, ya que exige costes unitarios. Se permiten operaciones de inserción, borrado y cambio de caracteres, todos ellas con coste 1, excepto el cambio de caracteres idénticos, cuyo coste es 0.

Dado que se trata de costes unitarios, la distancia obtenida se corresponderá con el número mínimo de borrados, inserciones y cambios necesarios para transformar una cadena en otra. Es, además, una distancia simétrica, $d(x, y) = d(y, x)$, y verifica $0 \leq d(x, y) \leq max\{|x|, |y|\}$. Ello equivale a la consideración de una métrica discreta como distancia de edición.

Distancia Hamming. Permite únicamente operaciones de cambio de caracteres con coste 1. Para soportar esa restricción, basta con asignar un coste infinito a las operaciones de borrado e inserción, de modo que dichos tipos de operaciones nunca serán tenidos en cuenta. En este caso la distancia no es simétrica, verificando que $0 \leq d(x, y) \leq |x|$.

Tamaño de la subsecuencia común más larga. Se trata de un problema emparentado con el reconocimiento de patrones en cadenas. El objetivo es medir la longitud de la mayor secuencia de caracteres, no necesariamente contiguos, que son comunes a dos cadenas [33].

Se permiten únicamente borrados en inserciones con coste 1. De nuevo, para dar soporte a esta medida, se fuerza a que el coste de las operaciones de cambio de caracteres sea infinito. De esta forma, la distancia obtenida será menor cuanto mayor sea la longitud de la secuencia común, y viceversa. La distancia es simétrica y verifica la relación $0 \leq d(x, y) \leq |x| + |y|$.

3.3. Reconocimiento aproximado en cadenas

El problema del reconocimiento aproximado de cadenas es una extensión del problema clásico de reconocimiento de patrones en cadenas, en la cual se pretende hacer más flexible el concepto de distancia manejado con anterioridad, de forma que facilite su utilización práctica. En general, al hablar de este tipo de reconocimiento aproximado, se diferenciará entre la *cadena patrón* que especifica de forma aproximada una cadena o un conjunto de cadenas de interés, y la *cadena dato*, en la cual se desea hacer encajar la cadena patrón. El interés de este tipo de técnicas deriva de sus aplicaciones prácticas en multitud de aplicaciones que impliquen el manejo de cadenas de caracteres, como por ejemplo en la búsqueda y tratamiento de datos textuales o en el procesamiento de secuencias de ADN [26].

En esta sección revisaremos dos alternativa empleadas comúnmente para permitir la especificación aproximada de cadenas patrón y veremos como se puede manejar estos nuevos tipos de reconocimiento de cadenas en el marco del algoritmo de Wagner y Fischer [34]. Comenzaremos revisando una variante del reconocimiento de patrones clásico que permite la eliminación de prefijos de la cadena dato sin afectar a la distancia obtenida. La segunda alternativa que presentaremos será la inclusión de símbolos VLDC[1] en la cadena patrón. Estos símbolos permiten especificar partes de la cadena patrón que no son de interés para el proceso de reconocimiento de patrones y que, por lo tanto, podrán ser sustituidos libremente por porciones de la cadena dato.

3.3.1. Eliminación de prefijos en la cadena patrón

La primera posibilidad que revisaremos a la hora de definir el reconocimiento aproximado de patrones en cadenas consiste en realizar la correspondencia parcial de la cadena patrón con la cadena dato permitiendo que en ésta sean eliminados prefijos, sin que el coste de esa eliminación afecte a la distancia entre ambas cadenas. Intuitivamente se trata de encontrar la porción de la cadena dato más próxima a la cadena patrón. Generalmente el patrón se corresponderá con una cadena de pequeño tamaño que representa algún elemento de interés. La cadena dato tendrá, generalmente, una longitud mucho mayor y se tratará de encontrar en ella ocurrencias de subcadenas similares a la cadena patrón.

Veamos en primer lugar como se puede formular el problema en términos de distancias de edición. Dadas una cadena patrón $p \in \Sigma$ y una cadena dato $d \in \Sigma$. El problema del reconocimiento aproximado de cadenas trata de determinar, para cada prefijo de d de longitud j, la distancia entre la cadena patrón p y ese prefijo $d_{1..j}$, permitiendo la eliminación de prefijos de cualquier longitud dentro de esta última subcadena dato, $d_{1..i}$. En la práctica nos interesará únicamente la menor de esas distancias con respecto al patrón p. A partir de esa distancia se podrá identificar la porción de d cuya distancia con respecto a p sea menor. El siguiente ejemplo ilustra la idea de reconocimiento aproximado permitiendo la eliminación de prefijos.

Ejemplo 3.3.
Dada la cadena patrón p=*"cena" y la cadena dato* d=*"concatenación" y suponiendo costes unitarios para las operaciones de edición, la distancia resultante será 1. Esta es la distancia que existe entre la*

[1]Por *Variable Length Don't Care*. Para mantener la terminología usual en este tipo de trabajos, se mantendrá la denominación anglosajona *símbolo* VLDC.

cadena patrón "cena" y la subcadena de d *"concatena", sobre la que se elimina el prefijo "conca". La distancia resultante se obtiene de aplicar de la operaciones de cambio $c \rightarrow t$, de coste 1, a la cadena "cena" para dar "tena".*

Se puede modificar el algoritmo de Wagner y Fischer [34] para realizar este tipo de reconocimiento aproximado. Intuitivamente, la aproximación utilizada se basa en considerar todas las posiciones de la cadena dato, d, como potenciales puntos de inicio de la cadena buscada. Para ello, simplemente es necesario modificar la inicialización de la primera fila de la matriz de distancias $D[i, j]$. Así, todas las celdas $D[0, j]$ con $1 \leq j \leq |d|$ se inicializarán a 0. Cada una de estas celdas $D[0, j]$ representa la distancia entre ε y el prefijo $d_{1..j}$ de la cadena dato, d. Al ser 0 ese valor, el algoritmo de Wagner y Fischer se comportará como si la cadena dato comenzará realmente en dicha posición j, con lo que las distancias que se obtengan no tendrán en cuenta a los caracteres de d anteriores a la posición j. Por lo tanto, basta sustituir en el algoritmo 3.1 las líneas

for $j = 1$**to** m
  $D[0, j] = D[0, j-1] + \gamma(\varepsilon \rightarrow y_j)$
end for

por

for $j = 1$**to** m
  $D[0, j] = 0$
end for

Para obtener la distancia deseada, simplemente se debe encontrar el valor mínimo de las celdas de la última fila, $D[n, j]$, $\forall j$. De esta forma, el índice j de dicha celda marca el final de la subcadena $d_{1..j}$ de d más próxima a la cadena patrón p, permitiendo la eliminación de caracteres al comienzo de $d_{1..j}$. Terminaremos este apartado presentando un ejemplo muestra que esta idea.

Ejemplo 3.4.
Dadas la cadena patrón p=*"faena" y la cadena dato* d=*"encadenar", aplicando las modificaciones anteriores al algoritmo 3.1, se obtiene la siguiente matriz de distancias:*

$D[i,j]$		e	n	c	a	d	e	n	a	r
	0	0	0	0	0	0	0	0	0	0
f	1	1	1	1	1	1	1	1	1	1
a	2	2	2	2	1	2	2	2	1	2
e	3	2	3	3	2	2	2	3	2	2
n	4	3	2	3	3	3	3	2	3	3
a	5	4	3	3	3	4	4	3	**2**	3

La celda marcada con un recuadro almacena el valor de la distancia aproximada entre p *y* d. *El valor de dicha distancia, 2, se corresponde con la distancia entre el patrón "faena" y la subcadena "encadena", en la cual se elimina el prefijo "en" con coste 0.*

3.3.2. Inclusión de símbolos VLDC en la cadena dato

Una variación del problema clásico de reconocimiento aproximado sobre cadenas, es la inclusión en la cadena patrón de símbolos VLDC. Se trata de símbolos especiales, no pertenecientes al alfabeto de las cadenas, que se incluyen en el patrón p y que pueden ser sustituidos por cero o más caracteres de la cadena dato, d, sin afectar a la distancia final. La inclusión de estos símbolos VLDC permite definir patrones en donde partes de la cadena no son relevantes y pueden admitir múltiples variaciones. El ejemplo típico de símbolo VLDC es el carácter comodín "$*$", de uso común en los intérpretes de comandos de muchos sistemas operativos, que puede ser sustituido por cualquier secuencia de caracteres con un coste nulo, esto es, su sustitución no afecta a la distancia final.

El problema del reconocimiento aproximado de patrones con símbolos VLDC se puede formular en términos de distancia de edición del siguiente modo. Dada una cadena patrón p, que incluya símbolos VLDC y una cadena dato d, definiremos $\bar{p}$ como una cadena resultante de sustituir los símbolos VLDC de p por subcadenas de d. La distancia de edición en presencia de símbolos VLDC entre las cadena p y d se define como la distancia existente entre la cadena $\bar{p}$ derivada de p, que sea más próxima a d, y la propia cadena dato, d. El siguiente ejemplo ilustra el uso del símbolo VLDC "`*`", y como son las distancias resultantes.

Ejemplo 3.5.
Ante la cadena patrón p=*"`bu*ador`", y suponiendo un coste 1 por cada operación de edición, la distancia con respecto a la cadena dato "`buscador`" sería 0, sustituyendo el símbolo* VLDC *"`*`" por la subcadena "`sc`".*
Para las cadenas "`purificador`" y "`embajador`", las distancias serían 1 y 3, respectivamente. En el primer caso se sustituye el símbolo VLDC *por la subcadena "`rific`" y se cambia el carácter b por p. En el segundo caso se sustituye el símbolo* VLDC *por la subcadena "`j`", se cambia el carácter "`u`" por "`a`" y se insertan los caracteres "`e`" y "`m`".*

Como en el caso anterior se puede modificar el algoritmo 3.1 para dar soporte a este tipo de reconocimiento aproximado de patrones con VLDC. Veremos como se realiza dicha modificación para el VLDC "`*`". En primer lugar se debe definir el coste de las operaciones de edición que involucren al símbolo VLDC. Dado que pretendemos que el símbolo sea sustituido por subcadenas sin afectar a la distancia resultante, el coste de las operaciones de borrado y de cambio de dicho símbolo debe ser 0. Es decir, la función de coste γ debe asegurar que $\gamma(* \to \varepsilon) = 0$ y $\gamma(* \to d_j) = 0\ \forall j\ 1 \leq j \leq |d|$. Además, se deberá modificar la fórmula que se aplica en el bucle principal del algoritmo para calcular las distancias parciales. De este modo, se debe sustituir en el algoritmo 3.1 la expresión:

$$D[i,j] = min \begin{cases} D[i-1,j] + \gamma(x_i \to \varepsilon) \\ D[i,j-1] + \gamma(\varepsilon \to y_j) \\ D[i-1,j-1] + \gamma(x_i \to y_j) \end{cases}$$

por

$$D[i,j] = \begin{cases} min \left\{ \begin{array}{l} D[i-1,j] + \gamma(p_i \to \varepsilon) \\ D[i,j-1] + \gamma(\varepsilon \to d_j) \\ D[i-1,j-1] + \gamma(p_i \to d_j) \end{array} \right\} & \textbf{si } p_i \neq \texttt{"*"} \\ min \left\{ \begin{array}{l} D[i-1,j] + \gamma(p_i \to \varepsilon) \\ D[i,j-1] + \gamma(\varepsilon \to d_j) \\ D[i-1,j-1] + \gamma(p_i \to d_j) \\ D[i,j-1] \end{array} \right\} & \textbf{si } p_i = \texttt{"*"} \end{cases}$$

La explicación intuitiva de esta nueva fórmula es la siguiente. En caso de que el símbolo actual, p_i, de la cadena patrón no sea el símbolo VLDC, se emplean las fórmulas originales del algoritmo de Wagner y Fischer. Sin embargo, en el caso de que el símbolo actual del patrón sea el símbolo "`*`" y nos encontremos en la posición i de la cadena dato, d, lo que se hará será recuperar el mejor valor de distancia obtenido para una subcadena de d hasta ese momento. Los casos $D[i,j] = D[i-1,j] + \gamma(p_i \to \varepsilon)$, $D[i,j] = D[i,j-1] + \gamma(\varepsilon \to d_j)$ y $D[i,j] = D[i-1,j-1] + \gamma(p_i \to d_j)$ son equivalentes a los del algoritmo original, puesto que asumimos que $\gamma(* \to \varepsilon) = 0$ y $\gamma(* \to d_j) = 0$, $\forall j\ 1 \leq j \leq |d|$.

La situación realmente interesante se produce cuando $D[i,j] = D[i,j-1]$. En este caso se utiliza el valor de la distancia entre la porción actual de p y la subcadena $d_{1..j-1}$. Esto equivale a considerar que el carácter en la posición i de la cadena patrón es igual a d_j, con lo cual no afectará a la distancia en ese punto. Por lo tanto, lo que hace el algoritmo es simular la sustitución del símbolo "`*`" por el carácter actual, d_j, de la cadena dato d. Al igual que en el algoritmo original, la distancia resultante se almacena en la celda $D[n,m]$ de la matriz de distancias.

Finalizaremos esta sección presentando un ejemplo que muestra el funcionamiento de las modificaciones anteriores en un caso práctico.

Ejemplo 3.6.
Dadas la cadena patrón p=*"f*na"* *y la cadena dato* d=*"cadena"*, *aplicando las modificaciones anteriores al algoritmo 3.1, se obtiene la siguiente matriz de distancias:*

$D[i,j]$		c	a	d	e	n	a
	0	1	2	3	4	5	6
f	1	1	2	3	4	5	6
*	1	1	1	1	1	1	1
n	2	2	2	2	2	1	2
a	3	3	2	3	3	2	1

El valor de la distancia entre las cadenas p *y* d *es 1, que se corresponde con la distancia entre la cadena "fadena", resultado de sustituir el símbolo "*" del patrón por la subcadena "ade", y la cadena dato "cadena". En la matriz se muestran marcadas con un recuadro las distancias parciales que intervienen en cálculo de la distancia. A partir de ellas es directo deducir la porción de la cadena dato que sustituye el símbolo "*".*

Capítulo 4

Reconocimiento de patrones en árboles. Primeras aportaciones

En este capítulo se presenta el problema de la comparación de árboles y los primeros algoritmos propuestos para resolverlo. Si bien estas técnicas no son especialmente eficientes, si es interesante revisarlas, dado que es en estos primeros trabajos donde se definen las propiedades y conceptos básicos que constituyen la base de los algoritmos que se presentarán en los siguientes capítulos y que son la base del trabajo de la tesis.

Se comenzará con una definición del problema del reconocimiento de patrones sobre árboles y con la presentación de la estrategia de resolución utilizada, introduciendo el concepto de operaciones de edición y el de distancia de edición entre árboles. Se presentarán dos de las primeras aproximaciones al problema, el algoritmo propuesto por Selkow [37] y el de Tai [43] que, como se verá, se plantean como extensiones del algoritmo básico de comparación de cadenas de Wagner y Fischer [34]. Se prestará especial atención al trabajo de Tai por su relevancia en relación a las aportaciones posteriores.

4.1. Definición del problema

La primera parte de esta memoria se centra en el estudio de las técnicas de reconocimiento de patrones sobre árboles ordenados etiquetados. La aproximación clásica a la hora de abordar el problema de la comparación de árboles, se basa en la noción de *distancia de edición*, que será la que se siga en este trabajo. Para determinar la proximidad entre dos árboles se comienza definiendo un conjunto de *operaciones de edición* sobre los nodos de un árbol, que nos permitirán transformar un árbol en otro. A cada una de esas operaciones se les asociará un coste numérico. De esta forma, se podrá cuantificar la similaridad entre dos árboles, calculando la suma de los costes de las operaciones necesarias para transformar uno en otro.

Dado un conjunto de operaciones y una función que les asigna el coste correspondiente, el problema de la comparación de árboles etiquetados ordenados trata de determinar, para dos árboles ordenados T y T', la distancia entre T y T' medida como el menor coste de todas las posibles secuencias de operaciones de edición que transforman T en T'. La definición exacta del conjunto de operaciones de edición varía ligeramente entre los distintos autores. Las usuales son la inserción de un nodo, el borrado de un nodo y el cambio de la etiqueta de un nodo, que se corresponden con una generalización de las operaciones de inserción, borrado e intercambio sobre cadenas que fueron presentadas en el capítulo 3.

4.2. Algoritmo de Selkow

El algoritmo propuesto por Selkow [37] es una extensión del algoritmo de reconocimiento de cadenas de Wagner y Fischer [34], que pretende aplicar la solución propuesta por estos autores al problema de la comparación de árboles.

El algoritmo considera las mismas operaciones de edición básicas que Wagner y Fischer, en este caso aplicadas a nodos en lugar de a caracteres: inserción de un nodo, borrado de un nodo y cambio de la etiqueta de un nodo. Selkow añade una restricción importante, se exige que las operaciones de inserción y borrado sólo sean aplicadas en las hojas de los árboles.

Definición 4.1 (operaciones de edición de Selkow)**.**
Sea T un árbol ordenado y etiquetado y sea Σ el alfabeto de sus etiquetas. Se considera el siguiente conjunto de operaciones de edición *sobre ese árbol:*

- Cambio de etiqueta*. Se reemplazará la etiqueta $a \in \Sigma$ asociada a un nodo* u *del árbol T por otra etiqueta $b \in \Sigma$. Tal operación se notará como $a \to b$.*
- Borrado*. Se elimina del árbol T un nodo hoja* u *etiquetado como $a \in \Sigma$. La operación de borrado del nodo etiquetado como a, se denotará como $a \to \varepsilon$, siendo $\varepsilon \notin \Sigma$ un símbolo que representa a un nodo nulo.*
- Inserción*. Se añade al árbol T un nodo hoja etiquetado como $b \in \Sigma$. La notación empleada para representar esta operación será $\varepsilon \to b$.*

De esta forma, al incluir las restricciones sobre inserciones y borrados, sólo cuando la totalidad de los nodos descendientes de un nodo determinado hayan sido eliminados, se podrá borrar ese nodo. De modo análogo, sólo se podrán insertar nodos sin descendientes, es decir, hojas. Esta limitación obliga a que, en el caso de que se decida que un nodo no se debe borrar o insertar, los padres de ese nodo tampoco podrán ser borrados ni insertados. La distancia calculada por el método de Selkow representa el coste de transformar un árbol T en otro T' mediante la secuencia de operaciones que, respetando las restricciones sobre inserciones y borrados, ofrezcan el mínimo coste.

La aproximación seguida por el algoritmo es recursiva. En la primera llamada al algoritmo se tratan las raíces de los dos árboles T y T', realizando una llamada recursiva al algoritmo para cada par de subárboles hijos en ambas raíces. En concreto, dados como entrada un árbol patrón A y un árbol dato B, se calcula el coste de la operación cambio de etiquetas de la raíz de A a la raíz de B y se añade a dicho coste la distancia existente entre el conjunto de subárboles hijos de A; $A_1, A_2, \ldots, A_n$ y el conjunto de subárboles hijos de B; $B_1, B_2, \ldots, B_m$.

Para el cálculo de la distancia entre los conjuntos de subárboles hijos de cada par de árboles A y B, se utiliza una solución idéntica a la del algoritmo de Wagner y Fischer [34]. De hecho, se puede considerar este último como una subrutina del algoritmo de Selkow.

Para cada par de árboles A y B que recibe el algoritmo, se crea una matriz de distancias D de dimensión $(n+1) \times (m+1)$, siendo n el número de hijos de la raíz de A, y m el número de hijos de la raíz de B. La celda $D[0,0]$ almacena el coste del cambio de etiquetas entre la raíz del árbol A y la raíz de B. Cada entrada de esa matriz se calcula de forma análoga a la del algoritmo de Wagner y Fischer [34]. Se utilizan dos funciones auxiliares, *inserción(B_i)* y *borrado(A_i)*. La primera de ellas calcula el coste de insertar todos los nodos del subárbol B_i. La segunda, *borrado(A_j)*, calcula el coste de eliminar todos los nodos de un subárbol A_i. De este modo, la distancia que se almacena en la celda $D[i,j]$ se calcula tomando el mínimo coste de las tres posibles opciones para transformar el conjunto de subárboles $A_1, A_2, \ldots, A_i$ en el conjunto de subárboles $B_1, B_2, \ldots, B_j$:

- Sumar al coste de transformar $A_1, A_2, \ldots, A_i$ en $B_1, B_2, \ldots, B_{j-1}$, el coste de insertar el subárbol B_j, calculado por la subrutina *inserción*.
- Sumar al coste de transformar $A_1, A_2, \ldots, A_{i-1}$ en $B_1, B_2, \ldots, B_j$, el coste de borrar el subárbol A_i, calculado por la subrutina *borrado*.
- Sumar al coste de transformar $A_1, A_2, \ldots, A_{i-1}$ en $B_1, B_2, \ldots, B_{j-1}$, el coste de transformar el subárbol A_i en B_j, calculado por una llamada recursiva al algoritmo.

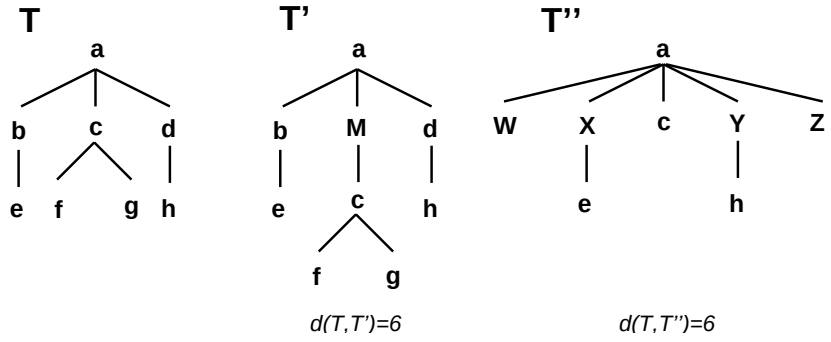


Figura 4.1: Ejemplos de distancias Selkow.

El cálculo de esta última opción supondrá realizar una llamada recursiva al algoritmo, pasándole como parámetro los dos nuevos subárboles, A_i y B_j. Como en el algoritmo para la distancia entre cadenas, al finalizar tendremos en la celda $D[n, m]$, la distancia entre los dos árboles. En el algoritmo 4.1 se muestra el pseudocódigo propuesto por Selkow.

Algoritmo 4.1 Distancia de Selkow

function *distancia*(A, B) **return** *real*;
Entrada: A: árbol patrón, B: árbol dato
Salida: distancia *Selkow* entre A y B

```
/*inicialización matriz*/
```

n = número de hijos de *raíz*(A)
m = número de hijos de *raíz*(B)
$D[0,0] = \gamma(raíz(A) \rightarrow raíz(B))$
for $i = 1$ **to** n **do**
  $D[i,0] = D[i-1,0] + borrado(A_i)$
end for
for $j = 1$ **to** m **do**
  $D[0,j] = D[0,j-1] + inserción(B_j)$
end for

```
/*cálculo de distancias*/
```

for $i = 1$ **to** n **do**
  for $j = 1$ **to** m **do**

$$D[i,j] = min \begin{cases} D[i,j-1] + \ inserción(B_j), \\ D[i-1,j] + \ borrado(A_i), \\ D[i-1,j-1] + \ distancia(A_i, B_j) \end{cases}$$ `/*llamada recursiva*/`

  end for
end for
return$(D[n,m])$;

a inconsistencias como la mostrada en la figura 4.1, y origina unos valores de distancias que no reflejan adecuadamente la proximidad entre dos árboles.

Suponiendo un coste unitario para cada operación de edición sobre los árboles de la figura 4.1. Las distancias entre T y T' y entre T y T'' son idénticas. En el caso de T y T', la distancia resulta de cambiar la etiqueta del nodo c por la etiqueta M, eliminar los nodos f y g de T e insertar los nodos c, f y g de T'. Para T y T'', se insertan los nodos W y Z, se elimina a f y a g de T y se cambia a b por X y a d por Y. Sin embargo, pese a las distancias obtenidas, es razonable pensar que el árbol T' es mucho más próximo a T que T'', puesto que sólo difieren en un nodo. Este ejemplo pone de manifiesto la principal debilidad del algoritmo de Selkow. El uso de un conjunto de operaciones de edición demasiado restrictivas, que no permite evaluar la similaridad entre árboles de forma adecuada.

4.3. Algoritmo de Tai

Kuo-Chung Tai [43] propone una solución al problema de la comparación de árboles utilizando un método que, cómo el de Selkow [37], es similar al algoritmo de comparación de cadenas de Wagner y Fischer [34]. Sin embargo, a diferencia de Selkow, Tai sigue una aproximación no recursiva, aunque también basada en programación dinámica.

La importancia de este trabajo se debe a que se trata del primer algoritmo que resuelve el problema de la comparación de árboles en tiempo polinomial. Además, en este artículo se incluye la definición de las tres operaciones básicas sobre árboles y la del concepto de distancia de edición que han sido utilizadas en los trabajos posteriores, con la ventaja adicional de no imponer las restricciones sobre borrados e inserciones que limitaban la aproximación de Selkow [37].

El algoritmo que propone este autor permite calcular la distancia entre dos árboles etiquetados y ordenados T y T' en un tiempo $O(|T| \times |T'| \times \text{prof}(T)^2 \times \text{prof}(T')^2)$ en el peor de los casos, siendo $|T|$ y $|T'|$ el número de nodos de cada árbol, y $\text{prof}(T)$ y $\text{prof}(T')$ su profundidad.

4.3.1. Operaciones de edición y distancia de edición

En su trabajo, Tai considera una ordenación en preorden de los nodos del árbol, de tal forma que siendo T un árbol, $t[i]$ representa al i-ésimo nodo de T en dicha ordenación. Partiendo de esa numeración, sobre los nodos de un árbol T es posible aplicar tres *operaciones de edición*: cambiar la etiqueta de un nodo, eliminar un nodo del árbol o insertar un nuevo nodo en el árbol, tal y como se muestra en la figura4.2. Formalmente, tenemos la siguiente definición.

Definición 4.2 (operaciones de edición sobre árboles)**.**
Sea T un árbol ordenado y etiquetado y sea Σ el alfabeto de sus etiquetas. Se considera el siguiente conjunto de operaciones *de edición sobre ese árbol:*

- Cambio de etiqueta*. Se reemplazará la etiqueta $a \in \Sigma$ asociada a un nodo* u *del árbol T por otra etiqueta $b \in \Sigma$. Tal operación se notará como $a \to b$.*
- Borrado*. Se elimina del árbol T un nodo* u *etiquetado como $a \in \Sigma$. Todos los hijos del nodo eliminado pasan a ser hijos del padre de dicho nodo. Esto es, el lugar que tenía el nodo eliminado entre sus hermanos es ocupado por la secuencia de sus hijos. La operación de borrado del nodo etiquetado como* a *se denotará como $a \to \varepsilon$, siendo $\varepsilon \notin \Sigma$ un símbolo que representa a un nodo nulo.*
- Inserción*. Se añade un nodo etiquetado como $b \in \Sigma$ como hijo de otro nodo de T etiquetado como $a \in \Sigma$. Un subconjunto consecutivo de los hijos del nodo* a *se convertirán en hijos del nodo insertado,* b*. Dicha secuencia de hijos dependerá, en cada caso, del contexto. La notación empleada para representar esta operación será $\varepsilon \to b$.*

El conjunto de todas estas operaciones de edición se denota OP*, definido como:*

$$\text{OP} \;=\; \{a \to b |\; a, b \in \Sigma \cup \{\varepsilon\} \setminus \{\varepsilon \to \varepsilon\}\}$$

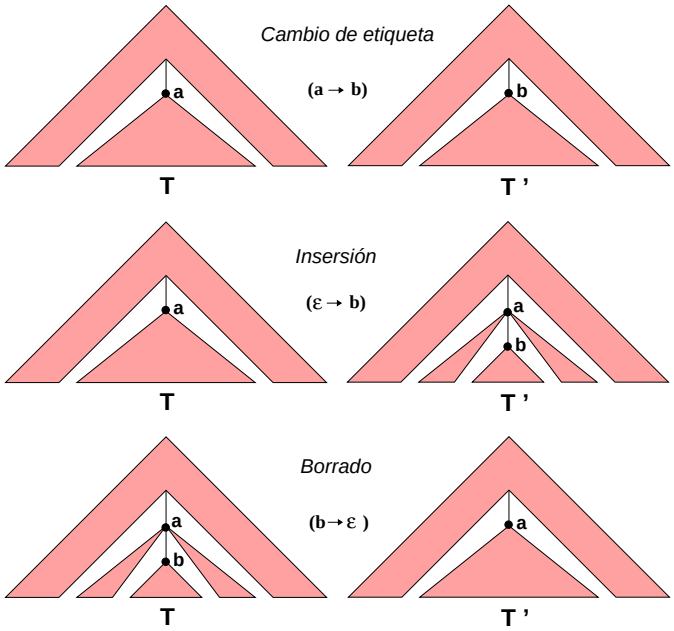


Figura 4.2: Operaciones de edición sobre árboles.

Tal como se ilustra en la figura 4.2, las operaciones de borrado supondrán la eliminación del arco asociado al nodo eliminado, que será sustituido por un conjunto de arcos que unirán al padre de dicho nodo con cada uno de los hijos que pudiera tener el nodo eliminado, manteniendo el orden existente entre esos hijos. En el caso de las inserciones se realiza la modificación complementaria. Se seleccionará un padre para el nuevo nodo y se añadirá un arco que los una. Además, parte de los hijos del nodo padre podrán pasar a ser hijos del nuevo nodo. La elección del nodo padre y de los hijos del nuevo nodo no se especifica directamente en la operación, sino que vendrá determinada por el contexto en el cual sea realizada. En nuestro caso, al tratarse del cálculo de distancias entre árboles, se considerará siempre que se eligen los nodos que den lugar al árbol más parecido al de partida. Normalmente, para transformar un árbol en otro no bastará con utilizar una única operación, siendo necesario introducir el concepto de *secuencia de edición.*

Definición 4.3 (secuencia de edición)**.**
Si T' es el árbol que se obtiene a partir de T después de aplicar una operación op$\in$ OP*, se dirá que* op transforma al árbol T en el árbol T' y se notará $T \Rightarrow_{op} T'$, o simplemente $T \Rightarrow T'$.

Una secuencia $S = op_1, op_2, \ldots, op_n$ de operaciones de edición, donde $op_i \in OP$, $\forall i$ $1 \leq i \leq n$, se denomina *secuencia de edición* si existe un conjunto de árboles T_i, $1 \leq i \leq n$ tales que la operación op_i transforma al árbol T_{i-1} en el árbol T_i, $\forall 1 \leq i \leq n$.

Además, en ese caso, se dirá que la secuencia S *transforma* al árbol $T = T_1$ en el árbol $T' = T_n$, lo que se notará como $T \Rightarrow^S T'$.

Al igual que en el caso de las cadenas, para obtener el valor de la distancia es necesario transformar la secuencia de operaciones necesarias para transformar un árbol en otro en un valor numérico. Esto se consigue asociando a cada operación de edición un número real no negativo, su coste. Para ello se utiliza una *función de coste*.

Definición 4.4 (función de coste)**.**
Se define una función de coste *como una función* $\gamma : OP \longrightarrow \mathbb{R}$ *que asocia a cada operación* $op \in \mathrm{OP}$ *un número real no negativo,* $\gamma(op)$*, que verifica las siguientes propiedades:*

1. $\gamma(op) \geq 0,\ \forall op \in \mathrm{OP}$
2. $\gamma(a \to a) = 0,\ \forall a \in \Sigma$
3. $\gamma(a \to b) = \gamma(b \to a),\ \forall a, b \in \Sigma \cup \{\varepsilon\}$
4. $\gamma(a \to c) \leq \gamma(a \to b) + \gamma(b \to c),\ \forall a, b, c \in \Sigma \cup \{\varepsilon\}$

Del mismo modo, dada una secuencia de operaciones de edición $S = op_1, op_2, \ldots, op_n$*. El* coste $\gamma(S)$ *de dicha secuencia se define como la suma del coste de cada una de las operaciones individuales que forman parte de ella, formalmente será:*

$$\gamma(S) = \sum_{i=1}^{n} \gamma(op_i)$$

Tal y como se muestra en la definición, se desea que la función γ verifique las propiedades de una métrica. De entre esas propiedades es especialmente importante la última, conocida como desigualdad triangular. Esta propiedad asegura que si un árbol se puede obtener directamente con una sola operación el coste de esa operación será menor o igual que el coste de dos operaciones diferentes que den lugar a ese mismo árbol. También se puede comprobar que la extensión de la función γ a secuencias de operaciones mantiene esas propiedades. En este caso, la desigualdad triangular garantiza que, cuando un árbol se pueda obtener por distintas secuencias de operaciones, una de ellas, al menos, tendrá un coste menor o igual que el de todas las demás. La consecuencia directa es que siempre existirá una secuencia de operaciones cuyo coste sea menor que el de cualquier otra. De todo esto, se llega a la definición de *distancia de edición*, $d(T, T')$, entre los árboles T y T'.

Definición 4.5.
Dados dos árboles ordenados y etiquetados T *y* T' *y una función de coste* γ*, se define la* distancia de edición entre T y T'*,* $d(T, T')$*, de la siguiente forma:*

$$d(T, T') = min\{\gamma(S) | S \text{ es una secuencia de operaciones de edición que transforma } T \text{ en } T'\}$$

Como ocurría en el caso de la distancia entre cadenas, se cuantifica la similaridad entre dos árboles, T y T', como el mínimo coste de todas las posibles secuencias de operaciones de edición válidas que transforman un árbol en el otro. Como se verá en la siguiente sección para calcular la distancia no será necesario estudiar todas las posibles secuencias de operaciones. Bastará con centrarse en un tipo especial de secuencias con unas características determinadas que se describirán a continuación.

4.3.2. Concepto de correspondencia

De forma análoga a las trazas en el caso de la distancia entre cadenas, se usa el concepto de *correspondencia entre árboles* para referirnos a una estructura que permite representar las operaciones de edición necesarias para transformar un árbol T en otro T'. Este concepto es la base del algoritmo de Tai [43], puesto que permite reducir el cálculo de la distancia de edición a la búsqueda de correspondencias de coste mínimo. Formalmente el autor define una correspondencia entre dos árboles como sigue:

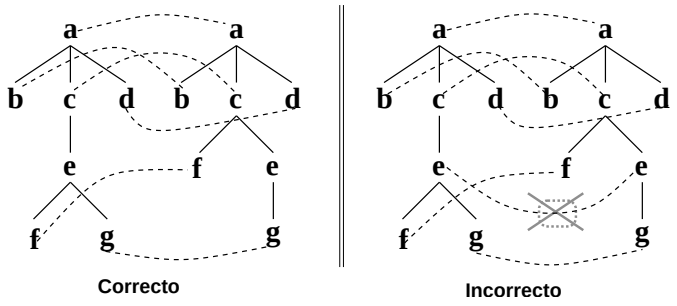


Figura 4.3: Ejemplos de correspondencias correctas e incorrectas.

Definición 4.6 (correspondencia entre árboles)**.**
Se define una correspondencia *entre dos árboles T y T' como una $3-upla$ (M,T,T'), donde M es un conjunto de pares de la forma:*

$$M = \{(i,j) \mid 1 \leq i \leq |T|,\ 1 \leq j \leq |T'| \text{ y } \exists\, t[i] \rightarrow t'[j] \text{ en la secuencia que transforma } T \text{ en } T'\}$$

Verificando que, dados los pares (i_1,j_1) y (i_2,j_2) en M :

1. *$i_1 = i_2$ si y sólo si $j_1 = j_2$.*
2. *$i_1 < i_2$ si y sólo si $j_1 < j_2$.*
3. *$t[i_1]$ es un ancestro (resp. descendiente) de $t[i_2]$ si y sólo si $t'[j_1]$ es un ancestro (resp. descendiente) de $t'[j_2]$.*

Un par $(i,j) \in M$ se denomina conexión entre $t[i]$ y $t'[j]$. *Además, se dirá que esos nodos $t[i]$ y $t'[j]$* participan *en la correspondencia M o están* afectados *por la correspondencia M.*

La propiedad *1* asegura que cada nodo de T y T' participa como máximo una vez en la correspondencia M. Con las propiedades *2* y *3* se asegura que en T' se mantiene el orden que existía entre los nodos de T. La propiedad *2* asegura que se mantiene el orden de izquierda a derecha entre nodos hermanos y la propiedad *3* mantiene el orden jerárquico entre ancestros y descendientes.

Frecuentemente se emplea una representación gráfica de las correspondencia que facilita su interpretación. Gráficamente, los pares (i,j) que forman una correspondencia M se identifican con un conjunto de líneas que unen los nodos de los árboles T y T'. Se dibujará una línea por cada par $(i,j) \in M$, uniendo a los nodos $t[i]$ y $t'[j]$, tal y como se muestra en la figura 4.3. Cada una de estas líneas significa que la etiqueta del nodo $t[i]$ deberá de cambiarse por la del nodo $t'[j]$, si $t[i] \neq t'[j]$. Los nodos de T que no tengan un línea asociada deberán borrarse, y los nodos de T' sin línea asociada deberán insertarse. Un resultado importante derivado de las propiedades *2* y *3* de la definición de correspondencia, es que no es posible que en la representación gráfica se crucen líneas entre nodos de distinta profundidad, tal y como se muestra en figura 4.3. En esta figura la secuencia de operaciones de la derecha no constituye una correspondencia correcta, puesto que no respeta el orden jerárquico existente en el árbol de partida entre los nodos f y e.

Una vez definido el concepto de correspondencia y revisada su representación gráfica, es trivial extraer la secuencia de operaciones de edición representadas por una correspondencia M entre dos árboles T y T'. Basta tener en cuenta que los pares $(i,j) \in M$ representan operaciones de cambio de etiqueta, $t[i] \rightarrow t'[j]$. Los nodos de T que no aparecen en ningún par de M serán eliminados, lo que supone una operación $t[k] \rightarrow \varepsilon$. Y los nodos de T' no afectados por la correspondencia se insertarán, generando la inserción $\varepsilon \rightarrow t'[l]$. Utilizando esta transformación se define el *coste de una correspondencia*.

Definición 4.7 (coste de una correspondencia)**.**
Sea M una correspondencia entre T y T' y sean I (resp. J) el conjunto de nodos de T (resp. T') no afectados por la correspondencia. Definimos el coste de M *como:*

$$coste(M) = \sum_{(i,j)\in M} \gamma(t[i] \to t'[j]) + \sum_{i\in I} \gamma(t[i] \to \varepsilon) + \sum_{j\in J} \gamma(\varepsilon \to t'[j])$$

Es decir, el coste de M es el coste de las sustituciones, mas el coste de los borrados, mas el coste de las inserciones realizadas en el seno de M. El siguiente paso será relacionar el coste de una correspondencia con la distancia entre dos árboles. En primer lugar, se presenta el siguiente lema, que proporciona la posibilidad de componer correspondencias.

Lema 4.1 (composición de correspondencias)**.**
Sea M_1 una correspondencia entre T_1 y T_2, y sea M_2 una correspondencia entre T_2 y T_3, entonces :

1. *$M_1 \circ M_2 = \{(i,k) \mid \exists i$ tal que $(i,j) \in M_1)$ y $(j,k) \in M_2\}$ es una correspondencia.*
2. *$coste(M) \leq coste(M_1) + coste(M_2)$.*

Demostración.
Demostraremos cada una de las tesis del lema por separado:

- Tesis *1*

  Es necesario comprobar que $M_1 \circ M_2$ verifica las tres propiedades de una correspondencia. Sean $(i_1,k_1),(i_2,k_2) \in M_1 \circ M_2$, dos pares de la correspondencia compuesta. Por la definición dada en *1*, existen dos nodos j_1 y j_2 en T_2, tales que $(i_1,j_1),(i_2,j_2) \in M_1$ y $(j_1,k_1),(j_2,k_2) \in M_2$. Además se verifica lo siguiente:

  - Dado que $(i_1,j_1),(i_2,j_2) \in M_1$, se cumple que $i_1 = i_2$ *sii* $j_1 = j_2$. Del mismo modo, como $(j_1,k_1),(j_2,k_2) \in M_2$, tenemos que $j_1 = j_2$ *sii* $k_1 = k_2$. De estas dos relaciones se llega a que $i_1 = i_2$ *sii* $k_1 = k_2$, con lo que se verifica la propiedad *1* de las correspondencias.
  - Como el caso anterior, tenemos que $(i_1,j_1),(i_2,j_2) \in M_1$, por lo tanto $i_1 < i_2$ *sii* $j_1 < j_2$. Para $(j_1,k_1),(j_2,k_2) \in M_2$ se verifica $(j_1 < k_1$ *sii* $j_2 < k_2)$. Combinando ambas desigualdades se verifica la propiedad *2* de las correspondencias. Esto es, $i_1 < i_2$ *sii* $k_1 < k_2$.
  - El mismo razonamiento se aplica para demostrar el cumplimiento de la propiedad *3* de las correspondencias. Puesto que $(i_1,j_1),(i_2,j_2) \in M_1$, $t_1[i_1]$ es un ancestro (resp. descendiente) de $t_1[i_2]$ *sii* $t_2[j_1]$ es un ancestro (resp. descendiente) de $t_2[j_2]$. De la misma forma, al ser $(j_1,k_1),(j_2,k_2)$ pares de M_2, $t_2[j_1]$ es un ancestro (resp. descendiente) de $t_2[j_2]$ *sii* $t_3[k_1]$ es un ancestro (resp. descendiente) de $t_3[k_2]$, con lo cual se cumple la propiedad exigida.

  Así se llega a que para los pares $(i_1,j_1),(i_2,j_2) \in M_1 \circ M_2$ se cumplen las propiedades de una correspondencia, y, por lo tanto, a que $M_1 \circ M_2$ es una correspondencia.

- Tesis *2*

  Para demostrar esta condición se utiliza el hecho de que la función de coste γ verifica las propiedades de una métrica, en particular la desigualdad triangular, $\gamma(a \to c) \leq \gamma(a \to b) + \gamma(b \to c)$. Tenemos que M_1 es una correspondencia de T_1 a T_2, y M_2 es una correspondencia de T_2 a T_3. Siendo $M_1 \circ M_2$ la composición de esas correspondencias, y I y J sus conjuntos de inserciones y borrados asociados, para cada nodo i del árbol T_1 y cada nodo j de T_3 pueden darse tres casos: $(i,j) \in M_1 \circ M_2$, $i \in I$ ó $j \in J$. Cada uno de esos casos se corresponde con una operación de edición $x \to y$, donde x e y son nodos de T_1 ó T_3, respectivamente, o son ε. En cualquier caso, dado que γ es una métrica, se deberá cumplir

  $$\gamma(x \to y) \leq \gamma(x \to z) + \gamma(z \to y)$$

siendo z un nodo de T_2 afectado por las correspondencias M_1 ó M_2, o siendo ε si x ó y pertenecen a los conjuntos de borrados o inserciones derivados de M_1 y M_2. En resumen, toda operación de edición representada en la correspondencia $M_1 \circ M_2$, se asociará con una o dos operaciones derivadas de M_1 y M_2. La desigualdad triangular asegura que el coste de cada operación inducida por la composición de correspondencias será menor o igual al coste de las operaciones inducidas por las correspondencias que se unen, tal y como señala la condición *2*.

□

La composición de correspondencias nos permitirá unir dos correspondencias o separar una correspondencia en dos *subcorrespondencias*. Sobre la primera de estas posibilidades se demuestra el lema 4.2, que afirma que para cualquier secuencia de operaciones es posible encontrar una correspondencia cuyo coste sea menor o igual al de dicha secuencia.

Lema 4.2.
Para toda secuencia de operaciones de edición $S = op_1, op_2, \ldots, op_n$, que transforma un árbol T en otro T', existe una correspondencia M de T a T', tal que $coste(M) \leq \gamma(S)$.

Demostración.
El lema puede probarse por inducción en la longitud, n, de la secuencia S, aplicando la composición de correspondencias.

- Caso $n = 1$.

  Se tiene una única operación de edición, op_1. Este caso es trivial ya que al tratarse de una única operación siempre se cumplen las tres propiedades de las correspondencias. Con lo que $coste(M) = \gamma(S)$.

- Caso $n - 1$.

  Supuesto cierto para toda secuencia de longitud menor que n.

- Caso n.

  Para el caso general tenemos $S = op_1, op_2, \ldots, op_n$. Se define la secuencia $S_1 = op_1, op_2, \ldots, op_{n-1}$, que tendrá una correspondencia asociada M_1 con $coste(M_1) \leq \gamma(S_1)$. Para la operación op_n se define la correspondencia M_2. Aplicando el lema 4.1, la composición de esas dos correspondencias verifica :

  $$coste(M) = coste(M_1 \circ M_2) \leq coste(M_1) + coste(M_2) \leq \gamma(S_1) + \gamma(op_n) = \gamma(S)$$

  con lo que queda demostrado el lema.

□

Como hemos visto, para toda secuencia de operaciones existe una correspondencia con un coste menor o igual. En concreto, en nuestro caso nos interesa la secuencia que da lugar a la menor distancia entre dos árboles y su correspondencia asociada. El siguiente teorema relaciona la distancia entre dos árboles con la correspondencia de coste mínimo entre ellos.

Teorema 4.1.
Sean T y T' dos árboles, entonces:

$$d(T, T') = min\{coste(M) |\ M \text{ es una correspondencia de } T \text{ en } T'\}$$

Demostración.
La demostración es directa a partir de la definición de $d(T, T')$ y del lema anterior. □

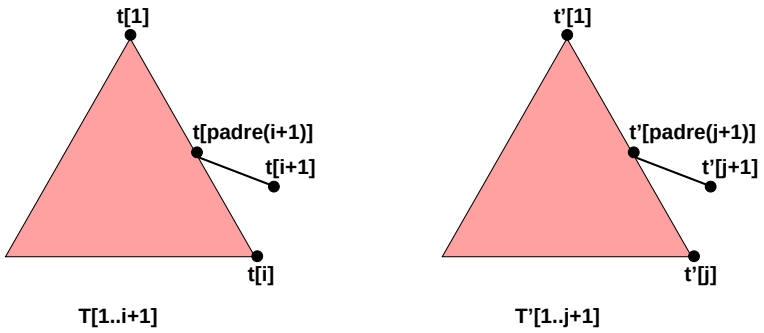


Figura 4.4: Distancia entre $T[1..i+1]$ y $T'[1..j+1]$.

Una consecuencia directa de este resultado, que será la base del algoritmo que propone Tai, es que para calcular la distancia entre dos árboles, bastará con encontrar una correspondencia de coste mínimo entre ellos. En ese caso, la distancia entre los árboles será igual al coste de esa correspondencia.

Antes de continuar, es necesario ampliar la notación manejada hasta ahora. Tal y como se señala en el capítulo 2, $T[i_1..i_2]$ representa al subárbol extraído de T, formado por los nodos $t[i_1], t[i_1+1], \ldots, t[i_2-1], t[i_2]$. Se empleará la notación $D(i,j)$ para referirse a la distancia entre el subárbol $T[1..i]$ y el subárbol $T'[1..j]$, es decir :

$$D(i,j) = d(T[1..i], T'[1..j])$$

Para simplificar la notación, se empleará $min\_m(i,j)$ para referirse al coste de la correspondencia de coste mínimo entre $T[1..i]$ y $T'[1..j]$ en la que se incluye el par (i,j). Formalmente tenemos :

Definición 4.8 (distancia $min\_m(i,j)$)**.**
$min\_m(i,j)$=$min\{coste(M) |\ M$ *es una correspondencia entre* $T[1..i]$ *y* $T'[1..j]$ *con* $(i,j) \in M\}$

Una vez establecida la relación entre correspondencia de coste mínimo y distancia de edición y utilizando la nueva notación, el siguiente teorema proporciona la forma de calcular la distancia $D(i+1,j+1)$ a partir de las distancias $D(i,j+1)$ y $D(i+1,j)$ y de $min\_m(i+1,j+1)$.

Teorema 4.2 (cálculo de $D(i+1,j+1)$)**.**
Sean T *y* T' *dos árboles, entonces:*

$$D(i+1,j+1) = min \begin{cases} D(i,j+1) + \gamma(t[i+1] \rightarrow \varepsilon), \\ D(i=1,j) + \gamma(\varepsilon \rightarrow t'[j+1]), \\ min\_m(i+1,j+1) \end{cases}$$

$$\forall i,j\ ,\ 1 \leq i \leq |T|,\ 1 \leq j \leq |T'|$$

Demostración.
La demostración se apoya en la figura 4.4, donde se muestran los árboles $T[1..j+1]$ y $T'[1..j+1]$. Supondremos que M es una correspondencia de coste mínimo entre $T[1..i+1]$ y $T'[1..j+1]$. Por el teorema 4.1 tenemos que $coste(M) = D(i+1,j+1)$. Pueden diferenciarse tres casos :

- Caso 1: $t[i+1]$ no está afectado por M

  En este caso la mejor opción es eliminar $t[i+1]$, resultando $coste(M) = D(i,j+1) + \gamma(t[i+1] \rightarrow \varepsilon)$. Lo que se corresponde con el coste de transformar $T[1..j]$ en $T'[1..j+1]$, mas el coste de eliminar $t[i+1]$.

- Caso 2: $t'[j+1]$ no está afectado por M

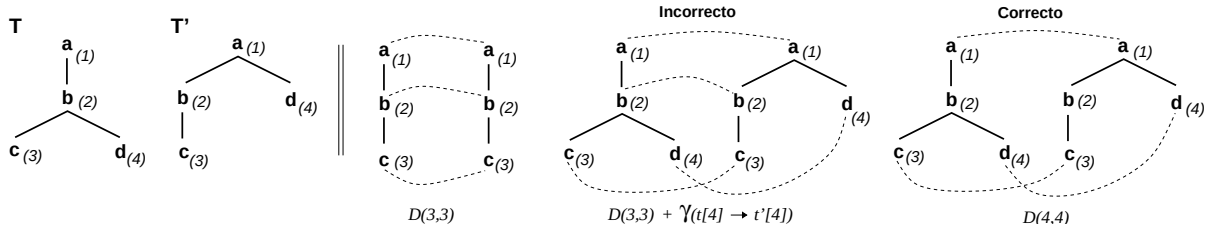


Figura 4.5: Cálculo distancias y orden jerárquico.

En este caso, la mejor opción es insertar $t'[j+1]$, resultando $coste(M) = D(i+1, j+) + \gamma(\varepsilon \rightarrow t'[j+1])$. Lo que se corresponde con el coste de transformar $T[1..i+1]$ en $t'[1..j]$, mas el coste de insertar $t'[j+1]$.

- Caso 3: $t[i+1]$ y $t'[j+1]$ están afectados por los pares $(i+1, q)$ y $(p, j+1)$ de M

  En este caso se demostrará que $t[i+1]$ y $t'[j+1]$ están incluidos en el mismo par de la correspondencia M, esto es, $i+1 = p$, y $j+1 = q$.

  - Por las propiedades de la correspondencia, tenemos que $i+1 = p$ *sii* $q = j+1$, y $i+1 > p$ *sii* $q > j+1$.
  - Por tratarse de los árboles $T[1..i+1]$ y $T'[1..j+1]$, tenemos que $1 \leq p \leq i+1$, y $1 \leq q \leq j+1$.

  Combinando ambas desigualdades, se llega a que $i+1 = p$, y $j+1 = q$, es decir, que $t[i+1]$ y $t'[j+1]$ están incluidos en el mismo par de M. De esta forma, puesto que $(i+1, j+1) \in M$, tenemos que $coste(M) \leq min\_m(i+1, j+1)$.

Dado que sólo son posibles estos tres casos a la hora de obtener la distancia $D(i+1, j+1)$, bastará con tomar la menor de ellas, con lo que queda demostrado el teorema. □

En todas las demostraciones y razonamientos que realicemos a partir de este momento, vamos a suponer que las raíces de los árboles T y T' tienen la misma etiqueta y que éstas no son cambiadas durante las operaciones de edición. De esta forma, todas las correspondencias entre T y T' contendrán al par $(1,1)$ y $D(1,1) = 0$. Esta suposición es correcta, ya que en cualquier caso se podrían añadir dos raíces ficticias con etiquetas idénticas. Sobre esta base, el siguiente corolario se deriva directamente del teorema anterior.

Corolario 4.2.1.
Sean T y T' dos árboles, suponiendo que las raíces de ambos árboles son idénticas, entonces:

$$\begin{array}{lcll} D(1,1) & = & 0 & \\ D(i,1) & = & \sum_{k=2}^{i} \gamma(T[k] \rightarrow \varepsilon),\ 1 \leq i \leq |T| & \textit{(Se eliminan todos menos la raíz)} \\ D(1,j) & = & \sum_{k=2}^{j} \gamma(\varepsilon \rightarrow T'[k]),\ 1 \leq j \leq |T'| & \textit{(Se insertan todos menos la raíz)} \end{array}$$

Demostración.
Trivial a partir del teorema 4.2. □

4.3.3. Cálculo de la distancia $min\_m(i, j)$

El teorema 4.2 proporciona la forma de calcular $D(i+1, j+1)$ a partir de $D(i, j+1)$, $D(i+1, j)$ y $min\_m(i+1, j+1)$. Sin embargo, como se muestra en el ejemplo 4.1, el cálculo de min_m no es trivial. La causa de esta complejidad es que se deben tener en cuenta las restricciones que imponen las correspondencias sobre la ordenación entre hermanos y, muy especialmente, sobre el mantenimiento del orden jerárquico.

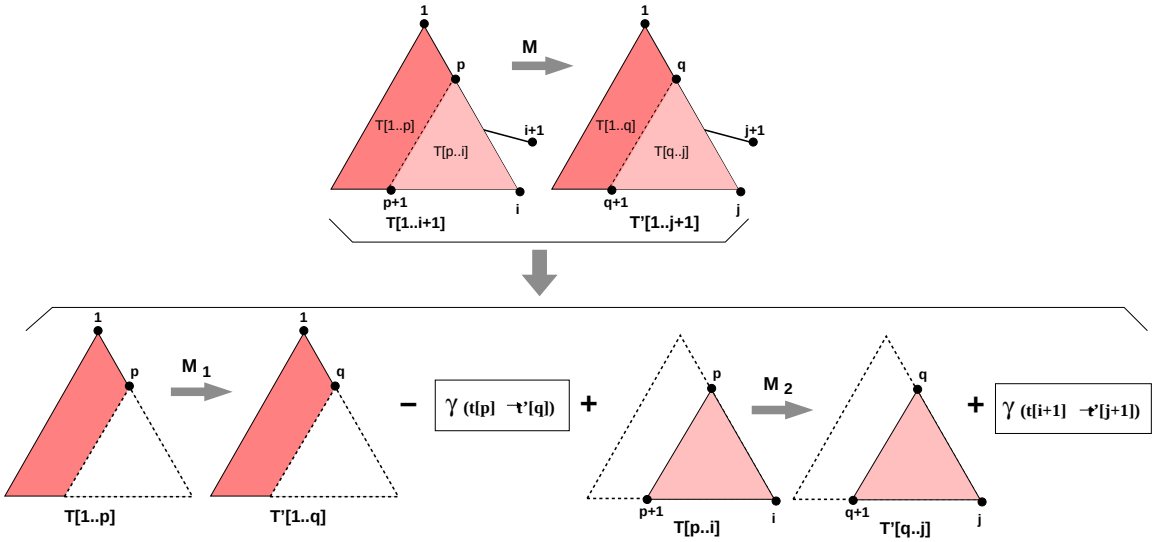


Figura 4.6: División del cálculo del coste de una correspondencia.

Ejemplo 4.1.
En la figura 4.5 se muestra un ejemplo de como afecta el mantenimiento del orden jerárquico al cálculo de las distancias. Dados los dos árboles T y T' mostrados en la figura, la distancia $D(3,3)$ es 0, resultando una correspondencia perfecta entre los subárboles $T[1.,3]$ y $T'[1.,3]$.

Si se pretendiera aplicar la misma idea que emplea el algoritmo de comparación de cadenas de Wagner y Fischer [34] se llegaría a una situación incorrecta. Al tratar de obtener la distancia $D(4,4)$ a partir de $D(3,3)$ y del coste de la operación de cambio de etiquetas $t[4] \rightarrow t'[4]$, obtendríamos una distancia final igual a 0. Sin embargo, como se muestra en la figura 4.5, las operaciones de edición que dan lugar a esa distancia no forman una correspondencia válida. Dichas operaciones no mantienen el orden ancestro-descendiente entre los nodos $t[2]$ y $t[4]$, es decir entre los nodos b *y* d *de T. La correspondencia de coste mínimo para $D(4,4)$ se muestra en el extremo derecho de la figura y su coste es 2, que se corresponde con el coste de las operaciones $t[2] \rightarrow \varepsilon$ y $\varepsilon \rightarrow t'[2]$.*

Como pone de manifiesto el ejemplo anterior, no es posible en general obtener la distancia $D(i+1, j+1)$ a partir de $D(i, j+1)$, $D(i+1, j)$ y $D(i, j)$; como sucedía en el algoritmo de comparación de cadenas de Wagner y Fischer [34]. Es por esto por lo que se hace necesario introducir la distancia *min_m* y un mecanismo para identificar la correspondencia correcta que la genere. En lo que resta de este apartado se presentan las bases formales de un método para calcular $min\_m(i+1, j+1)$ en un tiempo polinomial, utilizando para ello, la composición de correspondencias. El siguiente lema muestra como es posible descomponer una correspondencia en dos subcorrespondencias entre dos porciones de los árboles T y T'. Además, se demuestra que para que esa correspondencia sea de coste mínimo, ambas subcorrespondencias deberán serlo también. La figura 4.6 ilustra el resultado del lema.

Lema 4.3 (división de correspondencias)**.**
Sean T y T' árboles y sean $t[p]$ y $t'[q]$ ancestros de $t[i+1]$ y $t'[j+1]$ respectivamente. Sea M una correspondencia entre $T[1..i+1]$ y $T'[1..j+1]$ en la que estén incluidos los pares (p,q) y $(i+1, j+1)$. Se definen los subconjuntos de M, M_1 y M_2, de la siguiente forma :

- $M_1 = \{(m,n) | (m,n) \in M,\ 1 \leq m \leq p,\ 1 \leq n \leq q\}$ *(Subcorrespondencia de las raíces a $t[p]$ y $t'[q]$)*
- $M_2 = \{(m,n) | (m,n) \in M,\ p \leq m \leq i,\ q \leq n \leq j\}$ *(Subcorrespondencia desde $t[p]$ y $t'[q]$ a $t[i]$ y $t'[j]$)*

Entonces, las correspondencias M_1 y M_2 así definidas verifican :

1. *M_1 es una correspondencia entre $T[1..p]$ y $T'[1..q]$,*
   M_2 es una correspondencia entre $T[p..i]$ y $T'[q..j]$,
   $M = M_1 \cup M_2 \cup \{(i+1, j+1)\}$ y
   $coste(M) = coste(M_1) + coste(M_2) - \gamma(t[p] \to t'[q]) + \gamma(t[i+1] \to t'[j+1])$.

2. $coste(M) = min\{coste(M')|$ *M' es una correspondencia entre $T[1..i+1]$ y $T'[1..j+1]$, tal que $(p,q),(i+1,j+1) \in M'\}$*

   si y sólo si:

   a) $coste(M_1) = min\{coste(M_1')|$ *M_1' es una correspondencia entre $T[1..p]$ y $T'[1..q]$ tal que $(p,q) \in M_1'\}$*

   b) $coste(M_2) = min\{coste(M_2')|$ *M_2' es una correspondencia entre $T[p..i]$ y $T'[q..j]$ tal que $(p,q) \in M_2'$ y $M_2' \cup \{(i+1, j+1)\}$ es una correspondencia}*

Demostración.
La demostración se apoya en la figura 4.7. En ella se muestran los árboles $T[1..i+1]$ y $T'[1..j+1]$, la subcorrespondencia M_1 entre $T[1..p]$ y $T'[1..q]$ sombreada en color oscuro, y la subcorrespondencia M_2 entre $T[p..i]$ y $T'[q..j]$, sombreada en color claro.

- Propiedad *1*

  La demostración de esta primera propiedad se basa en el hecho de que la división se realiza utilizando el par $(p,q) \in M$ y que este es el único par común a M_1 y M_2. La definición de correspondencia asegura que todos los nodos de $T[1..i+1]$ (resp. $T'[1..j+1]$) afectados por M_1 están en $T[1..p]$ (resp. $T'[1..q]$). Del mismo modo, los nodos de $T[1..i+1]$ (resp. $T'[1..j+1]$) afectados por M_2 están en $T[p..i]$ (resp. $T'[q..j]$). Puesto que M es una correspondencia y sus pares cumplen la condición de mantener el orden jerárquico, es trivial comprobar que M_1 y M_2 también son correspondencias correctas.

  A partir de la construcción de M_1 y M_2, se llega de forma directa a que $M = M_1 \cup M_2 \cup \{(i+1, j+1)\}$, puesto que $(i+1, j+1)$ es el único par de M que no está en M_1 ni en M_2. Respecto a la relación entre los costes,

  $$coste(M) = coste(M_1) + coste(M_2) - \gamma(t[p] \to t'[q]) + \gamma(t[i+1] \to t'[j+1])$$

  Se obtiene directamente de la expresión anterior. Simplemente hay que tener en cuenta que, puesto que (p,q) es el único que par que está incluido a la vez en M_1 y M_2, su coste $\gamma(t[p] \to t'[q])$ debe ser descontado de la suma de los costes de las dos subcorrespondencias.

- Propiedad *2* Se demostrarán los dos sentidos de la doble implicación:

  "$\Leftarrow$"

  Partimos de que es cierto que

  $$coste(M) = min\{coste(M')| \quad M' \text{ es una correspondencia entre } T[1..i+1] \text{ y } T'[1..j+1], \text{ tal que } (p,q),(i+1,j+1) \in M'\}$$

  Suponiendo que sea cierta la relación,

  $$coste(M_1) > min\{coste(M_1')| \quad M_1' \text{ es una correspondencia entre } T[1..p] \text{ y } T'[1..q] \text{ tal que } (p,q) \in M_1'\}$$

  para esa valor mínimo debe existir una correspondencia $\tilde{M}_1$ entre $T[1..p]$ y $T'[1..q]$ con $(p,q) \in \tilde{M}_1$. De ser así, aplicando la propiedad *1*, con $\tilde{M}_1$ y M_2 podríamos formar una correspondencia $\tilde{M}$ verificando $coste(\tilde{M}) < coste(M)$, con lo que se contradice la suposición inicial.

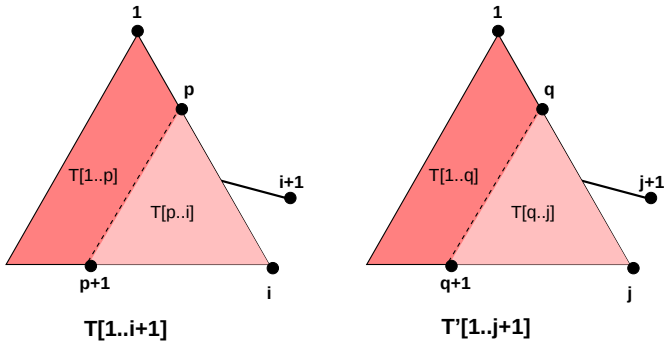


Figura 4.7: División en subcorrespondencias.

Un razonamiento análogo puede realizarse para el caso:

$$coste(M_2) > min\{coste(M_2')| \quad M_2' \text{ es una correspondencia entre } T[p..i] \text{ y } T'[q..j] \text{ tal que } (p,q) \in M_2'\}$$

Y, por lo tanto, la implicación "⇐" es correcta.

"⇒"

Supondremos que se cumple lo siguiente:

$$coste(M_1) = min\{coste(M_1')| \quad M_1' \text{ es una correspondencia entre } T[1..p] \text{ y } T'[1..q] \text{ tal que } (p,q) \in M_1'\}$$

y

$$coste(M_2) = min\{coste(M_2')| \quad M_2' \text{ es una correspondencia entre } T[p..i] \text{ y } T'[q..j] \text{ tal que } (p,q) \in M_2'\}$$

Supondremos que M no es una correspondencia de coste mínimo. Esto es:

$$coste(M) > min\{coste(M') \quad M' \text{ es una correspondencia entre } T[1..i+1] \text{ y } T'[1..j+1], \text{ tal que } (p,q),(i+1,j+1) \in M'\}$$

Existirá, entonces, otra correspondencia $\tilde{M}$ que si lo sea. Aplicando la primera propiedad de este lema, $\tilde{M}$ puede dividirse utilizando el par (p,q) en $\tilde{M}_1$ y $\tilde{M}_2$. Ahora bien, la única opción para que $coste(\tilde{M}) = coste(\tilde{M}_1) + coste(\tilde{M}_2) - \gamma(t[p] \to t'[q]) + \gamma(t[i+1] \to t'[j+1])$ sea menor que $coste(M) = coste(M_1) + coste(M_2) - \gamma(t[p] \to t'[q]) + \gamma(t[i+1] \to t'[j+1])$ es que, o bien $coste(M_1) > coste(\tilde{M}_1)$, o bien $coste(M_2) > coste(M_2)$. En cualquiera de los dos casos, se contradice la suposición inicial, con lo que se demuestra la implicación "⇒".

□

Intuitivamente, la consecuencia de este lema es que cualquier correspondencia M puede dividirse en subcorrespondencias que cubran distintas porciones de los dos árboles. Para ello, basta con elegir un nodo p en el árbol $T[1..i+1]$ y otro nodo q en $T'[1..j+1]$, que estén incluidos en el mismo par de M. En la primera subcorrespondencia, M_1, participarán los nodos desde las respectivas raíces de los dos árboles hasta esos dos nodos, y en el segundo los demás nodos hasta $i+1$ y $j+1$, respectivamente. El punto más interesante de este lema es que para que la correspondencia M sea de coste mínimo también deberán serlo las dos subcorrespondencias que la forman. Esto ofrece un procedimiento para construir correspondencias de coste mínimo a partir de subcorrespondencias, también de coste mínimo, que cubran porciones más pequeñas de los árboles iniciales. Además, el lema indica cómo combinar los costes de esas subcorrespondencias para calcular la distancia final.

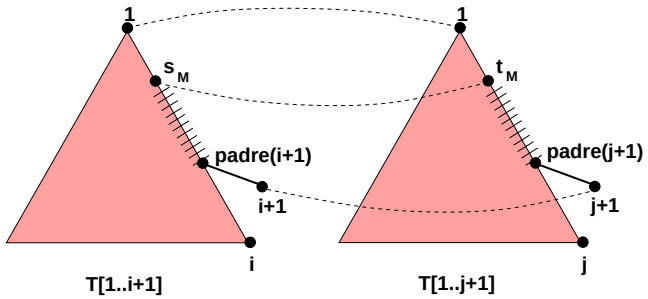


Figura 4.8: Nodos s_M y t_M.

Una vez que es posible simplificar la búsqueda de la correspondencia mínima, convirtiéndola en la búsqueda de sus dos subcorrespondencias, se aplicará esta idea para identificar la correspondencia cuyo coste es $min\_m(i+1, j+1)$. Se trata entonces de encontrar, entre los ancestros de $i+1$ y $j+1$, los nodos intermedios p y q, que separen los dos subcorrespondencias con menor coste. La selección de esos nodos intermedios p y q es la parte más compleja de la aproximación que propone Tai. Es necesario asegurar que el resultado de la unión de las dos subcorrespondencias, de acuerdo al lema 4.3, de lugar a una correspondencia correcta. En concreto se debe asegurar que la nueva correspondencia mantiene el orden jerárquico entre ancestros y descendientes para los nodos que intervienen en ella.

Se comenzará explicando cómo deben de ser esos nodos intermedios p y q, con respecto a los demás nodos que participan en la correspondencia y se razonará por qué siempre va a existir, al menos, un par de esos nodos p y q. Para ello nos apoyaremos en la figura 4.8. Siendo M una correspondencia entre $T[1..i+1]$ y $T'[1..j+1]$ que incluya al par $(i+1, j+1)$, siempre será posible encontrar dos nodos, s_M y t_M, que se correspondan con los ancestros más cercanos de $t[i+1]$ y $t[j+1]$ afectados por la correspondencia M. Es decir, ningún par de M afecta a un descendiente de $t[s_M]$ (resp. $t'[t_M]$) en el camino de $t[s_M]$ (resp. $t'[t_M]$) a $t[i+1]$ (resp. $t'[j+1]$), tal como se ilustra en la figura 4.8. En esta figura la línea rayada desde s_M (resp. t_M) a $padre(i+1)$ (resp. $padre(j+1)$) representa que ningún nodo en camino de $t[s_M]$ (resp. $t'[t_M]$) a $t[padre(i+1)]$ (resp. $t'[padre(j+1)]$) es afectado por un par de la correspondencia M, excepto el nodo $t[s_M]$ (resp. $t'[t_M]$). Además, y debido a las propiedades de la ordenación en preorden, $t[padre(i+1)]$ (resp. $t'[padre(j+1)]$) está en el camino de $t[s_M]$ (resp. $t'[t_M]$) a $t[i]$ (resp. $t'[j]$).

Dado que se ha supuesto que las raíces de T y T' son idénticas, el par $(1, 1)$ debe estar presente en todas las correspondencias. Por esto, siempre existirá, al menos, un par de dichos nodos, $t[s_M]$ y $t'[t_M]$. En el peor de los casos esos nodos serían las propias raíces de T y T'. Además, por las propiedades de las correspondencias, dado que $(i+1, j+1) \in M$, el par (s_M, t_M) también debe estar en M.

Una vez revisadas la características de los posibles nodos intermedios, se presenta el lema que justifica la selección de los nodos que dará lugar a la partición en las subcorrespondencias que se utilizarán en el cálculo de $min\_m(i+1, j+1)$.

Lema 4.4 (división de min_m en subcorrespondencias)**.**
Sean T y T' árboles y sean $t[s]$ y $t'[t]$ ancestros de los nodos $t[i+1]$ y $t'[j+1]$, respectivamente, entonces:

$$
\begin{aligned}
min\_m(i+1, j+1) = {} & \gamma(t[i+1] \to t'[j+1]) + \\
& min_{s,t}\{\ min\{\ coste(M_1) |\ M_1 \text{ es una correspondencia entre } T[1..s] \text{ y} \\
& \qquad T'[1..t], \text{ con } (s,t) \in M_1\} + \\
& \quad min\{\ coste(M_2) |\ M_2 \text{ es una correspondencia entre } T[s..i] \text{ y} \\
& \qquad T'[t..j],\ (s,t) \in M_2 \text{ y ningún par de } M_2 \text{ afecta a un} \\
& \qquad \text{descendiente de } t[s] \text{ (resp. } t'[t]) \text{ en el camino de } t[s] \\
& \qquad \text{(resp. } t'[t]) \text{ a } t[padre(i+1)] \text{ (resp. } t'[padre(j+1)])\} - \\
& \quad \gamma(t[s] \to t'[t])\ \}
\end{aligned}
$$

Demostración.
Para abreviar y simplificar la demostración se usará la abreviatura LD para referirse a la parte derecha de la igualdad anterior y LI para $min\_m(i+1, j+1)$. Primero se demostrará que LI $\geq$ LD y después que LI $\leq$ LD.

- Caso 1: LI $\geq$ LD

  Sea M cualquier posible correspondencia entre $T[1..i+1]$ y $T'[1..j+1]$ con $(i+1, j+1) \in M$, incluyendo la que da lugar a $min\_m(i+1, j+1)$.

  Tomamos los nodos $t[s_M]$ y $t'[t_M]$ y construimos M_1 y M_2. Donde M_1 es una correspondencia entre $T[1..s_M]$ y $T'[1..t_M]$, M_2 es una correspondencia entre $T[s_M..i]$ y $T'[t_M..j]$ y $M = M_1 \cup M_2 \cup \{(i+1, j+1)\}$. Dado que en LD se trabaja con mínimos se llega a:

  $$coste(M) = coste(M_1) + coste(M_2) + \gamma(t[i+1] \rightarrow t'[j+1]) \geq \text{LD}$$

  Puesto que la correspondencia que origina $min\_m(i+1, j+1)$ también se puede dividir de esta forma, se deduce que LI $\geq$ LD.

- Caso 2: LI $\leq$ LD

  Siendo $t[s]$ y $t'[t]$ dos ancestros cualesquiera de $t[i+1]$ y de $t'[j+1]$. Definimos M_1 como una correspondencia de $T[1..s]$ a $T'[1..t]$ que incluya al par (s, t). Y definimos M_2 como la correspondencia de $T[s..padre(i+1)]$ a $T'[t..padre(j+1)]$ en la que ningún par de M_2 afecta a los descendientes de $t[s]$ (resp. $t'[t]$) en el camino de $t[s]$ (resp. $t'[t]$) a $t[padre(i+1)]$ (resp. $t'[padre(j+1)]$).

  Construimos $M = M_2 \cup M_2 \cup \{(i+1, j+1)\}$, que es una correspondencia de $t[1..i+1]$ a $t'[1..j+1]$. Por la definición de $min\_m(i+1, j+1)$, tenemos que $min\_m(i+1, j+1) \leq coste(M)$, y por lo tanto LI $\leq$ LD.

De LI $\geq$ LD y de LI $\leq$ LD, se concluye que LI = LD, con lo que queda demostrado el lema. □

La consecuencia directa del lema anterior es que el cálculo de $min\_m(i+1, j+1)$ se reduce a buscar dos nodos intermedios s y t, que dividirán la correspondencia buscada en dos subcorrespondencias de coste mínimo. La primera irá desde las raíces de los dos árboles hasta $t[s]$ y $t'[t]$ y la segunda comprenderá desde $t[s]$ y $t'[t]$ hasta los padres de $t[i+1]$ y $t'[j+1]$. Con la condición añadida de que el par (s, t) esté presente en ambas subcorrespondencias y que no haya pares que afecten a los descendientes de $t[s]$ (resp. $t'[t]$) en el camino de $t[s]$ (resp. $t'[t]$) al padre de $t[i+1]$ (resp $t'[j+1]$). La necesidad de incluir esta última condición respecto a que no haya pares de la correspondencia entre s y t, y $i+1$ y $j+1$; es de asegurar que la correspondencia resultante sea válida. Concretamente, este requisito asegura que se mantendrá el orden ancestro-descendiente. Para abreviar la notación se define la distancia $E[s : u : i, t : v : j]$ de la siguiente forma.

Definición 4.9.
Sean T y T' árboles y sean $s \leq u \leq i$ y $t \leq v \leq j$, con $t[u]$ en el camino de $t[s]$ a $t[i]$ y con $t'[v]$ en el camino de $t'[t]$ a $t'[j]$, definimos:

$$E[s : u : i, t : v : j] = min\{\ coste(M) \mid \begin{array}{l} M \text{ es una correspondencia entre } T[s..i] \\ \text{y } T'[t..j] \text{ tal que } (s,t) \in M \text{ y ningún par de } M \\ \text{afecta a un descendiente de } t[s] \text{ (resp. } t'[t]) \\ \text{en el camino de } t[s] \text{ (resp. } t'[t]) \text{ a } t[u] \text{ (resp. } t'[v])\} \end{array}$$

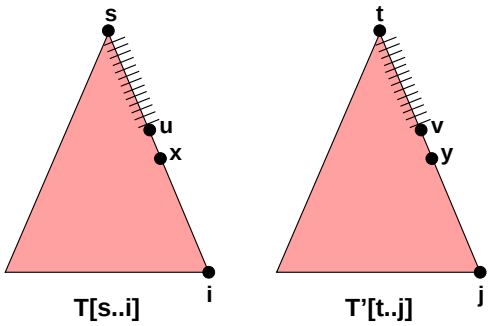


Figura 4.9: Cálculo de $E[s:u:i,t:v:j]$ con nodos intermedios.

Usando esta notación, llegamos al teorema 4.3, que deriva directamente del lema anterior.

Teorema 4.3 (cálculo de $min\_m(i+1,j+1)$)**.**
Sean T y T' árboles, y sean $t[s]$ y $t'[t]$ dos nodos ancestros de $t[i+1]$ y $t'[j+1]$, respectivamente. Entonces,

$$\begin{aligned} min\_m(i+1,j+1) &= \gamma(t[i+1] \rightarrow t'[j+1]) + \\ & \quad min_{s,t}\{\, min\_m(s,t) \;+\; E[s:u:i,t:v:j] \;-\; \gamma(t[s] \rightarrow t'[t])\} \end{aligned}$$

Demostración.
La demostración es directa a partir del lema 4.4 y de la definicion de $E[s:u:i,t:v:j]$. □

4.3.4. Cálculo de la distancia $E[s:u:i,t:v:j]$

El cálculo de la distancia $E[s:u:i,t:v:j]$ es la parte más compleja del algoritmo propuesto por Tai [43]. La mayor parte del coste, tanto en tiempo como en almacenamiento, se debe al cálculo de todos los valores $E[s:u:i,t:v:j]$ necesarios para aplicar el teorema 4.3. Será necesario calcular todos los $E[s:u:i,t:v:j]$ con $s \leq u \leq i,\ t \leq v \leq j$, y $t[u]$ (resp. $t'[v]$) en el camino desde $t[s]$ (resp. $t'[t]$) a $t[i]$ (resp. $t'[j]$). En los siguientes apartados se describe como realizar esos cálculos diferenciando dos casos en función de la topología de los dos subárboles considerados. En primer lugar se presentará el cálculo de $E[s:u:i,t:v:j]$ cuando existen nodos intermedios entre $t[u]$ y $t[i]$ y entre $t'[v]$ y $t'[j]$. Para continuar después, estudiando las distintas posibilidades que se presentan cuando no hay nodos intermedios en uno o en ambos subárboles.

Cálculo de $E[s:u:i,t:v:j]$ con nodos intermedios

En el primer caso tenemos que $s \leq u < i$ y $t \leq v < j$. Es decir, existe al menos un nodo $t[x]$, descendiente de $t[s]$, en el camino entre $t[s]$ y $t[i]$ y al menos un nodo $t'[y]$, descendiente de $t'[t]$, en el camino entre $t'[t]$ y $t'[j]$, tal como se ilustra en la figura 4.9. El siguiente lema ofrece el método para el cálculo de $E[s:u:i,t:v:j]$ de forma ascendente cuando nos encontramos en esta situación.

Lema 4.5.
Dados dos árboles T y T', sean $s \leq u < i$ y $t \leq v < j$, entonces :

$$E[s:u:i,t:v:j] = min \begin{cases} E[s:x:i,t:v:j], \\ E[s:u:i,t:y:j], \\ E[s:u:x-1,t:v:y-1] + E[x:x:i,y:y:j] \end{cases}$$

Siendo $t[x]$ (resp. $t'[y]$) el hijo de $t[u]$ (resp. $t'[v]$) en el camino de $t[u]$ (resp. $t'[v]$) a $t[i]$ (resp. $t'[j]$).

Demostración.
Se define M como la correspondencia de coste mínimo entre $T[s..i]$ a $T'[t..j]$, tal que $coste(M) = E[s : u : i, t : v : j]$, $(s,t) \in M$ y ningún par de M afecta a un nodo de T (resp. T') en el camino de $t[s]$ (resp. $t'[t]$) a $t[u]$ (resp. $t'[v]$). Se puede dar uno de estos tres casos:

- Caso 1: $t[x]$ no está afectado por M

  Entonces, el nodo $t[x]$ debe ser eliminado. Y dado que no participa en ningún par de M, tenemos que:

  $$E[s : u : i, t : v : j] = E[s : x : i, t : v : j]$$

- Caso 2: $t'[y]$ no está afectado por M

  Entonces, el nodo $t'[y]$ debe ser insertado. Y puesto que no participa en ningún par de M, tenemos que:

  $$E[s : u : i, t : v : j] = E[s : u : i, t : y : j]$$

- Caso 3: $t[x]$ está afectado por el par $(x,q) \in M$ y $t'[y]$ está afectado por $(p,y) \in M$

  Se demostrará que $t[x]$ y $t'[y]$ están afectados por el mismo par de M, es decir $x = p$, e $y = q$, y que el coste resultante se puede descomponer como se indica en el lema.

  Supongamos $p > x$, entonces:

  - Tenemos que $t[i]$ es un descendiente de $t[x]$ y $i \geq p \geq x$. Tal y como muestra el lema2.2, por las propiedades de la ordenación en preorden, tenemos que $t[p]$ será un descendiente de $t[x]$.
  - Por las propiedades de las correspondencias, si $p > x$ debe cumplirse $y > q$ y, por lo tanto, $t'[y]$ será un descendiente de $t'[q]$.

  Con esto llegamos a que $t'[q]$ es un descendiente de $t'[t]$ en el camino de $t'[t]$ a $t'[v]$, pero esto contradice la hipótesis de que no hay pares de M que afecten a los nodos presentes en ese camino, ya que se parte de que $t'[q]$ participa en la correspondencia.

  Para los casos $p < x$, $q > y$, y $q < y$ se llega a contradicciones análogas. Por lo tanto, tenemos que $p = x$, y $q = y$, es decir, $(x,y) \in M$.

  Ahora se demostrará que la correspondencia M puede descomponerse en M_1 y M_2, con $coste(M_1) = E[s : u : x-1, t : v : y-1]$ y $coste(M_2) = E[x : x : i, y : y : j]$.

  Definimos M_1 y M_2 de la siguiente forma:
  $M_1 = \{(m,n) | (m,n) \in M,\ m < x \text{ y } n < y\}$
  $M_2 = \{(m,n) | (m,n) \in M,\ x \geq m > i \text{ y } y \geq n > j\}$

  Tenemos entonces, que M_1 es una correspondencia de $T[s..x-1]$ a $T'[t..y-1]$ y M_2 es una correspondencia de $T[x..i]$ a $T'[y..j]$. Dado que $coste(M) = E[s : u : i, t : v : j]$ y que $t[u]$ ($t'[v]$) es el padre de $t[x]$ ($t'[y]$), tenemos que:

  $$coste(M_1) = E[s : u : x-1, v : y : y-1] \text{ y } coste(M_2) = E[x : x : i, y : y : j]$$

  Con lo que, aplicando la composición de correspondencias, queda probado que:

  $$E[s : u : i, t : v : j] = E[s : u : x-1, v : y : y-1] + E[x : x : i, y : y : j]$$

Dado que estos son los tres únicos casos posibles, bastará con tomar el mínimo para obtener la distancia $E[s : u : i, t : v : j]$ deseada. □

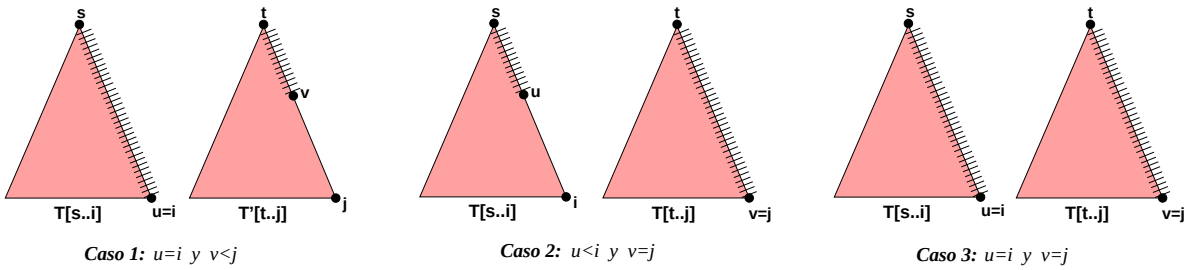


Figura 4.10: Cálculo de $E[s : u : i, t : v : j]$ sin nodos intermedios.

Cálculo de $E[s : u : i, t : v : j]$ en ausencia de nodos intermedios

Una vez obtenidas las expresiones a utilizar para calcular la distancia $E[s : u : i, t : v : j]$ cuando existen nodos intermedios, veremos ahora el segundo caso, cómo calcular $E[s : u : i, t : v : j]$ cuando $i = u$ o $j = v$. Es decir, estamos en alguna de las situaciones mostradas en la figura 4.10, donde o bien $t[u]$ y $t[i]$ son el mismo nodo, o bien $t'[v]$ y $t'[j]$ son el mismo nodo, o se dan ambas circunstancias a la vez. Se estudiarán las posibles situaciones que pueden darse en este caso, tomando como apoyo esa figura. Todos los casos y subcasos considerados se resumirán en las siguiente reglas:

1. Si $s = u = i$ y $t = v = j$: $E[s : u : i, t : v : j] = \gamma(t[i] \to t'[j])$
2. Si $s = u = i$ o $t < v = j$: $E[s : u : i, t : v : j] = E[s : u : i, t : padre(j) : j - 1] + \gamma(\varepsilon \to t'[j])$
3. Si $s < u = i$ o $t = u = j$: $E[s : u : i, t : v : j] = E[s : padre(i) : i - 1, t : v : j] + \gamma(t[i] \to \varepsilon)$

Comenzaremos el estudio definiendo M como una correspondencia de $T[s..i]$ a $T'[t..j]$, con $coste(M) = E[s : u : i, t : v : j]$, verificando la condición sobre los pares en el camino de $t[s]$ (resp. $t'[t]$) a $t[i]$ (resp. $t'[j]$). Pueden darse tres casos respecto de la distancia $E[s : u : i, t : v : j]$:

- Caso 1: $u = i$ y $v < j$

  Se da la situación mostrada en el esquema *(1)* de la figura 4.10, donde $t[u]$ y $t[i]$ son el mismo nodo y existen nodos entre $t'[v]$ y $t'[j]$. Se distinguen dos subcasos.

  (a) $s = u = i$

  En este caso $T[s..i]$ contiene un único nodo, $t[s]$. Además, ningún descendiente de $t'[t]$ está afectado por M. Como indica el lema 2.2, las propiedades de la ordenación en preorden aseguran que $t'[padre(j)]$ estará en el camino de $t'[t]$ a $t'[j - 1]$. Por lo tanto, el valor de $E[s : u : i, t : v : j]$ será la suma del coste de insertar $t'[j]$ y de $E[s : u : i, t : padre(j) : j - 1]$, que se corresponde con el coste mínimo de una correspondencia entre $T[s..u]$ con $T'[t..j - 1]$ que respete la condición usual sobre los caminos. El resultado es la siguiente expresión:

  $$E[s : u : i, t : v : j] = E[s : u : i, t : padre(j) : j - 1] + \gamma(\varepsilon \to t'[j])$$

  (b) $s < u = i$

  En este caso $T[s..i]$ contiene más de un nodo. Por las propiedades de la ordenación en preorden, $t[padre(i)]$ está en el camino de $t[s]$ a $t[i - 1]$. Y, dado que M no afecta a ninguno de los descendientes de $t[s]$ en el camino de $t[s]$ a $t[padre(i)]$, el valor de $E[s : u : i, t : v : j]$ se obtiene del coste de la inserción de $t[i]$ y del coste de la correspondencia entre $T[s..i - 1]$ y $T'[t..j]$, resultando la expresión siguiente:

  $$E[s : u : i, t : v : j] = E[s : padre(i) : i - 1, t : v : j] + \gamma(t[i] \to \varepsilon)$$

- Caso 2: $u < i$ y $v = j$

  Esta situación es simétrica a la descrita en el caso anterior, tal como se puede ver en la figura 4.10*(2)*. De nuevo, hay dos subcasos.

  (a) $t = v = j$
  Simétrico al subcaso *(a)* anterior, tenemos:

  $$E[s : u : i, t : v : j] = E[s : padre(i) : i - 1, t : v : j] + \gamma(t[i] \rightarrow \varepsilon)$$

  (b) $t < v = j$
  Simétrico al subcaso *(b)* anterior, tenemos:

  $$E[s : u : i, t : v : j] = E[s : u : i, t : padre(j) : j - 1] + \gamma(\varepsilon \rightarrow t'[j])$$

- Caso 3: ninguna de las combinaciones anteriores

  Se da la situación mostrada en la figura 4.10*(3)*, donde $t[u]$ y $t[i]$ son el mismo nodo y $t'[v]$ y $t'[j]$ son también el mismo nodo. Se diferencian cuatro subcasos:

  (a) $s = u = i$ y $t = v = j$
  Sólo hay un nodo en los árboles $T[s..i]$ y $T'[t..j]$. En este caso el valor de $E[s : u : i, t : v : j]$ será el coste del cambio de etiquetas entre $t[i]$ y $t'[j]$.

  $$E[s : u : i, t : v : j] = \gamma(t[i] \rightarrow t'[j])$$

  (b) $s = u = i$ y $t < v = j$
  El árbol $T[s..i]$ contiene un único nodo, $t[s]$, la situación es análoga al apartado*(a)* del primer caso, resultando:

  $$E[s : u : i, t : v : j] = E[s : u : i, t : padre(j) : j - 1] + \gamma(\varepsilon \rightarrow t'[j])$$

  (c) $s < u = i$ y $t = v = j$
  En este caso es $T'[t..j]$ quien contiene un único nodo, $t[s]$, la situación es análoga al apartado *(a)* del segundo caso. Tenemos entonces que:

  $$E[s : u : i, t : v : j] = E[s : padre(i) : i - 1, t : v : j] + \gamma(t[i] \rightarrow \varepsilon)$$

  (d) $s < u = i$ y $t < v = j$
  En esta situación hay más de un nodo tanto en $T[s..i]$ como en $T'[t..j]$. Además, ni $t[i]$, ni $t'[j]$ son afectados por la correspondencia M. Por lo tanto el valor $E[s : u : i, t : v : j]$ puede ser fruto de cualquiera de estas tres posibilidades: inserción de $t'[j]$, borrado de $t[i]$ o ambas operaciones. En todos los casos el valor será el mismo. En resumen, se obtiene la siguiente relación:

  $$\begin{aligned} E[s : u : i, t : v : j] &= E[s : padre(i) : i, t : v : j] + \gamma(t[i] \rightarrow \varepsilon) = \\ &= E[s : u : i, t : padre(j) : j - 1] + \gamma(\varepsilon \rightarrow t'[j]) = \\ &= E[s : padre(i) : i - 1, t : padre(j), j - 1] + \gamma(t[i] \rightarrow \varepsilon) + \gamma(\varepsilon \rightarrow t'[j]) \end{aligned}$$

4.3.5. Algoritmo para el cálculo de la distancia de edición

Una vez revisados los fundamentos teóricos, se presenta un algoritmo que aplica los lemas y teoremas anteriores para calcular la distancia de edición entre dos árboles etiquetados y ordenados, T y T', en un tiempo polinomial. El algoritmo necesita tres estructuras de almacenamiento permanentes, para mantener $E[s : u : i, t : v : j]$, $\text{MIN\_M}(i, j)$ y $D(i, j)$, y se divide en los tres pasos que se detallan a continuación.

Paso 1: Cálculo de $E[s:u:i,t:v:j]$, $\forall s,u,t,v,i,j$, con :

- $1 \leq i \leq |T|$,$1 \leq j \leq |T'|$.
- $t[u]$ (resp. $t'[v]$) está en el camino de $t[1]$ (resp.$t'[1]$) a $t[i]$ ($t'[j]$).
- $t[s]$ (resp. $t'[t]$) está en el camino de $t[1]$ (resp. $t'[1]$) a $t[u]$ (resp. $t'[v]$).

En este paso, mostrado en el algoritmo 4.2, se calculan todas las posibles distancias $E[s:u:i,t:v:j]$ aplicando el lema 4.5 y las reglas de la sección 4.3.4. Como resultado, se habrá almacenado el menor coste de las correspondencias entre todos los posibles subárboles de T y T' que cumplen la condición usual sobre los pares de la correspondencia en el camino de $t[s]$ (resp. $t'[t]$) a $t[u]$ (resp. $t'[v]$).
El orden de los bucles se establece de forma que los valores de $E[s:u:i,t:v:j]$ necesarios en una pasada hayan sido calculados en pasadas anteriores:

- Se recorre el árbol en preorden.
- Para cada par de nodos $i \in T$ y $j \in T'$, se van calculando todas las posibles distancias $E[s:u:i,t:v:j]$. En concreto, se recorre de forma ascendente el camino que comienza en $t[i]$ y acaba en la raíz de T. Haciendo lo mismo con el camino de $t'[j]$ a $t'[1]$.

Paso 2: Cálculo de $min\_m(i,j)$, $\forall i,j$, con $1 \leq i \leq |T|$,$1 \leq j \leq |T'|$.
Este paso se muestra en el algoritmo 4.2, que aplica el teorema 4.3. Se recorren los ancestros de i y j, estudiando todas las posibles formas de dividir en subcorrespondencias la correspondencia que produce $min\_m(i,j)$. Se utilizan los costes de correspondencias entre subárboles calculados en el paso *1.*

- En la primera parte del algoritmo se inicializan los valores de $min\_m(1,1)$, $min\_m(1,j)$, $\forall j$ y $min\_m(i,1)$, $\forall i$, de acuerdo al corolario 4.2.1, pero aplicado a $min\_m(i,j)$.
- Para cada par de nodos i, j, se buscan los nodos s y t que minimizan $min\_m(i,j)$.
- Para el cálculo de $min\_m(i,j)$ se necesitan los valores de min_m de todos sus ancestros.
  - Se recorren de forma ascendente hasta las raíces todos los ancestros de i y de j, buscando los mejores s y t.
  - Los valores de i y j recorren los árboles en preorden y en cada pasada se calcula un valor de min_m que se usará en la siguiente.

Paso 3: Cálculo de $D(i,j)$, $\forall i,j$, con $1 \leq i \leq |T|$,$1 \leq j \leq |T'|$.
Mostrado en el algoritmo 4.2, utiliza los resultados del teorema 4.2 y del corolario 4.2.1. Para cada $[1..i]$ y $T'[1..j]$ se calcula el coste de insertar $t[i]$, el de eliminar $t'[j]$ y el de la correspondencia directo entre los subárboles, seleccionando el menor de esos tres costes :

- En la primera parte del algoritmo se inicializan los valores de $D(1,1)$, $D(1,j)$, $\forall j$ y $D(i,1)$, $\forall i$; según el corolario 4.2.1.
- Para cada par de nodos i y j, se aplica el teorema 4.2 para calcular $D(i,j)$ utilizando :
  - $D(i-1,j)$ y $D(i,j-1)$, obtenidos durante pasadas anteriores de este algoritmo.
  - $min\_m(i,j)$, calculado en el paso 2.

Algoritmo 4.2 Paso 1 del algoritmo de Tai (cálculo de $E[s:u:i,t:v:j]$)

NOTA:

- Se define $padre^n(x) = padre^{n-1}(x)$ para $n \geq 1$ y $x > 1$, siendo $padre(x)$ el padre del nodo x y $padre^0(x) = x$
- $t[x]$ (resp. $t'[y]$) es el hijo de $t[u]$ resp. ($t'[v]$) en el camino de $t[u]$ (resp. $t'[v]$) hasta $t[i]$ (resp. $t'[j]$)

Entrada: T, T': árboles a comparar
Salida: valores $E[s:u:i,t:v:j]$, $\forall s,u,i,t,v,j$

for $i = 1,2,\ldots,|T|$ **do**
  for $j = 1,2,\ldots,|T'|$ **do**
    for $u = i, padre(i), padre^2(i), \ldots, 1$ **do**
      for $s = u, padre(u), padre^2(u), \ldots, 1$ **do**
        for $v = j, padre(j), padre^2(j), \ldots, 1$
          for $t = v, padre(v), padre^2(v), \ldots, 1$
            if $(s = u = i)$ **and** $(t = v = j)$
              $E[s:u:i,t:v:j] = \gamma(t[i] \rightarrow t'[j])$
            else if $(s = u = i)$ **or** $(t < v = j)$
              $E[s:u:i,t:v:j] = E[s:u:i,t:padre(j):j-1] + \gamma(\varepsilon \rightarrow t'[j])$
            else if $(s < u = i)$ **or** $(t = v = j)$
              $E[s:u:i,t:v:j] = E[s:padre(i):i-1,t:v:j] + \gamma(t[i] \rightarrow \varepsilon)$
            else

$$E[s:u:i,t:v:j] = \min \begin{cases} E[s:x:i,t:v:j], \\ E[s:u:i,t:y:j], \\ E[s:u:x-1,t:v:y-1] + E[x:x:i,y:y:j] \end{cases}$$

            end if
          end for
        end for
      end for
    end for
  end for
end for

4.3.6. Complejidad

En el siguiente teorema se muestra la complejidad temporal del algoritmo anterior. A la vista del pseudocódigo mostrado en los algoritmos 4.2, 4.3 y 4.4, es claro que el mayor coste se debe al cálculo de todos los $E[s:u:i,t:v:j]$. Y será ese cálculo, realizado en el paso 1 del algoritmo, el que determine la complejidad temporal de la totalidad del algoritmo.

Teorema 4.4.
Dados dos árboles, T y T', el algoritmo de Tai calcula la distancia de edición entre T y T' en un tiempo $O(|T| \times |T'| \times prof(T)^2 \times prof(T')^2)$, siendo $|T|$ y $|T'|$ el número de nodos de T y de T', y $prof(T)$ y $prof(T')$ la máxima profundidad de cada árbol.

Demostración.
Para comprobar que esta complejidad es correcta se revisarán los tres pasos del algoritmo:

- En el paso 1 se calcula $E[s:u:i,t:v:j]\ \forall s,u,t,v,i,j$, donde :

  $1 \leq i \leq |T|,\ 1 \leq j \leq |T'|$
  $t[u]$ (resp. $t'[v]$) está en el camino de $t[1]$ (resp. $t'[1]$) a $t[i]$ (resp. $t'[j]$)
  $t[s]$ (resp. $t'[t]$) está en el camino de $t[1]$ (resp. $t'[1]$) a $t[u]$ (resp. $t'[v]$)

  Por lo tanto, el paso 1 consume un tiempo $O(|T| \times |T'| \times prof(T)^2 \times prof(T')^2)$.

- El paso 2 calcula $min\_m(i,j)\ \forall i,j$, con $1 \leq i \leq |T|, 1 \leq j \leq |T'|$.

  Para cada par i, j, se recorren todos sus ancestros hasta llegar a las raíces. Así, se llega a que el paso 2 necesita un tiempo $O(|T| \times |T'| \times prof(T) \times prof(T'))$

- Por último, el paso 3 calcula $D(i,j)\ \forall i,j$, con $1 \leq i \leq |T|, 1 \leq j \leq |T'|$, consumiendo un tiempo $O(|T| \times |T'|)$.

De donde se llega a que $D(|T|,|T'|)$ puede calcularse en un tiempo $O(|T| \times |T'| \times prof(T)^2 \times prof(T')^2)$ siguiendo el algoritmo propuesto. □

Algoritmo 4.3 Paso 2 del algoritmo de Tai (cálculo de $min\_m(i,j)$)

Entrada: T, T', valores $E[s:u:i,t:v:j$ del paso 1
Salida: valores $min\_m(i,j)$ para $1 \leq i \leq |T|$, $1 \leq j \leq |T'|$

$min\_m(1,1) = 0$
$min\_m(i,1) = min\_m(i-1,1) + \gamma(t[i] \rightarrow \varepsilon),\ \forall i, 2 \leq i \leq |T|$
$min\_m(1,j) = min\_m(1,j-1) + \gamma(\varepsilon \rightarrow t'[j]),\ \forall j, 2 \leq j \leq |T'|$
for $i = 2,3,\ldots,|T|$ **do**
  for $j = 2,3,\ldots,|T'|$ **do**
    $min\_m(i,j) = \infty$
    for $s = padre(i), padre^2(i),\ldots,1$ **do**
      for $t = padre(j), padre^2(j),\ldots,1$ **do**
        $temp = min\_m(s,t) + E[s:padre(i):i-1,t:padre(j):j-1] - \gamma(t[s] \rightarrow t'[t])$
        $min\_m(i,j) = min\{temp, min\_m(i,j)\}$
      end for
    end for
    $min\_m(i,j) = min\_m(i,j) + \gamma(t[i] \rightarrow t'[j])$
  end for
end for

Algoritmo 4.4 Paso 3 del algoritmo de Tai (cálculo de $D(i,j)$)

Entrada: T, T', valores min_m del paso 2
Salida: valores $D(i,j)$ para $1 \leq i \leq |T|$, $1 \leq j \leq |T'|$

$D(1,1) = 0$
$D(i,1) = D(i-1,1) + \gamma(t[i] \rightarrow \varepsilon),\ \forall i, 2 \leq i \leq |T|$
$D(1,j) = D(1,j-1) + \gamma(\varepsilon \rightarrow t'[j]),\ \forall j, 2 \leq j \leq |T'|$
for $i = 2, 3, \ldots, |T|$ **do**
  for $j = 2, 3, \ldots, |T'|$ **do**

$$D(i,j) = min \begin{cases} D(i-1,j) + \gamma(t[i] \rightarrow \varepsilon), \\ D(i,j-1) + \gamma(\varepsilon \rightarrow t'[j]), \\ min\_m(i,j) \end{cases}$$

  end for
end for

Capítulo 5

Reconocimiento de patrones en arboles. Algoritmo de Zhang y Shasha

En este capítulo se presenta el algoritmo para la comparación de árboles propuesto por Shasha y Zhang [65]. Como veremos, este método tiene una complejidad menor que la del propuesto por Tai [43] y sus requisitos de espacio son mucho menores, además de resultar más simple e intuitivo. Estos autores siguen manteniendo la mayoría de los conceptos y definiciones introducidos por Tai, con tan sólo una ligera modificación de la definición de correspondencia. La principal diferencia es que el nuevo algoritmo procesa los árboles ordenados en postorden, lo que dará lugar a un procedimiento de cálculo de distancias ascendente, en oposición al algoritmo de Tai que era descendente.

En cuanto a la estrategia en sí, además de la forma de recorrer el árbol, la característica más destacada es que trabaja con bosques ordenados. Esos bosques se van formando a medida que se avanza en el algoritmo y son el resultado de ir eliminando raíces y nodos en los distintos subárboles que se pueden crear en los árboles originales. A grandes rasgos, lo que el algoritmo hace es ir construir bosques ordenados en los dos árboles e ir comparándolos entre sí, comenzando en las hojas y terminando en las raíces. Como se verá en este capítulo, dependiendo de si el bosque ordenado contiene un único árbol o contiene más de uno, el cálculo de la distancia se hará de forma distinta, consiguiendo con esa distinción reducir la cantidad de cálculos necesaria. El resultado es un algoritmo que calcula la distancia entre dos árboles en un tiempo $O(|T| \times |T'| \times min\{prof(T),\ hojas(T)\} \times min\{prof(T'),\ hojas(T')\})$, siendo $prof$ la profundidad del árbol y $hojas$ el número de hojas.

Comenzaremos presentando la notación utilizada en por Zhang y Shasha [65] y las modificaciones introducidas en los conceptos definidos por Tai [43]. Después presentaremos los lemas y teoremas que conforman la base teórica sobre la que se realiza el cálculo de distancias de edición. Describiremos, a continuación, un algoritmo que aglutina todos esos resultados para obtener la distancia de edición entre dos árboles. Una vez estudiada la complejidad temporal y espacial del algoritmo propuesto, finalizaremos el capítulo mostrando un ejemplo de su funcionamiento.

5.1. Definiciones y notación

Como hemos adelantado, Zhang y Shasha [65] mantienen las definiciones usadas en los trabajos de Tai [43]. La principal diferencia surge del hecho de que en este algoritmo se trabaja con árboles ordenados en postorden. Así, se modifica ligeramente la definición de correspondencia dada por Tai haciéndola independiente del tipo de ordenación de los árboles.

Definición 5.1 (correspondencia entre árboles)**.**
Se define una correspondencia *entre los árboles T y T', como una 3-upla (M, T, T'), donde M es un*

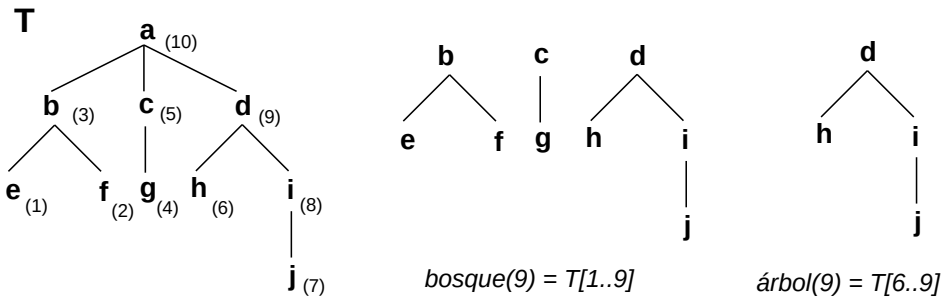


Figura 5.1: Numeración en postorden. Arboles y bosques.

conjunto de pares (i, j)*, con* $1 \leq i \leq |T|$ *y* $1 \leq j \leq |T'|$*, que verifica las siguientes propiedades para cualesquiera pares* $(i_1, j_1), (i_2, j_2) \in M$*:*

1. $i_1 = i_2$ *si y sólo si* $j_1 = j_2$*.*
2. $t[i_1]$ *está a la izquierda de* $t[i_2]$ *si y sólo si* $t'[j_1]$ *está a la izquierda de* $t'[j_2]$*.*
3. $t[i_1]$ *es un ancestro de* $t[i_2]$ *si y sólo si* $t'[j_1]$ *es un ancestro de* $t'[j_2]$*.*

Como en Tai [43], la propiedad *1* asegura que se manejan asignaciones uno a uno entre los nodos afectados por la correspondencia. La propiedad *2* asegura la conservación del orden entre hermanos y la propiedad *3* garantiza la conservación del orden jerárquico entre ancestros y descendientes. Esta nueva definición tiene la ventaja de que hace posible extender el concepto de correspondencia entre árboles para permitir *correspondencias entre bosques*. Las propiedades que deben cumplir los pares son las mismas, simplemente basta con tener en cuenta que los nodos proceden de bosques y no de árboles. Formalmente, tenemos la siguiente definición.

Definición 5.2 (correspondencia entre bosques)**.**
Se define una correspondencia entre los bosques B y B'*, como una 3-upla* (M, B, B')*, donde* M *es un conjunto de pares* (i, j)*, con* $1 \leq i \leq |B|$ *y* $1 \leq j \leq |B'|$*, que verifica las mismas propiedades que se exigen para las correspondencias entre árboles.*

Por lo que respecta a los demás conceptos, se sigue manteniendo su significado. La definición de las operaciones de edición es la misma y también la de la función de coste γ y la del coste de una correspondencia. Tampoco cambia la definición de la distancia entre árboles, aunque como se verá, se introduce en concepto de distancia entre bosques ordenados. Por último, en el trabajo de Zhang y Shasha [65] no se utilizan las notaciones $min\_m(i, j)$, ni $E[s : u : i, t : v : j]$ introducidas por Tai [43].

Pasaremos a presentar la notación empleada por Zhang y Shasha. Como viene siendo habitual, $t[i]$ denotará al i-ésimo nodo del árbol T según la ordenación en postorden. Las nuevas notaciones introducidas por Zhang y Shasha se enumeran a continuación:

- Se usará el símbolo $\emptyset$ para referirnos a un árbol, o a un bosque, sin ningún nodo. Esto es, a un árbol o un bosque vacío.
- La notación $l(i)$ se utiliza para representar la posición de la hoja más a la izquierda de las que descienden del nodo i-ésimo. Si $t[i]$ es una hoja, entonces $l(i) = i$.
- Como en los casos anteriores, $T[i..j]$ se refiere al bosque ordenado formado por los nodos de T entre i y j, ambos incluidos. Si $i > j$, entonces $T[i..j] = \emptyset$. Cuando sea necesario diferenciar entre árboles y bosques se empleará la siguiente distinción:

- Se usará $bosque(i)$ para referirse a $T[1..i]$.
- Se usará $arbol(i)$ en el caso $T[l(i)..i]$.

Como se puede ver en la figura 5.1, $arbol(i)$ se refiere al caso de un bosque con un sólo árbol, cuya raíz es $t[i]$. Del mismo modo, $bosque(i)$ es el bosque que contiene a ese árbol, $T[i]$, junto con un conjunto de uno o más árboles, que conforman su contexto izquierdo. Como se verá al describir el algoritmo que proponen estos autores, esta distinción entre $bosque(i)$ y $arbol(i)$ es importante, puesto que permitirá simplificar el cálculo de las distancias.

- Se usará $distBosque(T[i'..i], T'[j'..j])$ para denotar la distancia mínima entre los bosques $T[i'..i]$ y $T'[j'..j]$. Esta distancia no es más que la extensión del concepto de distancia de edición entre árboles a distancia de edición entre bosques. Formalmente tenemos:

$$distBosque(T[i'..i], T'[j'..j]) = min\{\gamma S| \quad \begin{array}{l} S \text{ es una secuencia de operaciones de edición} \\ \text{que transforma } T[i'..i] \text{ en } T'[j'..j]\} \end{array}$$

Que también se puede definir en base al coste de la correspondencia mínima:

$$distBosque(T[i'..i], T'[j'..j]) = min\{coste(M)| \quad \begin{array}{l} M \text{ es una correspondencia entre} \\ \text{los bosques } T[i'..i] \text{ y } T'[j'..j]\} \end{array}$$

Para abreviar la notación se empleará $distBosque(i'..i, j'..j)$ cuando no haya ambigüedad posible, sobreentendiendo que se refiere a los árboles T y T'.

- Análogamente, $distArbol(i, j)$ será la distancia mínima entre el subárbol con raíz en $t[i]$ y el subárbol con raíz en $t'[j]$. Es decir:

$$distArbol(i, j) = distBosque(l(i)..i, l(j)..j)$$

5.2. Cálculo de la distancia de edición

En este apartado se exponen los resultados teóricos que dan lugar al algoritmo propuesto por Zhang y Shasha [65]. Se presentarán los tres lemas que son la base del algoritmo y se explicará su significado de forma intuitiva. Finalmente se describe un algoritmo que combina estos resultados teóricos para efectuar el cálculo de la distancia de edición entre dos árboles, T y T'.

5.2.1. Distancias entre bosques y árboles

Comenzaremos presentando un lema, análogo al lema 4.2.1 de Tai [43], que permitir definir los casos triviales en el cálculo de distancias. Esto es, el cálculo de distancias en las que intervienen bosques vacíos.

Lema 5.1 (inicialización.)**.**
Sean T y T' árboles y sean $i \in T$ y $j \in T'$, entonces:

1. $distBosque(\emptyset, \emptyset) = 0$.
2. $distBosque(l(i)..s, \emptyset) = distBosque(l(i)..s-1, \emptyset) + \gamma(t[s] \rightarrow \varepsilon)$.
3. $distBosque(\emptyset, l(j)..t) = distBosque(\emptyset, l(j)..t-1) + \gamma(\varepsilon \rightarrow t'[t])$.

donde $s \in desc(i)$ y $t \in desc(j)$.

Demostración.
La demostración se realiza a partir de la definición de $distBosque(i'..i, j'..j)$:

- En el caso *1* no es necesaria ninguna operación de edición para transformar un bosque vacío, por lo que el coste es 0.
- En el caso *2*, la distancia de un bosque cualquiera al bosque vacío es igual a la suma del coste de las operaciones de eliminación de todos y cada uno de sus nodos.
- El caso *3* es simétrico. El coste de pasar del bosque vacío a un bosque $T'[l(j)..t]$ es el coste de insertar todos y cada uno de los nodos de ese bosque.

□

La interpretación intuitiva es bastante directa. La distancia entre dos bosques vacíos es 0. Para obtener un bosque vacío hay que eliminar todos los nodos de un bosque. Y, del mismo modo, para obtener un bosque determinado a partir de uno vacío se deben insertar todos los nodos necesarios. La utilidad de este lema es que determina como realizar la inicialización de las distancias que serán utilizadas en los sucesivos pasos del algoritmo.

El siguiente lema proporciona el método general para calcular las distancias entre bosques ordenados en postorden y determinará la estructura general del algoritmo de cálculo de distancias. Como sucedía con el lema anterior, este es análogo al teorema 4.2 de Tai, con la salvedad de que ahora se aplica sobre bosques ordenados en postorden.

Lema 5.2.
Sean T y T' árboles, sean $t[i]$ y $t'[j]$ dos de sus nodos y sean $s \in desc(i)$ y $t \in desc(j)$, entonces :

$$distBosque(l(i)..s, l(j)..t) \;=\; min \begin{cases} distBosque(l(i)..s-1, l(j)..t) \;+\; \gamma(t[s] \to \varepsilon), \\ distBosque(l(i)..s, l(j)..t-1) \;+\; \gamma(\varepsilon \to t'[t]), \\ distBosque(l(i)..l(s)-1, l(j)..l(t)-1) + \\ \qquad distBosque(l(i)..s-1, l(j)..t-1) + \\ \qquad \gamma(t[s] \to t'[t]) \end{cases}$$

Demostración.
Supondremos que M es una correspondencia de coste mínimo entre el bosque $T[(i)..s]$ y el bosque $T[l(i)..t]$, cuyo coste es $distBosque(l(i)..s, l(j)..t)$. Tenemos las siguientes posibilidades:

- Caso 1: $t[s]$ no está afectado por M

  En este caso el nodo $t[s]$ se debe eliminar. De esta forma, la distancia obtenida será igual al coste de ese borrado más la distancia entre el bosque restante, $T[l(i)..s-1]$, y $T'[l(j)..t]$, es decir:

  $$distBosque(l(i)..s, l(j)..t) \;=\; distBosque(l(i)..s-1, l(j)..t) \;+\; \gamma(t[s] \to \varepsilon)$$

- Caso 2: $t'[t]$ no está afectado por M

  En este caso el nodo $t'[t]$ se debe insertar. Este caso es simétrico al anterior. La distancia resultante será igual al coste de la inserción de ese nodo más la distancia entre el bosque $T[l(i)..s]$ y el bosque que resulta del borrado, $T'[l(j)..t-1]$, es decir:

  $$distBosque(l(i)..s, l(j)..t) \;=\; distBosque(l(i)..s, l(j)..t-1) \;+\; \gamma(\varepsilon \to t'[t])$$

- Caso 3: Tanto $t[s]$ como $t'[t]$ están afectados por M

  Este caso se ilustra en la figura 5.2. Supongamos que $(s,k) \in M$ y $(h,t) \in M$ son los pares que afectan a $t[s]$ y $t'[t]$. Se demostrará que $s = h$ y $k = t$, es decir que $t[s]$ y $t'[t]$ están afectados por el mismo par de M. Para ello, se usará la propiedad de conservación del orden de las correspondencias. Pueden darse dos casos respecto a la localización de los nodos $t[h]$ y $t[s]$:

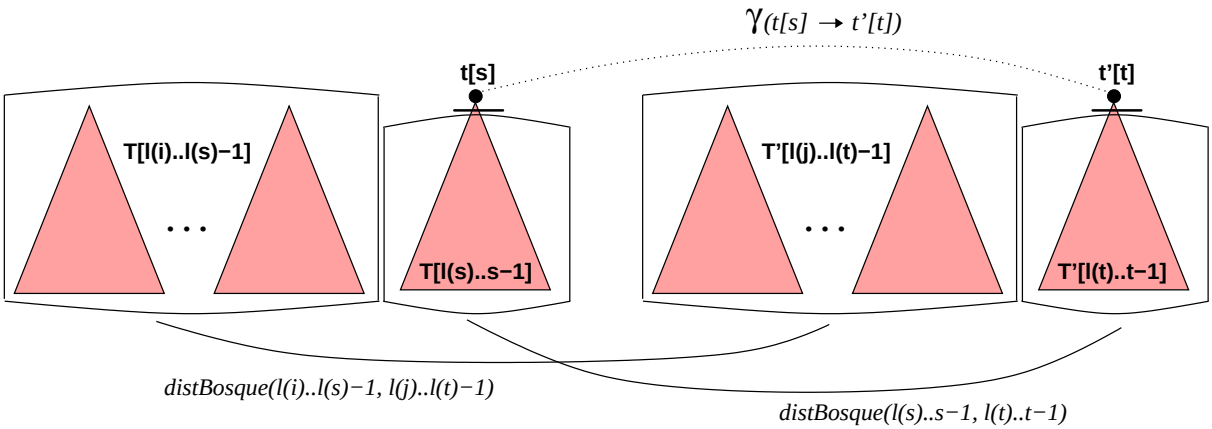


Figura 5.2: Caso 3 del lema 5.2.

(a) $\underline{l(i) \leq h \leq l(s) - 1}$
Esto es, $t[h]$ está en uno de los subárboles a la izquierda de $T[s]$. Para mantener el orden izquierda-derecha en M debería cumplirse $k > t$, lo cual es imposible en $T'[l(j)..t]$.

(b) $\underline{l(s) \leq h < s}$
Esto es, $t[h]$ está en el subárbol con raíz en $t[s]$. En este caso, dado que $t[s]$ es un ancestro de $t[h]$ y por la propiedad de conservación del orden ancestro-descendiente en M, $t'[k]$ debería ser un ancestro de $t'[t]$, con lo que se vuelve a que $k > t$, que es imposible en $T'[l(j)..t]$.

De estas dos contradicciones se llega a que $s = h$. Para el caso $k = t$ se puede realizar un razonamiento similar, resultando que $(s, t) \in M$.

Dado que hemos visto que $(s, t) \in M$, usando la condición sobre el orden ancestro-descendiente, se llega a que todo par de M que afecte a un nodo del subárbol con raíz en $t[s]$, esto es $T[l(s)..s]$, debe afectar a un nodo en el subárbol que parte de $t'[t]$, esto es $T'[l(t)..t]$. De esta forma, los pares de M se dividen en dos grupos, como muestra la figura 5.2. Los que afectan a nodos de $arbol(s)$ y $arbol(t)$, y los que afectan al contexto izquierdo de esos árboles, los bosques $T[l(i)..l(s) - 1]$ y $T'[l(j)..l(t) - 1]$, respectivamente. Con lo que se llega a que, para este caso, la distancia a calcular se puede obtener de la siguiente forma :

$$\begin{aligned} distBosque(l(i)..s, l(j)..t) \quad = \quad & distBosque(l(i)..l(s) - 1, l(j)..l(t) - 1) + \\ & distBosque(l(s)..s - 1, l(t)..t - 1) \; + \; \gamma(T[s] \rightarrow T'[t]) \end{aligned}$$

Es decir, como se aprecia en la figura 5.2, la distancia entre los dos bosques se obtiene a partir de la distancia entre los últimos subárboles del bosque, excluidos los nodos $t[s]$ y $t'[t]$, más la distancia entre los dos bosques que contienen a los nodos restantes, mas el coste de la operación $t[s] \rightarrow t'[t]$.

Todos las posibles correspondencias que originan la distancia $distBosque(l(i)..s, l(j)..t)$ serán de una de estas tres formas, y bastará con tomar el mínimo de esos tres valores, con lo que se demuestra el lema. □

El resultado de este lema permite calcular la distancia entre los bosques $T[l(i)..s]$ y $T'[l(j)..t]$ una vez que se conozcan los valores de las distancias $distBosque(l(i)..s - 1, l(t)..t)$, $distBosque(l(i)..s, l(t)..t - 1)$, $distBosque(l(i)..l(s) - 1, l(t)..l(t) - 1)$ y $distBosque(l(s)..s - 1, l(t)..t - 1)$ Estas se corresponden con distancias entre bosques de menor tamaño incluidos dentro de los dos bosques de partida, $T[l(i)..s]$ y $T'[l(j)..t]$. En concreto, se elige la menos costosa de estas tres opciones, cada una asociada a una de las tres posibles operaciones de edición:

- Transformar $T[l(i)..s - 1]$ en $T'[l(j)..t]$ y luego eliminar $t[s]$.

- Transformar $T[l(i)..s]$ en $T'[l(j)..t-1]$ y luego insertar $t'[t]$.
- Transformar $T[l(i)..l(s)-1]$ en $T'[l(j)..l(t)-1]$ y $T[l(s)..s-1$ en $T'[l(t)..t-1]$, para después cambiar la etiqueta de $t[s]$ por la de $t'[t]$.

El cálculo de distancias utilizando el lema anterior puede refinarse aún más, tal como se muestra en el siguiente lema. La idea es distinguir dos casos en el cálculo de distancias: distancias entre bosques y distancias entre árboles, obteniendo dos conjuntos de fórmulas distintos para cada caso.

Lema 5.3.
Sean T y T' árboles, sean $t[i]$ y $t'[j]$ dos de sus nodos y sean $s \in desc(i)$ y $t \in desc(j)$, entonces:

1. *Si $l(s) = l(i)$ y $l(t) = l(j)$: (Distancia entre árboles)*

$$distBosque(l(i)..s, l(j)..t) = min \begin{cases} distBosque(l(i)..s-1, l(j)..t) \ + \ \gamma(t[s] \to \varepsilon), \\ distBosque(l(i)..s, l(j)..t-1) \ + \ \gamma(\varepsilon \to t'[t]), \\ distBosque(l(i)..s-1, l(j)..t-1) \ + \ \gamma(t[s] \to t'[t]) \end{cases}$$

2. *Si $l(s) \neq l(i)$ y $l(t) \neq l(j)$: (Distancia entre bosques)*

$$distBosque(l(i)..s, l(j)..t) = min \begin{cases} distBosque(l(i)..s-1, l(j)..t) \ + \ \gamma(t[s] \to \varepsilon), \\ distBosque(l(i)..s, l(j)..t-1) \ + \ \gamma(\varepsilon \to t'[t]), \\ distBosque(l(i)..l(s)-1, l(j)..l(t)-1) \ + \ distArbol(s,t) \end{cases}$$

Demostración.
Las dos primeras expresiones de cada conjunto de fórmulas coinciden con las empleadas en el lema anterior y el razonamiento en estos casos es el mismo que el realizado en la demostración anterior. Pasaremos entonces a estudiar el tercer miembro de cada uno de los dos conjuntos de cálculos:

- Caso 1: $distBosque(l(i)..s-1, l(j)..t-1) \ + \ \gamma(t[s] \to t'[t])$

  La parte superior de la figura 5.3 muestra este caso. La demostración es directa a partir del lema anterior y de la definición de *distBosque*. Se trata de calcular la distancia entre dos bosques formados, cada uno de ellos, por un único árbol, dado que $l(s) = l(i)$ y $l(t) = l(j)$. Por lo tanto, resulta que los bosques $T[l(i)..l(s)-1]$ y $T'[l(j)..l(t)-1]$ están vacíos. Aplicando la tercera expresión del lema 5.2 y teniendo en cuenta que $distBosque(\emptyset, \emptyset) = 0$, llegamos a:

$$\begin{aligned} distBosque(l(i)..s, l(j)..t) &= distBosque(\emptyset, \emptyset) + distBosque(l(i)..s-1, l(j)..t-1) + \gamma(t[s] \to t'[t]) \\ &= distBosque(l(i)..s-1, l(j)..t-1) + \gamma(t[s] \to t'[t]) \end{aligned}$$

- Caso 2: $distBosque(l(i)..l(s)-1, l(j)..l(t)-1) \ + \ distArbol(s,t)$

  Este caso se corresponde con la situación mostrada en la parte inferior de la figura 5.3. Tomando como punto de partida el caso 3 del lema 5.2, se deberá demostrar:

$$distArbol(s,t) = distBosque(l(s)..s-1, l(t)..t-1) + \gamma(t[s] \to t'[t])$$

  Veremos como esto es así utilizando las definiciones de *distBosque* y *distArbol*:

  - La distancia $distBosque(l(i)..l(s)-1, l(j)..l(t)-1) + distArbol(s,t)$ se corresponde con el coste de una correspondencia entre $T[l(i)..s]$ y $T'[l(j)..t]$. Dado que $distBosque(l(i)..s, l(j)..t)$ es el coste de la correspondencia mínima entre esos bosques, tenemos que:

$$distBosque(l(i)..s, l(j)..t) \ \leq \ distBosque(l(i)..l(s)-1, l(j)..l(t)-1) + distArbol(s,t)$$

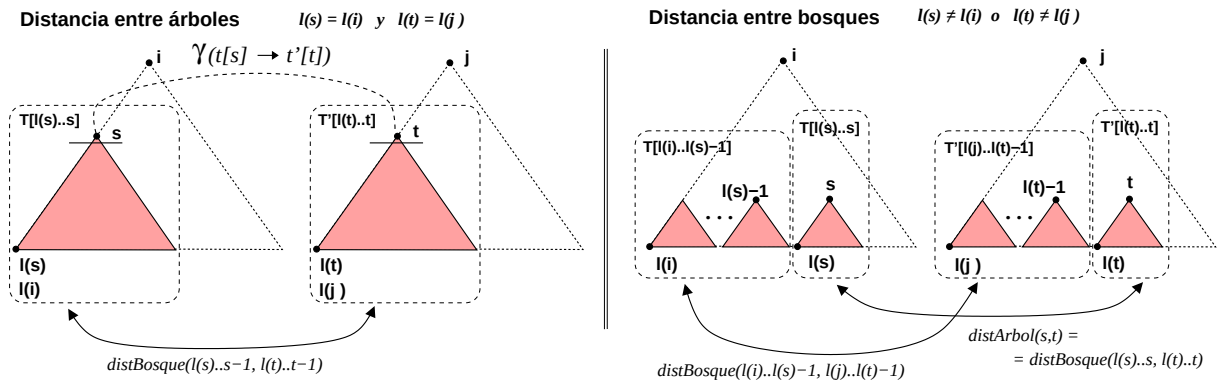


Figura 5.3: Distancias entre árboles y entre bosques.

- Para $distArbol(s,t)$ tenemos una situación análoga con respecto a la distancia $distBosque(l(s)..s-1, l(t)..t-1) + \gamma(t[s] \rightarrow t'[t])$, resultando que:

$$distArbol(s,t) \leq distBosque(l(s)..s-1, l(t)..t-1) + \gamma(t[s] \rightarrow t'[t])$$

Combinado ambas desigualdades,

$$\begin{array}{rcl} distBosque(l(i)..s, l(j)..t) & \leq & distBosque(l(i)..l(s)-1, l(j)..l(t)-1) + distArbol(s,t) \leq \\ & \leq & distBosque(l(i)..l(s)-1, l(j)..l(t)-1) + \\ & & distBosque(l(s)..s-1, l(t)..t-1) + \gamma(t[s] \rightarrow T'[t]) \end{array}$$

con el caso 3 del lema 5.2:

$$\begin{array}{rcl} distBosque(l(i)..s, l(j)..t) & = & distBosque(l(i)..l(s)-1, l(j)..l(t)-1) + \\ & & distBosque(l(s)..s-1, l(t)..t-1) + \gamma(t[s] \rightarrow T'[t]) \end{array}$$

llegamos a que, en este caso:

$$distArbol(s,t) = distBosque(l(s)..s-1, l(t)..t-1) + \gamma(t[s] \rightarrow t'[t])$$

Con lo que se demuestra que el caso *2* es correcto.

□

A partir de este lema se puede esbozar un método ascendente para el cálculo de las distancias entre bosques. A grandes rasgos y apoyándonos en la ordenación en postorden, se inspeccionarán todos los posibles subbosques comenzando por las hojas más a la izquierda de cada bosque. Esos subbosques se irán ampliando hasta llegar a los bosques iniciales. Para cada par de bosques generados de esa forma se aplicará, dependiendo de su topología de, uno de los dos conjuntos de fórmulas anteriores, utilizando distancias ya calculadas en pasos anteriores.

5.2.2. Algoritmo para el cálculo de las distancias

Veremos ahora como utilizar los resultados teóricos que acabamos de presentar para construir un algoritmo que efectúe el cálculo de la distancia de edición entre dos árboles T y T'. En esencia, se seguirá el método esbozado al final de las sección anterior. Sin embargo, será necesario tener en cuenta una serie de consideraciones que afectan a la forma de organizar el cálculo de las distancias y que harán más eficiente el mecanismo de cálculo. Comenzaremos revisando el lema 5.3 y sus consecuencias más importantes:

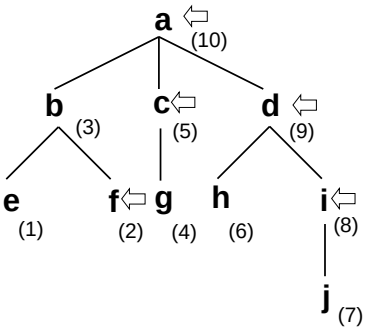


Figura 5.4: Nodos del conjunto $raíces\_I(T)$.

1. Las fórmulas propuestas en dicho lema permiten utilizar técnicas de programación dinámica para realizar el cálculo de las distancias necesarias. Tal como muestran ambas fórmulas, toda distancia entre bosques se puede obtener a partir de las distancias de subbosques más pequeños incluidos dentro de ellos.

   De esta forma, una vez calculadas las distancias entre un par de bosques, éstas podrán ser almacenadas para utilizarlas, más tarde, en el cálculo de distancias entre bosques más grandes. La forma natural de hacer esto es de forma ascendente, siguiendo la numeración en postorden de los nodos del árbol.

2. Tal y como se muestra en el caso *2* del lema 5.3, para el cálculo de $distArbol(i, j)$ serán necesarios la mayoría de los valores de $distArbol(s, t)$, siendo s un nodo del subárbol con raíz en i y t un nodo del subárbol con raíz en j. Esto lleva, de nuevo, a plantear un procedimiento ascendente, que vaya calculando las distancias entre subárboles partiendo de los nodos hoja hasta llegar a las raíces.

3. Como consecuencia del caso *1* del mismo lema, se observa que no es necesario calcular de forma separada el valor $distArbol(s, t)$ cuando el nodo s está en el camino de $l(i)$ a i y el nodo t está en el camino de $l(j)$ a j, es decir cuando $l(i) = l(s)$ y $l(j) = l(t)$. Las distancias entre los subárboles que cumplan esta condición se irán obteniendo como subproducto durante el cálculo de $distArbol(i, j)$.

En resumen, para calcular la distancia entre dos árboles T y T' será necesario calcular de forma iterativa, y siguiendo la ordenación en postorden, las distancias $distArbol(i, j)$ entre todos los posibles pares de subárboles $T[i]$ y $T'[j]$ en T y T'. Para obtener esas distancias entre subárboles, tal como indica el caso *2* del lema 5.3, es necesario calcular, también siguiendo la numeración en postorden, todas las distancias $distBosque(l(i)..s, l(j)..t,)$, para valores crecientes de s y t, con $l(i) \leq s \leq i$ y $l(j) \leq t \leq j$.

El aspecto crucial de la propuesta de Zhang y Shasha [65] es que realmente no será necesario calcular de forma separada las distancias $distArbol(i, j)$ para todos los posibles subárboles $T[i]$ y $T'[j]$ en T y T'. Algunas de esas distancias se pueden obtener durante el cálculo de distancias entre otros subárboles de mayor tamaño. De esta forma, se introduce el conjunto $raíces\_I(T)$[1] que representa a las raíces de los subárboles de T para las que es necesario calcular de forma separada sus distancias. Formalmente, tenemos la definición que sigue:

Definición 5.3 (conjunto $raíces\_I(T)$)**.**
Sea T un árbol, se define el conjunto $raíces\_I(T)$ de la siguiente forma:

$$raíces\_I(T) = \{k |\ 1 \leq k \leq |T|\ y\ \nexists k' > k\ tal\ que\ l(k) = l(k')\}$$

[1] En el artículo original se emplea el término anglosajón *LR_keyroot*.

Algoritmo 5.1 Algoritmo de Zhang y Shasha

Entrada: árboles T y T'
$raíces\_I(T)$ y $raíces\_I(T')$ ordenados de menor a mayor
Salida: distancias $distArbol(s,t)$ para todos los subárboles $t[s] \in T$ y $t'[t] \in T'$

```
/*recorrer conjunto de raíces_I*/

for i = 1 to |raíces_I(T)| do
  for j = 1 to |raíces_I(T')| do

    /* inicialización distBosque(l(i)..s, l(j)..s) (lema 5.1) */

    distBosque(∅, ∅) = 0
    distBosque(l(i)..s, ∅)) = distBosque(l(i)..s − 1, ∅) + γ(t[s] → ε), ∀s con l(i) ≤ s ≤ i
    distBosque(∅, l(j)..t) = distBosque(∅, l(j)..t − 1) + γ(ε → t'[t]), ∀t con l(j) ≤ t ≤ j

    for s = l(i) to i do
      for t = l(j) to j do
        if (l(s) = l(i)) and (l(t) = l(j)) then
          /* distancia árbol-árbol (caso 1 lema 5.3) */

          distBosque(l(i)..s, l(j)..t) = min { distBosque(l(i)..s − 1, l(j)..t)     + γ(t[s] → ε),
                                             { distBosque(l(i)..s, l(j)..t − 1)     + γ(ε → t'[t]),
                                             { distBosque(l(i)..s − 1, l(j)..t − 1) + γ(t[s] → t'[t])}
          distArbol(s,t) = distBosque(l(i)..s, l(j)..t) /* guardar valor intermedio */
        else
          /* distancia bosque-bosque (caso 2 lema 5.3) */

          distBosque(l(i)..s, l(j), t) = min { distBosque(l(i)..s − 1, l(j)..t)                + γ(t[s] → ε),
                                             { distBosque(l(i)..s], l(j)..t − 1)               + γ(ε → t'[t]),
                                             { distBosque(l(i)..l(s) − 1], l(j)..l(t) − 1)     + distArbol(s,t)}
        end if
      end for
    end for
  end for
end for
```

Intuitivamente, los nodos de $raíces\_I(T)$ no comparten con ningún otro nodo de mayor nivel su hoja más a la izquierda. Esto es, si el nodo k-ésimo de T está en $raíces\_I(T)$, entonces, o bien k es la raíz de T, o tiene, al menos, un hermano a su izquierda. Además, en este último caso se verifica que $l(k) \neq l(padre(k))$, puesto que existen hermanos izquierdos de k. La figura 5.4 muestra los nodos del conjunto $raíces\_I(T)$ para un árbol de ejemplo. Los nodos pertenecientes a ese conjunto están marcados con una flecha, y como hemos visto se corresponden con la raíz del árbol y con los nodos con hermanos por la izquierda.

Zhang y Shasha [65] demuestran que para obtener la distancia entre dos árboles T y T', sólo es necesario calcular las distancias $distArbol(i,j)$ cuando ambos nodos son $raíces\_I$, esto es, $i \in raíces\_I(T)$ y $j \in raíces\_I(T')$. Las distancias entre los demás subárboles se podrán obtener durante el cálculo de las distancias entre las $raíces\_I$ anteriores.

Suponiendo que los nodos de los conjuntos $raíces\_I(T)$ y $raíces\_I(T')$ están ordenados según su numeración en postorden. El algoritmo recorre esos conjuntos y, para cada par de nodos $i \in raíces\_I(T)$ y $j \in raíces\_I(T')$, se calcula la distancia $distArbol(i,j)$. Para este cálculo se construyen, siguiendo la numeración en postorden, los bosques $T[l(i)..s]$ y $T'[l(j)..t]$, con s tomando valores desde $l(i)$ hasta i y t

con valores desde $l(j)$ a j. Aplicando el lema 5.3 se calcula la distancia entre todos los pares de bosques así construidos y se guardan esos valores.

En cuanto al almacenamiento, se utiliza una matriz permanente de orden $|T| \times |T'|$ para mantener los valores de $distArbol(i, j)$. Se emplea también una matriz temporal para guardar los valores de $distBosque(l(i)..s, l(j)..t)$ necesarios en durante el cálculo de la distancia $distArbol(i, j)$ para $i \in raíces\_I(T)$ y $j \in raíces\_I(T')$. El orden de esta matriz es $(|T[i]| + 1) \times (|T'[j]| + 1)$ y se inicializa aplicando el lema 5.1. Cada vez que se calcule una distancia $distBosque(l(i)..s, l(j)..t)$ con $l(s) = l(i)$ y $l(t) = l(j)$ su valor se almacena en la matriz $distArbol$ puesto que se corresponde con una distancia entre subárboles[2]. Una vez que finalice el algoritmo, en la matriz $distArbol$ habrán sido almacenadas todas las distancias $distArbol(i, j)$ para todos los pares de nodos $i \in T$ y $j \in T'$. Finalmente, la distancia entre T y T estará almacenada en la última celda, la correspondiente a $distArbol(|T|, |T'|)$. El procedimiento que se ha descrito se muestra en al algoritmo 5.1

Una vez presentado el algoritmo que proponen Zhang y Shasha [65] y explicado su funcionamiento de forma intuitiva, terminaremos esta sección demostrando formalmente que el cálculo que realiza es correcto.

Teorema 5.1.
El algoritmo 5.1 calcula correctamente la distancia de edición entre dos arboles T y T'

Demostración.
Debemos demostrar que en todo momento los valores de las distancias necesarias para efectuar los cálculos del lema 5.3 ya hayan sido calculadas y estén disponibles cuando sean necesarias.
Comenzaremos con la inicialización de las matrices temporales. Como consecuencia del lema 5.1, la inicialización de la matriz de distancias $distBosque$ es correcta y se realiza antes iniciar el cálculo de todas las demás distancias entre subbosques.
Para las demás distancias deberemos demostrar que, para cualquier par de nodos i y j, tales que $i \in raíces\_I(T)$ y $j \in raíz\_I(T')$ se cumple lo siguiente:

1. Antes de iniciar el cálculo de la distancia $distArbol(i, j)$, todas las distancias $distArbol(s, t)$, para $l(i) \leq s < i$ y $l(j) \leq t < j$, con $l(s) \neq (i)$ o $l(t) \neq l(j)$, ya habrán sido calculadas. Esto es, están disponibles todas las distancias $distArbol(s, t)$, en las que $t[s]$ no está en el camino desde $t[l(i)]$ a $t[i]$, y $t'[t]$ tampoco está en el camino desde $t'[l(j)]$ a $t'[j]$.

2. Justo después de calcular $distArbol(i, j)$, todas las distancias $distArbol(s, t)$, con $l(i) \leq s \leq i$ y $l(j) \leq t \leq j$, estarán disponibles. Esto es, se habrán calculado las distancias $distArbol$ entre todos los pares de subárboles existentes en $T[l(i)..i]$ y $T'[l(j)..j]$.

Para probar que la condición *1* siempre se cumple, estudiaremos las dos situaciones siguientes:

- Caso *(a)*: $l(s) \neq l(i)$

  Supongamos que i' es el ancestro[3] más cercano de s tal que $l(s) = l(i')$ y $i' \in raíces\_I(T)$, esto es, ambos nodos tienen en común la hoja más a la izquierda e $i' \in raíces\_I(T)$. Que i' sea una $raíz\_I$ implica que la distancia $distArbol(s, t)$ se calculará durante el cálculo de $distArbol(i', j)$. Puesto que $l(i') = l(s) \neq l(i)$, tenemos que $i' \neq i$. Además, como i' es un ancestro de s más cercano que i, $i' \leq i$. Por lo que se llega a que $i' < i$.

- Caso *b*: $l(t) \neq l(j)$

  Usaremos un razonamiento análogo. Sea j' es el ancestro más cercano de t tal que $l(t) = l(j')$ y $j' \in raíces\_I(T')$. Como en el caso anterior, la distancia $distArbol(s, t)$ se calculará durante el cálculo de $distArbol(i, j')$. Puesto que $l(j') = l(t) \neq l(j)$, tenemos que $j' \neq j$. De nuevo, j' es un ancestro de t más cercano que j, entonces $j' \leq j$. Por lo que se llega a que $j' < j$.

[2] Esto incluye también a las distancias entre pares de $raíces\_I$.
[3] Ese ancestro puede ser el propio nodo s, puesto que suponemos que $anc^0(i) = i$.

Tenemos entonces que $l(s) = l(i')$ y $l(t) = l(j')$, y que $i' \in raíces\_I(T)$ y $j' \in raíces\_I(T')$. Esto significa que la distancia $distArbol(s,t)$ se podrá obtener[4] durante el cálculo de $distArbol(i',j')$. Dado que partimos de que los conjuntos $raíces\_I(T)$ y $raíces\_I(T')$ están ordenados de forma ascendente y hemos llegado a que $i' < i$ y $j' < j$, el cálculo de $distArbol(i',j')$ tendrá lugar antes que el $distArbol(i,j)$. De esta forma, para este último cálculo, el valor $distArbol(s,t)$ ya estará disponible en la matriz $distArbol$.

Veremos ahora como si la condición *1* se cumple, también se cumple la condición *2*. De acuerdo con lo que dice el lema 5.3, necesitamos todas las distancias $distArbol(s,t)$, para $l(i) \leq s < i$ y $l(j) \leq t < j$, con $l(s) \neq (i)$ o $l(t) \neq l(j)$. Puesto que partimos de que *1* es cierto, esas distancias ya estaban disponibles antes de iniciar el cálculo de $distArbol(i,j)$. Las distancias $distArbol(s,t)$ para los subárboles en los que $l(s) = l(j)$ y $l(t) = l(j)$, se irán obteniendo a medida que se aplique el primer conjunto de fórmulas del lema 5.3 y se almacenarán en la matriz permanente $distArbol$. Por lo tanto, todas las distancias exigidas en la condición *2* estarán disponibles una vez finalizado el cálculo de $distArbol(i,j)$.

Con todo esto se demuestra que empleando el algoritmo 5.1, todas las distancias necesarias para aplicar el lema 5.3 estarán disponibles en el momento que vayan a ser utilizadas. □

5.2.3. Complejidad

Una vez probada la corrección del algoritmo de Zhang y Shasha [65], comprobaremos cual es su complejidad. En esencia, lo que hace el algoritmo es calcular las distancias $distArbol(i,j)$ para cada par de nodos $i \in raíces\_I(T)$ y $j \in raíces\_I(T')$. De este modo, en una primera aproximación la complejidad temporal resultante sería:

$$O\left(\sum_{i \in raíces\_I(T)} \sum_{j \in raíces\_I(T')} |T[i]| \times |T'[j]|\right)$$

Con un análisis más detallado Zhang, y Shasha [65] llegan a una complejidad $O(|T| \times |T'| \times min\{prof(T), hojas(T)) \times min(prof(T'), hojas(T')\}$, siendo $prof(T)$ y $prof(T')$ la profundidad de cada árbol, y $hojas(T)$ y $hojas(T')$, el número de nodos hoja en cada árbol. Para demostrarlo necesitaremos una serie de resultados previos. En primer lugar, acotaremos el número de nodos $raíz\_I$ presentes en un árbol T.

Lema 5.4.
Dado un árbol T, $|raíces\_I(T)| \leq hojas(T)$, siendo $hojas(T)$ el número de nodos hojas del árbol T.

Demostración.
Probaremos en primer lugar que, para cualesquiera $i, j \in raíces\_I(T)$, $l(i) \neq l(j)$.
Sean $i, j \in raíces\_I(T)$, tales que $i < j$. Si fuera cierto que $l(i) = l(j)$, tendríamos que i estaría en el camino de $l(j)$ a i, dado que $i < j$. Por la definición de $l(j)$, i no podría tener hermanos izquierdos. Esto contradice la afirmación de partida, $i \in raíces\_I(T)$. Por lo tanto, $l(i) \neq l(j)$ para $i, j \in raíces\_I(T)$.

El anterior razonamiento significa que cada hoja del árbol T puede ser el descendiente más a la izquierda de un miembro de $raíces\_I(T)$, como máximo. Con lo que se verifica $|raíces\_I(T)| \leq hojas(T)$. □

Una vez acotado el número de $raíces\_I$, determinaremos el número de cálculos de distancias $distArbol$ en los que va a participar un nodo de T. Esto es, dado un nodo $s \in T$, queremos conocer de cuantos nodos $raíces\_I$ es descendiente. Conocido el número de $raíces\_I$ que incluyen a s, se sabrá el número de veces va a ser procesado ese nodo. Por esta razón, se introduce el concepto de $prof\_raíces\_I$.

Definición 5.4.
Sea T un árbol y sea i uno de sus nodos. Se define la $prof\_raíces\_I$ de i de la siguiente forma:

$$prof\_raíces\_I(i) = |anc(i) \cap raíces\_I(T)|$$

[4] Empleando el primer conjunto de fórmulas del lema 5.3.

Se extenderá este concepto para definir la prof_raíces_I de un árbol T de la forma:

$$prof\_raíces\_I(T) = \max_{1 \leq i \leq |T|} \{prof\_raíces\_I(i)\}$$

Intuitivamente $prof\_raíces\_I(i)$ se corresponde con el número de ancestros de un nodo i que son $raíces\_I$. Como es obvio, ese número nunca podrá ser mayor que la profundidad máxima del árbol, $prof(T)$. Por otra parte, aplicando el lema 5.4, tampoco podrá ser mayor que el número total de $raíces\_I$ de T, que a su vez está acotado por $hojas(T)$. Resulta entonces que

$$prof\_raíces\_I(i) \leq min\{prof(T), hojas(T)\} \;\; \forall i \; 1 \leq i \leq |T|$$

y en consecuencia, $prof\_raíces\_I(T) \leq min\{prof(T), hojas(T)\}$.

El siguiente lema relaciona los sumatorios que aparecen en la expresión de complejidad inicial con $prof\_raíces\_I(T)$.

Lema 5.5.
Sea T un árbol, se verifica que:

$$\sum_{i \in raíces\_I(t)} |T[i]| = \sum_{1 \leq i' \leq |T|} prof\_raíces\_I(i')$$

Demostración.
En el primer sumatorio, el nodo $s \in T$ será contado una vez por cada uno de sus ancestros que pertenezca a $raíces\_I(T)$. Es decir, si tenemos en cuenta la definición de $prof\_raíces\_I$, cada nodo s de T será contado $prof\_raíces\_I(s)$ veces, con lo que se cumple la igualdad anterior. □

Intuitivamente, lo que dice el lema es que la suma de los tamaños de los subárboles cuya raíz sea una $raíz\_I$ es igual a la suma del número de ancestros $raíz\_I$ que tiene cada uno de los nodos de T. El lema simplemente ofrece dos visiones del número de nodos que serán procesados por el algoritmo. Con esta relación en mente se demostrará el teorema que nos da expresión de la complejidad del algoritmo.

Teorema 5.2.
Dados dos árboles T y T′, la complejidad temporal del algoritmo 5.1 en el peor de los casos es:

$$O(|T| \times |T'| \times min\{prof(T), hojas(T)) \times min(prof(T'), hojas(T')\}$$

siendo $prof(T)$ y $prof(T')$ la profundidad de cada árbol, y $hojas(T)$ y $hojas(T')$, el número de nodos hoja en cada árbol. Siendo su complejidad espacial $O(|T| \times |T'|)$, en el peor de los casos.

Demostración.
Comenzaremos estudiando la complejidad espacial y después la temporal.

- Complejidad espacial

  El algoritmo utiliza una matriz permanente para guardar las distancias $distArbol(i, j)$, con $1 \leq i \leq |T|$ y $1 \leq j \leq |T'|$. También se utiliza una matriz temporal asociada a cada par de nodos $i \in raíces\_I(T)$ y $j \in raíces\_I(T')$, donde se almacenan los valores de $distBosque(l(i)..s, l(j)..t)$, con $l(i) \leq s \leq i$ y $l(j) \leq t \leq j$. Por lo tanto, cada una de estas matrices requiere un espacio $O(|T| \times |T'|)$ como máximo.

- Complejidad temporal

  El preprocesamiento necesario para construir los conjuntos $raíces\_I(T)$ y $raíces\_I(T')$ y las relaciones $l(i)$, $\forall \; 1 \leq i \leq T$ y $l(j)$, $\forall \; 1 \leq j \leq T'$ es lineal.

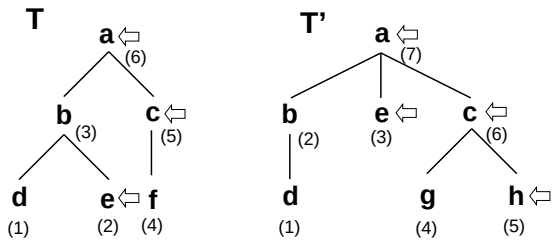


Figura 5.5: Arboles empleados en el ejemplo de cálculo de distancias.

El bucle más interno calcula la distancia $distArbol(i, j)$ entre cada par de nodos $i \in raíces\_I(T)$ y $j \in raíces\_I(T')$ en un tiempo proporcional a $|T[i]| \times |T'[j]|$. Dado que ese bucle es llamado para cada par de $raíces\_I$, el tiempo del bucle principal es:

$$\begin{array}{rl} & \sum_{i \in raíces\_I(T)} \sum_{j \in raíces\_I(T')} |T[i]| \times |T'[j]| \quad = \\ = & \sum_{i \in raíces\_I(T)} |T[i]| \ \times \ \sum_{j \in raíces\_I(T')} |T'[j]| \end{array}$$

Aplicando el lema 5.5, la expresión anterior será igual a:

$$\sum_{1 \leq i \leq |T|} prof\_raíces\_I(i) \ \times \sum_{1 \leq j \leq |T'|} prof\_raíces\_I(j)$$

Que, a su vez, es menor que:

$$|T| \times |T'| \times prof\_raíces\_I(T) \times prof\_raíces\_I(T')$$

Por las propiedades de $prof\_raíces\_I$, tenemos que la complejidad temporal del algoritmo es:

$$O(|T| \times |T'| \times min\{prof(T), hojas(T)) \times min(prof(T'), hojas(T')\}$$

□

5.3. Ejemplo práctico

Finalizaremos el presente capítulo presentando un ejemplo práctico que ilustra el funcionamiento del algoritmo para el cálculo de la distancia de edición entre árboles propuesto por Zhang y Shasha [65]. Aplicando los resultado teóricos descritos con anterioridad, calcularemos, paso a paso, la distancia de edición entre los dos árboles, T y T', de la figura 5.5. En dicha figura, se muestra la numeración en postorden de los nodos de cada uno de los árboles y se señalan sus respectivas $raíces\_I$ mediante flechas.

El algoritmo comienza tomando la primera $raíz\_I$ de T, el nodo $t[2]$ etiquetado como e, e inicia el cálculo de las distancias $distArbol$ con cada una de las $raíz\_I$ de T'. Dichas $raíces\_I$ son, en orden ascendente, $t'[3]$, etiquetada e, $t'[5]$, etiquetada h, $t'[6]$, etiquetada c, y $t'[7]$, etiquetada a. Para cada uno de estos nodos y sus respectivos subárboles se inicializa la correspondiente matriz temporal $distBosque$ tal como indica el lema 5.1. Después, se comienza a calcular, de modo ascendente, las distancias entre pares de subbosques, aplicando según el caso uno de los conjuntos de fórmulas que indica el lema 5.3. En el caso de distancias entre pares subárboles, una vez obtenido su valor, éste debe ser almacenado en la celda correspondiente de la matriz permanente $distArbol$.

La figura 5.6 muestra las matrices temporales de distancias entre bosques, $distBosque$, obtenidas durante el cálculo de la $distArbol$ entre el subárbol con raíz en $t[2]$ y los subárboles asociados a las $raíces\_I$ de T', $t'[3]$, $t'[5]$, $t'[6]$ y $t'[7]$, respectivamente. Para facilitar la interpretación de dichas distancias, se han

distBosque

	∅	*e..e*
∅	0	1
e..e	1	0

	∅	*h..h*
∅	0	1
e..e	1	1

	∅	*g..g*	*g..h*	*g..c*
∅	0	1	2	3
e..e	1	1	2	3

	∅	*d..d*	*d..b*	*d..e*	*d..g*	*d..h*	*d..c*	*d..a*
∅	0	1	2	3	4	5	6	7
e..e	1	1	2	2	3	4	5	6

distArbol

	d	*b*	*e*	*g*	*h*	*c*	*a*
d							
e	**1**	**2**	**0**	**1**	**1**	**3**	**6**
b							
f							
c							
a							

Figura 5.6: Distancias para $T[2]$ con $T'[2]$, $T'[5]$, $T'[6]$ y $T'[7]$.

distBosque

	∅	*e..e*
∅	0	1
f..f	1	1
f..c	2	2

	∅	*h..h*
∅	0	1
f..f	1	1
f..c	2	2

	∅	*g..g*	*g..h*	*g..c*
∅	0	1	2	3
f..f	1	1	2	3
f..c	2	2	3	2

	∅	*d..d*	*d..b*	*d..e*	*d..g*	*d..h*	*d..c*	*d..a*
∅	0	1	2	3	4	5	6	7
f..f	1	1	2	3	4	5	5	6
f..c	2	2	2	3	4	5	5	6

distArbol

	d	*b*	*e*	*g*	*h*	*c*	*a*
d							
e	1	2	0	1	1	3	6
b							
f	**1**	**2**	**1**	**1**	**1**	**3**	**6**
c	**2**	**2**	**2**	**2**	**2**	**2**	**6**
a							

Figura 5.7: Distancias para $T[5]$ con $T'[2]$, $T'[5]$, $T'[6]$ y $T'[7]$.

utilizado las etiquetas de los nodos en lugar de sus índices en postorden para delimitar los subbosques considerados en cada caso. En la misma figura se muestra la matriz permanente *distArbol* con las distancias entre subárboles obtenidas durante el procesamiento de los árboles con raíz en las *raíces_I* anteriores. Los valores $distBosque(l(i)..s, l(j)..t)$, con $l(s) = l(i)$ y $l(t) = l(j)$, que dan lugar a dichas distancias entre subárboles se señalan en sus respectivas matrices *distBosque* con un recuadro.

Para ilustrar como se efectúa el proceso de cálculo, revisaremos detenidamente el cálculo de la distancia entre las *raíces_I* $t[2]$ de T y $t'[6]$ de T'. En este caso, el subárbol de T con raíz en $t[2]$ contiene un único nodo, que es el propio $t[2]$. Por el contrario, el subárbol de T' contiene tres nodos, $t'[4]$, $t'[5]$ y $t'[6]$. Una vez inicializada la matriz *distBosque* de acuerdo al lema 5.1, el cálculo de las distancias entre subbosques comienza en la hoja más a la izquierda de ambos árboles, es decir, con los bosques $T[e..e]$ y $T[g..g]$. Dado que ambos son árboles, se aplica el primer conjunto de fórmulas del lema 5.3, utilizando las distancias $distBosque(\emptyset, 4., 4)$, $distBosque(2., 2, \emptyset)$ y $distBosque(\emptyset, \emptyset)$, obtenidas todas ellas durante la inicialización de la matriz temporal *distBosque*. La distancia resultante es 1, que se corresponde a la operación de cambio de etiqueta $e \rightarrow g$. Como se trata de una distancia entre árboles, su valor se copia en la celda correspondiente de la matriz permanente *distArbol*.

El siguiente subbosque de T' a tratar es $T'[g..h]$. El objetivo ahora es el cálculo de una distancia entre bosques y se emplea, por tanto, el segundo conjunto de fórmulas del lema 5.3. Se emplean las distancias $distBosque(\emptyset, g..h)$ y $distBosque(e..e, g..g)$ y la suma de $distBosque(\emptyset, g..g)$ y $distArbol(e, h)$. Los dos primeros valores fueron obtenidos durante la inicialización y en el paso que acabamos de describir, respectivamente. Respecto a los otros dos valores, $distBosque(\emptyset, g..g)$ es resultado de la inicialización y

distBosque *distArbol*

	$\emptyset$	$e..e$
$\emptyset$	0	1
$d..d$	1	**1**
$d..e$	2	1
$d..b$	3	**2**
$d..f$	4	3
$d..c$	5	4
$d..a$	6	**5**

	$\emptyset$	$h..h$
$\emptyset$	0	1
$d..d$	1	**1**
$d..e$	2	2
$d..b$	3	**3**
$d..f$	4	4
$d..c$	5	5
$d..a$	6	**6**

	$\emptyset$	$g..g$	$g..h$	$g..c$
$\emptyset$	0	1	2	3
$d..d$	1	**1**	2	**3**
$d..e$	2	2	2	3
$d..b$	3	**3**	3	**2**
$d..f$	4	4	4	4
$d..c$	5	5	5	5
$d..a$	5	**6**	6	**6**

	$\emptyset$	$d..d$	$d..b$	$d..e$	$d..g$	$d..h$	$d..c$	$d..a$
$\emptyset$	0	1	2	3	4	5	6	7
$d..d$	1	**0**	**1**	2	3	4	5	**6**
$d..e$	2	1	2	1	2	3	4	5
$d..b$	3	**2**	**1**	2	3	4	5	**6**
$d..f$	4	3	2	2	3	4	5	6
$d..c$	5	4	3	3	4	5	4	5
$d..a$	6	**5**	**4**	4	5	6	5	**4**

	d	b	e	g	h	c	a
d	**0**	**1**	**1**	**1**	**1**	**3**	**6**
e	1	2	0	1	1	3	6
b	**2**	**1**	**2**	**3**	**3**	**3**	**6**
f	1	2	1	1	1	3	6
c	2	2	2	2	2	2	6
a	**5**	**4**	**5**	**6**	**6**	**6**	**4**

Figura 5.8: Distancias para $T[6]$ con $T'[2]$, $T'[5]$, $T'[6]$ y $T'[7]$.

$distArbol(e, h)$ habrá sido almacenado en la matriz *distArbol* durante el cálculo de las distancias entre la *raíz_I* de T, $t[2]$, y la *raíz_I* de T', $t'[5]$. La mejor opción es utilizar el tercer valor, resultando una distancia 2. El último subbosque a procesar en T' es $T'[g..c]$, que coincide con el propio subárbol. De nuevo, se aplican las fórmulas para distancias entre árboles, empleando los valores de $distBosque(\emptyset, g..c)$, $distBosque(e..e, g..h)$ y $distBosque(\emptyset, g..h)$. Las mejores opciones son la segunda o la tercera y la distancia que resulta es 3. Al tratarse de una distancia entre árboles, este valor debe ser almacenado en la matriz *distArbol*.

El siguiente nodo de *raíz_I*(T) en ser procesado es $t[5]$, etiquetado como c. Los valores obtenidos con esta *raíz_I* se muestran en la figura 5.7. Como en el caso anterior, el subárbol asociado a este nodo debe compararse en orden ascendente con los subárboles de las *raíces_I* de T', $t'[3]$, $t'[5]$, $t'[6]$ y $t'[7]$. La situación es similar a la descrita anteriormente. La única diferencia es que, en este caso, el árbol $T[5]$ contiene dos subbosques, $T[f..f]$ y $T[f.c]$. Esto dará lugar a unas matrices *distBosque* mayores, pero no habrá diferencias en cuanto al cálculo en sí, puesto que ambos bosques contienen únicamente un árbol.

La última *raíz_I* de T en ser procesada es $t[6]$. Su subárbol coincide con la totalidad del árbol T y se compara, de nuevo, con los subárboles $T'[3]$, $T'[5]$, $T'[6]$ y $T'[7]$ de T'. Las distancias obtenidas se muestran en la figura 5.8. En este caso, examinaremos únicamente el cálculo de dos de las distancias entre bosques obtenidas durante el procesamiento de los subárboles $T[6]$ y $T'[7]$. Comenzaremos con la distancia entre los bosques $T[d..b]$ y $T'[d..b]$. Se trata de una distancia entre árboles y se usa el primer conjunto de fórmulas del lema 5.3. Podemos comprobar como en T los nodos a y b comparten su hoja más a la izquierda, el nodo d, y lo mismo sucede en el árbol T'. Así pues, se toman las distancias $distBosque(d..e, d..b)$, $distBosque(d..b, d..d)$ y $distBosque(d..e, d..d)$, optándose por la tercera opción cuyo coste es 1. Dicho valor es, además, guardado en la matriz *distArbol*. El otro caso que veremos es el de la distancia entre los bosques $T[d..c]$ y $T'[d..c]$. En este caso si se trata de bosques y se utilizan las fórmulas del segundo caso del lema 5.3. Tomamos las distancias $distBosque(d..f, d..c)$ y $distBosque(d..c, d..h)$ y la suma de $distBosque(d..b, d..e)$ y $distArbol(c, c)$. La mejor opción es la tercera, resultando una distancia de 4, que

se corresponde con la suma del coste de transformar el bosque $T[d..b]$ en $T'[d..e]$, cuyo coste es 2 , y el de transformar el árbol $T[f..c]$ en $T'[g..c]$, con un coste igual a 2.

Por último, señalaremos que una vez finalizado el algoritmo, la matriz *distArbol* contendrá las distancias de edición entre todos los pares de subárboles de T y T', incluida la distancia resultante entre ambos árboles T y T', que se muestra en la matriz *distArbol* de la figura 5.8 resaltada con un rectángulo doble. Dicha distancia fue obtenida durante el procesamiento de las *raíces_I* $t[6]$ de T y $t[7]$ de T' y, de hecho, coincide con la distancia $distBosque(d..a, d..a)$, calculada durante ese procesamiento.

Capítulo 6

Reconocimiento aproximado de patrones en árboles

El reconocimiento aproximado de patrones sobre árboles es una extensión del concepto de reconocimiento aproximado sobre cadenas. En este capítulo se presenta una generalización del algoritmo de Zhang y Shasha [65] presentado en el capítulo 5, propuesta por los mismos autores en [66].

El interés de este tipo de técnicas de reconocimiento aproximado de patrones, en oposición a las técnicas de reconocimiento exacto, es que permiten trabajar con árboles patrón más generales, en los cuales se podrán omitir detalles estructurales que no sean relevantes. Se consigue de esta forma un mecanismo de identificación de estructuras más flexible y eficaz de cara a las posibles aplicaciones prácticas.

Se comenzará exponiendo distintas posibilidades a la hora de generalizar el reconocimiento aproximado de patrones en cadenas para extenderlo al caso del reconocimiento aproximado de árboles. Una vez estudiadas estas ampliaciones, se presentarán los fundamentos teóricos y se describirán los algoritmos que se encargarán de aplicar este tipo de reconocimiento de patrones.

6.1. Definición del problema

De modo general, el problema del reconocimiento aproximado de árboles se puede formular de la siguiente forma: dados dos árboles etiquetados y ordenados, donde se interpreta uno de ellos como un árbol patrón, P, y el otro como un árbol dato, D[1]. Se desea calcular la distancia de edición entre ambos, en el supuesto de que no se exija una correspondencia exacta del árbol patrón, P, con la totalidad del árbol, D. Esto es, se permite que el árbol P encaje sólo con parte del árbol D, de forma que las partes de D no incluidas no afectarán a la distancia resultante.

A la hora de generalizar el reconocimiento aproximado sobre cadenas, surgen diferentes aproximaciones en cuanto a la forma de extender el problema al caso de los árboles etiquetados ordenados, tal como señalan Zhang y Shasha [66]. Las técnicas empleadas usualmente para permitir comparaciones aproximadas entre árboles son dos. En primer lugar, se permitirá que se eliminen libremente subárboles del árbol dato D, sin que este borrado de subárboles incremente el valor de la distancia con respecto al árbol patrón P. En segundo lugar, se podrán incluir símbolos VLDC en el árbol patrón, que permitirán especificar que determinadas porciones del árbol patrón no son relevantes. En este punto es posible definir distintos tipos de símbolos con significados diferentes, tal como se verá más adelante.

Se procederá a exponer las distintas alternativas en la definición del reconocimiento aproximado de árboles. En primer lugar, se estudian las posibilidades a la hora de introducir la eliminación de subárboles en al árbol dato. Seguidamente, se presentan posibles definiciones de símbolos VLDC y como afectará su inclusión en el árbol patrón al algoritmo de reconocimiento aproximado.

[1] En este caso existe una notable diferencia entre el papel que juega el árbol patrón P y el árbol dato D. Por ello. se usará la convención de emplear las letras P y D para nombrarlos, en lugar de T y T' como venía siendo habitual.

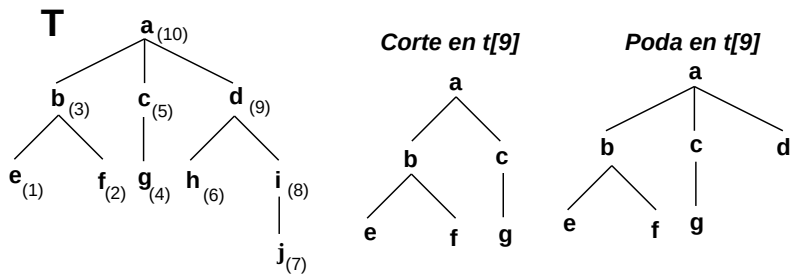


Figura 6.1: Comparación entre *corte* y *poda* de subárboles.

6.1.1. Eliminación de subárboles en el árbol dato

La primera opción para definir el reconocimiento aproximado de árboles es incluir la posibilidad de eliminar subárboles en el árbol dato D, sin afectar a la distancia final entre el árbol patrón y el árbol dato D' resultante. Esta eliminación de subárboles puede verse como una extensión de la eliminación de prefijos en la cadena dato que se utilizaba en el reconocimiento aproximado de cadenas. En este caso, la contrapartida a la eliminación de prefijos en la cadena dato, es la eliminación de conjuntos de subárboles del árbol homónimo. Dentro del marco del algoritmo propuesto por Zhang y Shasha [65], se pueden considerar dos posibilidades en cuanto a la eliminación de subárboles:

- *Cortar un nodo i de un árbol T*. Ello significa eliminar el nodo $t[i]$ y todos sus descendientes. Esto es, se elimina completamente el subárbol con raíz en $t[i]$. Formalmente tenemos que dado un árbol T, *cortar un nodo i* da lugar a un nuevo árbol T' definido de la siguiente forma:

$$T' = \{\ j \mid j \in T \text{ y } j \notin T[l(i)..i]\ \}$$

- *Podar un nodo i de un árbol T*. Ello significa eliminar todos los descendientes del nodo $t[i]$, sin incluir a ese nodo. Esto es, se elimina los nodos del subárbol con raíz en $t[i]$, pero no a $t[i]$. Formalmente tenemos que dado un árbol T, *podar un nodo i* da lugar a un nuevo árbol T' definido de la siguiente forma:

$$T' = \{\ j \mid j \in T \text{ y } j \notin T[l(i)..i-1]\ \}$$

Ambas opciones se ilustran en la figura 6.1. En nuestro caso, la distinción entre corte y poda es simplemente una cuestión de matiz. Estructuralmente, realizar la poda de un nodo $t[i]$ es equivalente a realizar un conjunto de cortes en todos y cada uno de los hijos de $t[i]$. La razón de diferenciar ambas posibilidades viene del hecho de que si estas operaciones tuviesen un coste asociado, si sería relevante distinguir entre podar en un nodo y que permaneciera el nodo raíz o realizar un corte, eliminando esa raíz. Como en el caso que aquí se estudia no se tienen en consideración el coste de este tipo de eliminación de subárboles a la hora de determinar la distancia entre árbol patrón y árbol dato, ambas opciones serán equivalentes. Por lo tanto, de ahora en adelante se trabajará únicamente con cortes de subárboles en el árbol dato.

Para definir formalmente la *distancia con cortes* entre dos árboles, se introduce el concepto de *conjunto de cortes consistentes*.

Definición 6.1.
Sea T un árbol y sea CCC *un conjunto de nodos de T. Se dirá que* CCC *es un* conjunto de cortes consistentes *si verifica lo siguiente:*

- $t[i] \in$ CCC *implica que* $1 \leq i \leq |T|$.

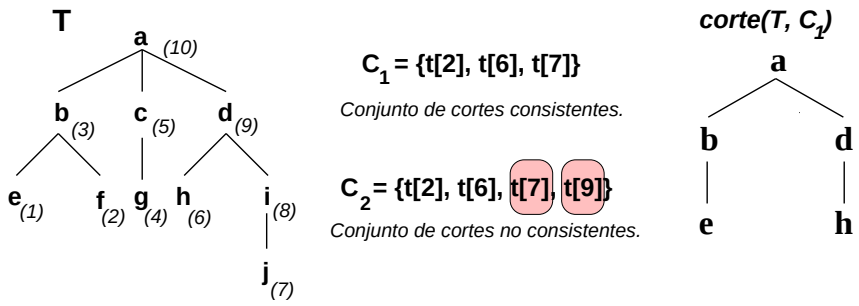


Figura 6.2: Ejemplo de cortes consistentes.

- $t[i_1], t[i_2] \in$ CCC *implica que ninguno de los dos es un ancestro del otro.*

Se usará la notación cortar(T, CCC)*, siendo* CCC *un conjunto de cortes consistentes, para representar el árbol resultante de la eliminación del árbol T de los subárboles asociados a los nodos contenidos en* CCC

En la figura 6.2 se muestra un ejemplo de un conjunto de cortes consistentes, y de un conjunto de cortes no consistentes. La razón de que el conjunto C_2 no sea consistente se debe a que el nodo $t[3]$ es el padre de $t[2]$, contradiciendo la definición anterior. También se muestra el árbol $T_1 = cortar(T, C_1)$, resultado de aplicar cortes en los nodos del conjunto C_1. Utilizando estos conceptos, se puede definir formalmente la *distancia de edición con cortes* de la forma que sigue:

Definición 6.2.
Dados un árbol patrón P y un árbol dato D, y siendo $subarboles(D)$ el conjunto de todos los posibles conjuntos de cortes consistentes que se pueden construir en D, se define la distancia de edición con cortes *entre los árboles P y D como:*

$$d_{cortes}(P, D) = min_{C \in subarboles(D)}\{d(P, cortar(D, C))\}$$

Intuitivamente, esta distancia con cortes se corresponde con la distancia de edición obtenida entre el árbol patrón P y el resultado de aplicar sobre el árbol D el conjunto de cortes consistentes que genera el nuevo árbol D' que sea más próximo al patrón P .

Como se verá con más detalle en secciones posteriores, no es necesario construir explícitamente todos los posibles conjuntos de cortes consistentes, ni calcular todas las distancias asociadas para obtener la mejor distancia. En este punto, Zhang y Shasha [66] proponen una variante de su algoritmo de cálculo de distancias [65] que permite calcular la distancia con cortes manteniendo la misma complejidad espacial y temporal.

6.1.2. Símbolos VLDC en el árbol patrón

Tal y como se había comentado para el caso de las cadenas de caracteres, una técnica alternativa empleada para permitir comparaciones aproximadas es la inclusión de símbolos VLDC en el árbol patrón. A la hora de generalizar este concepto al caso de los árboles surgen, de nuevo, varias posibilidades en función de la interpretación que se asocie a estos símbolos VLDC. En el caso de árboles etiquetados tenemos la siguiente definición general.

Definición 6.3 (símbolo VLDC)**.**
Se define un símbolo VLDC *como un símbolo especial que podrá aparecer en los nodos de un árbol patrón P y que no pertenece al alfabeto de las etiquetas Σ. De cara al cálculo de la distancia, cada uno de estos*

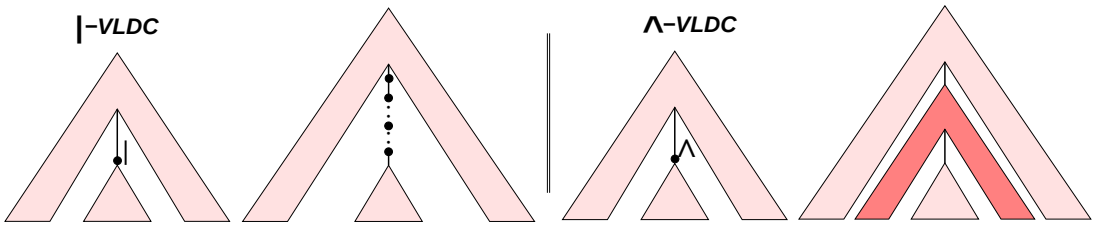


Figura 6.3: Representación esquemática de los símbolos VLDC.

símbolos VLDC *podrá ser sustituido por una porción del árbol dato D, sin que ello afecte al valor final de la distancia de edición con respecto al patrón.*

En función de como se especifique la sustitución que un símbolo VLDC realiza con respecto al árbol dato surgen diferentes opciones. Siguiendo a Zhang y Shasha [66] se definirán dos símbolos VLDC para incluir en los árboles patrón: el $\wedge$-VLDC[2], representado por el símbolo "$\wedge$z el |-VLDC, representado por una barra vertical "|". Tal y como se muestra en la figura 6.3, el significado de estos símbolos es el siguiente:

Definición 6.4.
Dados un árbol patrón P y un árbol dato D, se permitirá la inclusión de los siguientes símbolos VLDC *en los nodos del árbol P, con el significado que se indica a continuación:*

- *El* |-VLDC*, "|", sustituye a una secuencia consecutiva de nodos que formen parte de un camino desde la raíz del árbol dato hasta una de sus hojas.*
- *El* $\wedge$-VLDC*, "$\wedge$", además de sustituir a esa secuencia de nodos, incluye a todos los subárboles que emanan de los nodos de ese camino, con la posibilidad de excluir de esa sustitución a parte de los subárboles correspondientes a los hijos del último nodo incluido en el camino.*

El significado intuitivo del |-VLDC es directo. Se corresponde, simplemente, con la sustitución del nodo etiquetado "|" por una secuencia consecutiva de nodos tomados de uno de los caminos que se pueden construir en el árbol dato. La interpretación del $\wedge$-VLDC es un poco más compleja, ya que no se especifica estrictamente cómo es la selección de los subárboles del nivel más bajo que van a ser incluidos en la sustitución. La figura 6.4 aclara estos detalles. En ella, $d[u]$ es el último nodo de uno de los caminos de D, y $d[h_1], d[h_2], \ldots, d[h_k]$ son sus hijos. El $\wedge$-VLDC sustituirá a los nodos presentes en el camino desde la raíz hasta $d[u]$ y a los subárboles que surgen de estos nodos. En el nodo $d[u]$ la sustitución también incluirá a los subárboles con raíz en $d[h_1], \ldots, d[h_i]$ y en $d[h_j], \ldots, d[h_k]$, con $1 \leq i \leq j \leq k$. En ese caso, el conjunto de subárboles $d[h_{i+1}], \ldots, d[h_{j-1}]$, que eventualmente podría ser vacío, no estaría afectado por el $\wedge$-VLDC, y todos sus nodos serían susceptibles de participar en operaciones de edición y, en consecuencia, afectarán a la distancia final.

La diferencia entre ambos símbolos VLDC está en sí en la sustitución se incluyen sólo los nodos que forman parte de un camino o sí se añaden también los subárboles que emanan de ellos. Intuitivamente, el $\wedge$-VLDC es el menos restrictivo de los dos. En la práctica, al usar $\wedge$-VLDC no será necesario tener en cuenta los posibles nodos intermedios que puedan aparecer ni los subárboles que tengan asociados. En el caso del |-VLDC si es necesario tenerlos en cuenta, ya que los costes de las posibles inserciones o borrados que se aplicarán sobre los subárboles que emanan de los nodos intermedios si afectarían a la distancia final.

Para formalizar la idea de que los símbolos VLDC de un árbol patrón reemplazan porciones del árbol dato, se introduce el concepto de *sustitución*-VLDC .

[2]En el artículo original [66] se emplean los términos anglosajones *umbrella*-VLDC y *path*-VLDC , como consecuencia de la forma de las porciones del árbol dato que sustituyen.

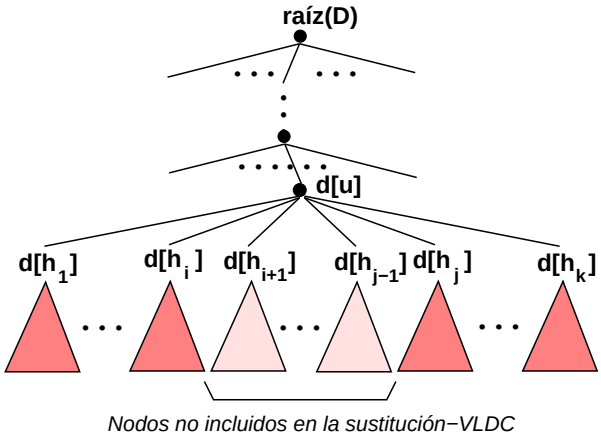


Figura 6.4: Sustitución de un ∧-VLDC.

Definición 6.5 (sustitución-VLDC)**.**
Dados un árbol patrón P que incluya símbolos VLDC *"|", y "∧" y un árbol dato D. Una* sustitución-VLDC *s sobre P reemplazará cada nodo de P etiquetado con "|" por una secuencia de nodos pertenecientes a uno de los posibles caminos que se inician en la raíz de D, y reemplazará cada nodo etiquetado con "∧" por los nodos pertenecientes a un fragmento de uno de esos caminos, a los que se incluirán los subárboles que emanen de ellos.*
*Se usará la notación $\bar{P} = sust$-VLDC (P, D, s) para representar al árbol $\bar{P}$ resultante de aplicar una sustitución-*VLDC *s sobre un árbol patrón P, con respecto a un árbol dato D.*

Como resultado de aplicar una sustitución-VLDC s a un árbol patrón P, se producirá un nuevo árbol $\bar{P}$ sin símbolos VLDC. Como condición adicional, se requiere que la sustitución de los símbolos VLDC tenga coste cero. Para conseguir esto se exige que los nodos añadidos para crear el árbol $\bar{P}$ sean tomados del conjunto de nodos presentes en el árbol D. De esta forma, al calcular la distancia final, la correspondencia resultante entre $\bar{P}$ y D hará que los nodos introducidos en $\bar{P}$ como consecuencia de la sustitución-VLDC se correspondan exactamente con los nodos originales de D.

En el siguiente ejemplo se muestra esta condición adicional. Se puede ver como se sustituyen los símbolos VLDC de P por nodos de D, de forma que en el cálculo de la distancia resultante entre $\bar{P}$ y D, sólo se consideren los nodos de P que no estaban etiquetados con símbolos VLDC.

Ejemplo 6.1.
*La figura 6.5 muestra dos sustituciones-*VLDC*, la primera para el caso de un patrón con un único* |-VLDC *y la segunda para un patrón con un* ∧-VLDC*. En el primer caso la distancia resultante es 1, que se corresponde con sustituir el símbolo "|" del patrón por la secuencia de nodos* {`c,e`}*, y realizar la operación de inserción del nodo* `f`*, $\varepsilon \rightarrow f$. En el ejemplo que incluye el* ∧-VLDC *la distancia sería 0, dado que el árbol resultante de la sustitución-*VLDC *coincide exactamente con el árbol dato.*

Utilizando el concepto de sustitución-VLDC, la *distancia de edición con símbolos* VLDC se define de la siguiente forma:

Definición 6.6.
Dados un árbol patrón P que incluya símbolos VLDC *y un árbol dato D. Siendo S el conjunto de todas las posibles sustituciones-*VLDC *para P y D, la* distancia de edición con símbolos VLDC entre P y D*, se definirá como:*

$$d_{\text{VLDC}}(P, D) = min_{s \in S}\{d(\bar{P}, D) | \bar{P} = sust\text{-VLDC } (P, D, s)\}$$

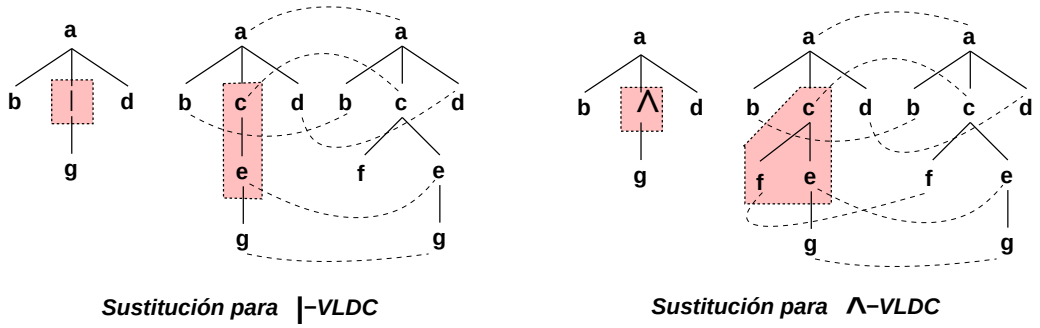


Figura 6.5: Ejemplos de sustituciones-VLDC.

Es decir, la distancia de edición en presencia de símbolos VLDC entre P y D, es la distancia de edición que se podría obtener entre el árbol patrón resultante $\bar{P}$ y el árbol dato D, en caso de aplicar la mejor sustitución-VLDC posible. Esto es, se utiliza la sustitución-VLDC que origine un árbol $\bar{P}$ más próximo a D.

6.2. Cálculo de la distancia edición con símbolos VLDC

En esta sección se presentará el cálculo de la distancia entre árboles para el caso en el que los árboles patrón incluyan símbolos VLDC. En [66] Zhang y Shasha parten del algoritmo básico para el cálculo de la distancia de edición entre árboles [65], mostrado en el capítulo 5, definiendo nuevos conjuntos de cálculos que tengan en cuenta el funcionamiento de las sustituciones-VLDC para los dos símbolos VLDC considerados, el |-VLDC y el ∧-VLDC.

Se comenzará repasando el mecanismo básico de cálculo de distancias presentado en el capitulo 5. Salvo que se indique expresamente, se mantendrán las definiciones, la notación y las convenciones introducidas en dicho capítulo. Así, se seguirán utilizando las notaciones $distBosque(i'..i, j'..j)$ y $distArbol(i, j)$, que ahora denotarán distancias de edición con símbolos VLDC. Esto es, representan, respectivamente, la distancia entre los bosques $P[i'..i]$ y $D[j'..j]$ y la distancia entre los subárboles $P[i]$ y $D[i]$, suponiendo que se haya aplicado la sustitución-VLDC óptima sobre los símbolos VLDC presentes en el árbol P.

Se verá que el cálculo de distancias entre subbosques es independiente del hecho de que en el árbol patrón se incluyan o no símbolos VLDC. Será en el caso de las distancias entre pares de subárboles, cuando se deberá diferenciar entre tres formas distintas de calcular las distancias, en función del tipo de etiqueta asociada a la raíz del subárbol de P que se esté considerando. Cuando la raíz del subárbol patrón no sea un símbolo VLDC seguirá siendo válido el conjunto de cálculos usado en el algoritmo original [65]. Para los casos en los que esas raíces sean símbolos VLDC se utilizarán dos nuevos conjuntos de fórmulas, uno para los |-VLDC y otro para los ∧-VLDC.

6.2.1. Cálculo básico para subárboles y bosques

Antes de revisar el cálculo de distancias presentado en el capítulo 5 y determinar cuales de esos conjuntos de fórmulas siguen siendo válidos en el caso de árboles patrón con símbolos VLDC, es necesario definir como se comportarán los nodos etiquetados con símbolos VLDC ante las operaciones de edición y cuales serán los costes correspondientes.

Dado que los símbolos VLDC sólo pueden aparecer en el árbol patrón P, sólo es necesario determinar el coste de las operaciones de borrado y cambio de etiqueta. Se supondrá que el coste del borrado o cambio de etiqueta de un nodo VLDC será siempre cero. Esto es, $\gamma(p[s] \rightarrow \varepsilon) = 0$ y $\gamma(p[s] \rightarrow d[t]) = 0$, cuando la etiqueta del nodo $p[s]$ sea "|" ó "∧". Se consigue así que la eliminación o cambio de símbolos VLDC no afecte a la distancia final. Teniendo en cuenta estas consideraciones, se verá que el lema 5.1, presentado

en el capítulo 5, sigue siendo válido en el caso de que el árbol patrón P contenga símbolos VLDC. Para aclarar esta afirmación se enuncia de nuevo el lema y se razona su validez teniendo en cuenta la presencia de símbolos VLDC.

Lema 6.1.
Sean P y D árboles y sean $i \in P$ y $j \in D$ dos de sus nodos, entonces:

1. $distBosque(\emptyset, \emptyset) = 0$.
2. $distBosque(P[l(i)..s], \emptyset) = distBosque(P[l(i)..s-1], \emptyset) + \gamma(p[s] \rightarrow \varepsilon)$.
3. $distBosque(\emptyset, D[l(j)..t]) = distBosque(\emptyset, D[l(j)..t-1]) + \gamma(\varepsilon \rightarrow d[t])$.

donde $p[s] \in desc(i)$ y $d[t] \in desc(j)$.

Demostración.
El razonamiento para los casos *1* y *3* es el mismo que el utilizado en el capítulo 5 durante la demostración del lema 5.1. En el caso *2*, se desea que los símbolos VLDC no afecten al valor de la distancia. Esto es, cuando $p[s] = |$ ó $p[s] = \wedge$ debe verificarse $distBosque(p[l(i)..s, \emptyset) = distBosque(p[l(i)..s-1], \emptyset)$. Dado que se parte de que $\gamma(| \rightarrow \varepsilon) = 0$ y $\gamma(\wedge \rightarrow \varepsilon) = 0$, se puede mantener el término $\gamma(p[s] \rightarrow \varepsilon)$ en el lado derecho de la expresión *2*, puesto que en el caso de aplicarse a símbolos VLDC no afectará al valor final. □

Una vez que se ha comprobado que los cálculos de inicialización del algoritmo básico de Zhang y Shasha [65] siguen siendo válidos, se estudiarán los otros dos tipos de cálculos, para distancias entre subbosques y para distancias entre subárboles. Para el caso del cálculo de la distancia entre subbosques, el conjunto de fórmulas del algoritmo básico, presentadas en el lema 5.3, sigue siendo correcto. De nuevo, para facilitar la comprensión, se presentan estas fórmulas en el siguiente lema y se demuestra su validez para el caso de árboles patrón con símbolos VLDC.

Lema 6.2.
Sean P y D árboles, sean $p[i]$ y $d[j]$ dos de sus nodos, y sean $p[s] \in desc(p[i])$ y $d[t] \in desc(d[j])$. Si $l(s) \neq l(i)$ o $l(t) \neq l(j)$, entonces:

$$distBosque(l(i)..s, l(j)..t) = \ min \begin{cases} distBosque(l(i)..s-1, l(j)..t) + \gamma(p[s] \rightarrow \varepsilon), \\ distBosque(l(i)..s, l(j)..t-1) + \gamma(\varepsilon \rightarrow d[t]), \\ distBosque(l(i)..l(s)-1, l(j)..l(t)-1) + distArbol(s,t) \end{cases}$$

Demostración.
Pueden distinguirse dos situaciones:

- Caso 1: $p[s] \neq |$ y $p[s] \neq \wedge$

  En este caso el último nodo del bosque $p[l(i)..s]$ no es un VLDC. En caso de existir símbolos VLDC en ese bosque ya se habrá encontrado la sustitución-VLDC más apropiada en pasos anteriores del algoritmo y habrá sido tenida en cuenta para el cálculo de las distancias precedentes, $distBosque(l(i)..s-1, l(j)..t)$, $distBosque(l(i)..s, l(j)..t-1)$ y $distBosque(l(i)..l(s)-1, l(j)..l(t)-1)$.

  Por lo tanto, al no estar afectado el nodo $p[s]$ por un símbolo VLDC, el razonamiento aquí es idéntico al expuesto en la demostración de los lemas 5.2 y 5.3.

- Caso 2: $p[s] = |$ ó $p[s] = \wedge$

  En este caso si se debe manejar el símbolo VLDC $p[s]$. Supondremos que M representa a una correspondencia de coste mínimo. Dado que en el subárbol patrón pueden existir símbolos VLDC, esta correspondencia M se establece entre el subbosque $\bar{P}[l(i)..s]$, obtenido a partir de $P[l(i)..s]$ como resultado de aplicar la mejor sustitución-VLDC a sus símbolos VLDC, incluido $p[s]$, y el subbosque $D[l(j)..t]$.

Teniendo en cuenta las posibilidades a la hora de sustituir el símbolo VLDC $p[s]$ y como se vea afectado el nodo $d[t]$ por la correspondencia M, se distinguen tres posibilidades:

(a) En este caso se supone que en la mejor sustitución-VLDC, $p[s]$ es sustituido por un conjunto de nodos vacío, esto es, $distBosque(l(i)..s, l(j)..t) = distBosque(l(i)..s-1, l(j)..t)$. Dado que $\gamma(|\rightarrow \varepsilon) = 0$ y $\gamma(\wedge \rightarrow \varepsilon) = 0$, el lado derecho de la igualdad puede escribirse como $distBosque(l(i)..s-1, l(j)..t) + \gamma(p[s] \rightarrow \varepsilon)$ sin afectar al valor de la distancia [3].

(b) En este caso se tiene que, en la mejor sustitución, $p[s]$ es sustituido por un conjunto de nodos no vacío [4]. Se supone, además, que el nodo $d[t]$ no está afectado por la correspondencia M. Esto significa que en esa correspondencia óptima el nodo $d[t]$ debe ser insertado, por lo que

$$distBosque(l(i)..s, l(j)..t-1) = distBosque(l(i)..s, l(j)..t-1) + \gamma(\varepsilon \rightarrow d[t])$$

(c) En el tercer caso se supone que $p[s]$ tiene una sustitución-VLDC no vacía y que el nodo $d[t]$ está afectado por la correspondencia M. Por lo tanto, $d[t]$ participa en una operación de cambio de etiqueta. Supongamos que se aplica la sustitución-VLDC correspondiente a $p[s]$, obteniéndose un nuevo subárbol que reemplaza al árbol $P[s]$, cuya raíz será el nodo N, tal y como se muestra en la figura 6.6. La correspondencia M se establece entre el subbosque resultado de la sustitución, $\bar{P}[l(i)..N]$, y el subbosque $D[l(j)..t]$.

Aplicando el mismo razonamiento basado en el mantenimiento del orden ancestro-descendiente utilizado en los lemas 5.2 y 5.3, se llega a que en M debe existir una operación de cambio de etiqueta entre $\bar{p}[N]$ y $d[t]$ y que, por lo tanto

$$\begin{aligned} distBosque(\bar{P}[l(i)..N], D[l(j)..t]) = \quad & distBosque(\bar{P}[l(i)..l(N)-1], D[l(j)..l(t)-1]) + \\ & distArbol(\bar{P}[N], D[t]) \end{aligned}$$

A partir de esta última expresión, dado que $\bar{P}[l(i)..l(N)-1] = P[l(i)..l(s)-1]$[5] y que, en base a la definición de sustitución-VLDC, $distArbol(P[s], D[t]) = distArbol(\bar{P}(N), D[t])$, resulta que:

$$distBosque(l(i)..s, l(j)..t) = distBosque(l(i)..l(s)-1, l(j)..l(t)-1) + distArbol(s, t)$$

Como en ocasiones anteriores, estos tres casos cubren todas las posibles forma de obtener la distancia deseada y, por lo tanto, bastará con calcular el mínimo de esos tres valores.

□

Una conclusión importante de este lema y, en concreto, del razonamiento empleado en el tercer caso de la demostración, es que la tarea de determinar cual es la mejor sustitución para cada símbolo VLDC formará parte del cálculo de la distancia entre pares de subárboles. Revisando de nuevo las fórmulas definidas en el lema 6.2, en el caso de que $p[s]$ sea un símbolo VLDC, la tercera expresión, $distBosque(l(i)..s, l(j)..t) = distBosque(l(i)..l(s)-1, l(j)..l(t)-1) + distArbol(s, t)$, simplemente utiliza el término $distArbol(s, t)$. Puesto que se supone que dicha distancia ya habrá sido calculada previamente tomando en consideración la mejor sustitución posible para el nodo $p[s]$.

Intuitivamente, el hecho de que la identificación de sustituciones-VLDC se realice sólo durante el cálculo de distancias entre subárboles está en consonancia con el tipo de símbolos VLDC definidos. Tal como se indica en la definición 6.5, la forma en que ambos símbolos, "|" y "∧", son sustituidos sólo considera nodos tomados del subárbol del árbol dato con respecto al cual se calcula la distancia. Es decir, en la sustitución de símbolos VLDC no intervienen nodos de los subárboles dato que los rodean. Puede decirse, entonces, que la sustitución de símbolos VLDC es una operación local a un subárbol.

[3] Intuitivamente, sustituir un VLDC por un árbol vacío es equivalente a eliminarlo.

[4] Esto es, se sustituye por una parte de un árbol si es un ∧-VLDC o por una secuencia de nodos si es un |-VLDC.

[5] El contexto izquierdo de los subárboles $\bar{P}[N]$ y $P[s]$ es el mismo.

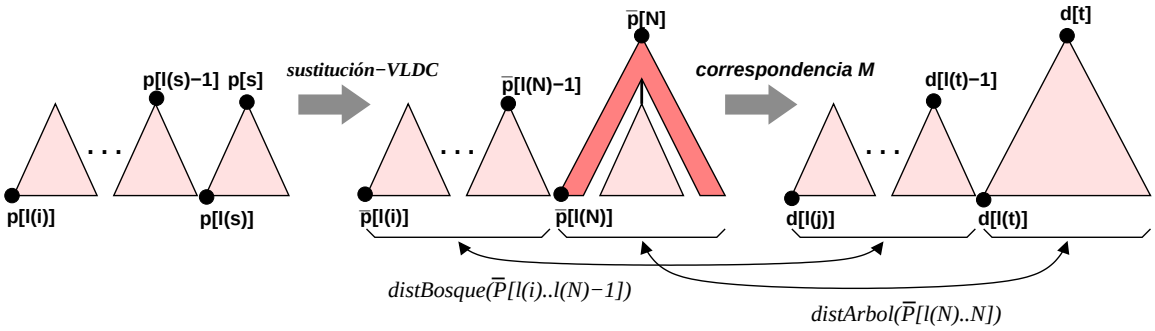


Figura 6.6: Caso 2(c) de la demostración del lema 6.2.

Una vez que se ha visto que la decisión de como realizar las sustituciones de símbolos VLDC se determina durante el cálculo de la distancia entre pares de subárboles, se estudian ahora las distintas posibilidades a la hora de calcular esas distancias. Se considerarán tres situaciones, cada una con un conjunto de fórmulas distinto.

En primer lugar, se presenta en el siguiente lema, la forma en que se realizará el cálculo de distancias entre subárboles cuando la raíz del subárbol patrón no esté etiquetada con un símbolo VLDC. Los cálculos definidos por este lema son idénticos a los del primer caso del lema 5.3 del capítulo 5. Se vuelven a mostrar de nuevo de cara a facilitar la posterior explicación del algoritmo de comparación.

Lema 6.3.
Dados dos árboles P y D, dados los nodos $p[i]$ y $d[j]$, y sus descendientes $p[s] \in des(i)$ y $d[t] \in des(j)$. Suponiendo que $l(s) = l(i)$ y $l(t) = l(j)$ y que $p[s] \neq |$ y $p[s] \neq \wedge$, entonces:

$$distBosque(l(i)..s, l(j)..t) = \ min \begin{cases} distBosque(l(i)..s-1, l(j)..t) + \gamma(p[s] \rightarrow \varepsilon), \\ distBosque(l(i)..s, l(j)..t-1) + \gamma(\varepsilon \rightarrow d[t]), \\ distBosque(l(i)..s-1, l(j)..t-1) + \gamma(p[s] \rightarrow d[t]) \end{cases}$$

Demostración.
La demostración es la misma que en el caso del lema 5.3. La única diferencia es que, en este caso, se considera que los valores para las distancias $distBosque(l(i)..s-1, l(j)..t)$, $distBosque(l(i)..s, l(j)..t-1)$ y $distBosque(l(i)..s-1, l(j)..t-1)$ fueron calculados en base a las sustituciones-VLDC óptimas para cada uno de los nodos de $P[s]$ etiquetados con símbolos VLDC. □

Los otros dos casos a considerar, esto es, el cálculo de distancias entre subárboles cuando la raíz del subárbol patrón es uno de los dos símbolos VLDC, constituyen la parte más importante del algoritmo de Zhang y Shasha [66] para el reconocimiento aproximado de patrones sobre árboles. Ambos casos se estudiarán con detalle en las secciones siguientes, y se verá como se combinan con los resultados de los lemas anteriores para construir el algoritmo de cálculo de distancias con símbolos VLDC.

6.2.2. Distancia con |-VLDC

La primera posibilidad que se va a estudiar dentro del cálculo de la distancia entre subárboles con respecto a subárboles patrón con símbolos VLDC en la raíz, es el caso en que ese símbolo sea un |-VLDC. Este será el caso más sencillo de los dos que se pueden presentar. Como se verá más adelante, para manejar distancias donde esté involucrado un $\wedge$-VLDC no basta con utilizar las distancias de edición de nodos consideradas anteriormente, sino que es necesario el cálculo de distancias auxiliares, con restricciones adicionales. Es por ello por lo que se comenzará estudiando el manejo de los |-VLDC. La base del cálculo de la distancia de edición bajo estas condiciones es el lema que sigue:

Lema 6.4.
Sean P y D árboles, sean $p[i]$ y $d[j]$ dos de sus nodos y sean $p[s] \in des(i)$ y $d[t] \in des(j)$. Suponiendo que $l(s) = l(i)$ y $l(t) = l(j)$, y que $p[s] = |$, entonces:

$$distBosque(l(i)..s, l(j)..t) = \ min \begin{cases} distBosque(l(i)..s-1, l(j)..t) + \gamma(p[s] \to \varepsilon), \\ distBosque(l(i)..s, l(j)..t-1) + \gamma(\varepsilon \to d[t]), \\ distBosque(l(i)..s-1, l(j)..t-1) + \gamma(p[s] \to d[t]), \\ distBosque(\emptyset, l(j)..t-1) + \\ \qquad min\ _{t_k}\{distArbol(s, t_k) - distArbol(\emptyset, t_k) \mid 1 \le k \le n_t\} \end{cases}$$

donde los nodos $d[t_k]$, con $1 \le k \le n_t$, son los n_t hijos del nodo $d[t]$[6].

Demostración.
Como en demostraciones anteriores, se considera que M es la correspondencia de coste mínimo entre el bosque $P[l(i)..s]$ y el bosque $D[l(j)..t]$. Esa correspondencia habrá sido determinada teniendo en cuenta las mejores sustituciones para los símbolos VLDC que pudieran estar incluidos en $P[l(i)..s]$. Al igual que en el caso *2* de la demostración del lema 6.2, existen tres posibilidades:

1. $p[s]$ es reemplazado por un conjunto vacío de nodos.
2. $p[s]$ es reemplazado por un conjunto no vacío de nodos y el nodo $d[t]$ no está afectado por las operaciones de cambio de etiqueta de la correspondencia M.
3. $p[s]$ es reemplazado por un conjunto no vacío de nodos y el nodo $d[t]$, de acuerdo con M, participa en una operación de cambio de etiqueta.

En los casos *1* y *2*, el razonamiento es el mismo que el empleado en la demostración del lema 6.2 y las fórmulas obtenidas son idénticas. Para el caso *3* hay dos situaciones posibles:

(a) El nodo $d[t]$ es una hoja

En este caso tenemos $t = l(j)$. Por lo tanto |-VLDC de $p[s]$ sólo puede ser reemplazado por el nodo $d[t]$. De esta forma, y dado que $\gamma(| \to [t]) = 0$, pude reescribirse el lado derecho de la expresión $distBosque(l(i)..s, l(j)..t) = distBosque(l(i)..s-1, l(j)..t-1)$ como $distBosque(l(i)..s-1, l(j)..t-1) + \gamma(p[s] \to d[t])$, que se corresponde con la tercera expresión del cálculo de la distancia mínima.

(b) El nodo $d[t]$ tiene hijos

Puesto que $d[t]$ no es una hoja, es decir, $t \neq l(j)$, el |-VLDC $p[s]$ debe ser reemplazado por una secuencia de nodos extraída de uno de los caminos que arrancan de la raíz del subárbol $D[t]$. Se supondrá que $d[u]$ es el último nodo de esa secuencia. Suponiendo que los hijos de $d[t]$ son, de izquierda a derecha, los nodos $d[t_1], d[t_2], \ldots, [t_{n_t}]$, se pueden distinguir, tal y como se muestra en la figura 6.7, los dos subcasos siguientes:

(I) $u = t$

Puesto que $d[t]$ y $d[u]$ coinciden, la sustitución del |-VLDC $p[s]$ contiene un único nodo, que es $d[t]$. De forma que, de nuevo, se obtiene

$$distBosque(l(i)..s, l(j)..t) = distBosque(l(i)..s-1, l(j)..t-1) + \gamma(p[s] \to d[t])$$

(II) $u \neq t$

En este caso $p[s]$ es sustituido por, al menos, dos nodos que pertenecen a un camino comienza en $d[t]$. Se supondrá que el siguiente nodo en ese camino es el hijo de $d[t]$ en la posición t_k, esto es, el nodo $d[t_k]$. Así, la distancia que se desea calcular se puede formular en base a la distancia

[6] En el caso de que $d[t]$ sea una hoja, es decir, $t = l(t) = l(j)$, sólo serán aplicables las tres primeras expresiones.

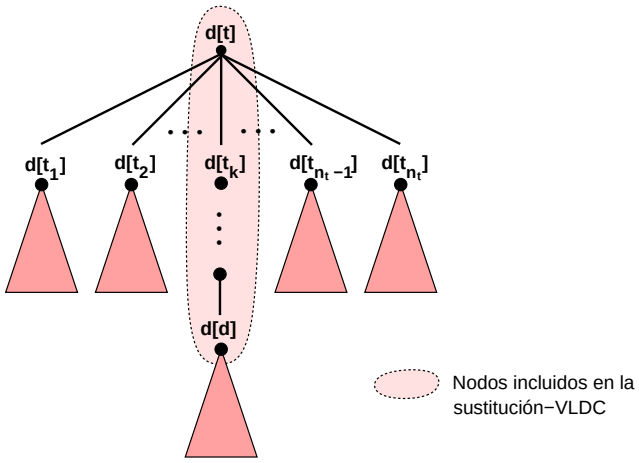


Figura 6.7: Caso 3(b) de la demostración del lema 6.4.

entre el subárbol $P[s]$ y el subárbol $D[t_k]$[7], junto con el coste correspondiente a la eliminación de todos los demás hijos de $d[t]$ y de sus descendientes. La razón de esas eliminaciones es que esos nodos no participan en la sustitución del |-VLDC $p[s]$. La correspondencia M se establecerá entre los subárboles $P[s]$ y $D[t_k]$, pero todos los demás descendientes de $d[t]$ al estar afectados por la correspondencia deben ser borrados. El resultado de este razonamiento es que la distancia viene dada por:

$$\begin{aligned} distBosque(l(i)..s, l(j)..t) = \quad & distArbol(s, t_k) + distArbol(\emptyset, t_1) + \ldots + distArbol(\emptyset, t_{k-1}) + \\ & + distArbol(\emptyset, t_{k+1}) + \ldots + distArbol(\emptyset, t_{n_t}) \end{aligned}$$

Aplicando la relación $distBosque(\emptyset, l(j)..t-1) = \sum_{l=1}^{n_t} distArbol(\emptyset, t_l)$, derivada del lema 6.1, la parte derecha de esta expresión se puede reescribir como:

$$distBosque(\emptyset, l(i)..t-1) + distArbol(s)t_k - distArbol(\emptyset, t_k)$$

Dado que $d[t]$ tiene n_t hijos, será necesario comprobar cual de ellos es el $d[t_k]$ que da lugar a la menor distancia. Por lo tanto, en este caso, para obtener la distancia distancia final se buscará el mínimo valor de entre los costes correspondientes a los n_t hijos de $d[t]$.

$$\begin{aligned} distBosque(l(i)..s, l(j)..t) = \quad & distBosque(\emptyset, l(j)..t-1) + \\ & \min_{t_k} \{distArbol(s)t_k - distArbol(\emptyset)t_k \mid 1 \leq k \leq n_t\} \end{aligned}$$

Con lo que obtenemos la cuarta expresión del lema.

Como las fórmulas anteriores cubren todos los casos posibles, basta tomar el menor de esos valores para obtener la distancia deseada.

□

6.2.3. Distancia con ∧-VLDC

Una vez estudiado el cálculo de distancias para el caso de subárboles patrón cuyas raíces sean |-VLDC, se describirá el cálculo de la distancia cuando el símbolo VLDC asociado a esas raíces sea un ∧-VLDC. El cálculo de las distancias donde esté involucrado un ∧-VLDC es mucho más complejo que en el caso de los

[7]Distancia que habría sido calculada en pasos anteriores del algoritmo.

|-VLDC. La diferencia entre ambos situaciones se debe a que en los cálculos en presencia de ∧-VLDC hay que tener en cuenta que el coste asociado a los subárboles que formen parte su sustitución-VLDC es nulo y por lo tanto no se tendrá en cuenta en la distancia final.

Como se vio anteriormente, en el cálculo de distancias con |-VLDC no se necesita tener en cuenta ese tipo de consideraciones, basta con conocer los valores de las distancias *distBosque* y *distArbol* de los descendientes del nodo |-VLDC. Sin embargo, la principal dificultad del cálculos para ∧-VLDC surge del hecho de que la distancia entre bosques *distBosque* no contempla la posibilidad de omitir un conjunto de subárboles, de forma que estos no se tengan en cuenta en el valor final de la distancia. Será necesario definir una nueva distancia en la que sea posible no tomar en consideración determinados subárboles. Para ello Zhang y Shasha[66] definen la *distancia entre sufijos* [8], abreviada *dbs*, que toma como base la distancia entre bosques, *distBosque*, añadiendo la posibilidad de que conjuntos de subárboles consecutivos puedan ser eliminados con coste cero. La definición formal es la siguiente.

Definición 6.7 (distancia entre bosques sufijos)**.**
Dados un árbol patrón P y un árbol dato D. Siendo B_P un bosque de P y B_D un bosque de D, se define la distancia entre sufijos *entre los bosques B_P y B_D, en forma abreviada $dbs(B_P, B_D)$, como la distancia entre los bosques B_P y B'_D, en donde B'_D es un subbosque de B_D en cual ha sido eliminado un conjunto de subárboles verificando las siguientes restricciones:*

- *Todos ellos son* subárboles completos[9]
- *Son consecutivos*
- *Están situados en el extremo más a la izquierda del bosque B_D*
- *Todos tienen el mismo nodo padre.*

resultando que $dbs(B_P, B_D) = min_{B'_D} distBosque(B_P, B'_D)$.

Intuitivamente, la distancia entre sufijos entre dos bosques B_P y B_D es la distancia que existe entre el bosque B_P y un fragmento del bosque B_D, en el cual se han eliminados sus k primeros árboles por la izquierda. En el siguiente ejemplo se muestra como la distancia *distBosque* por si sola no es suficiente para manejar distancias con ∧-VLDC e ilustra la noción de distancia entre sufijos.

Ejemplo 6.2.
En la figura 6.8, $P[s]$ y $D[t]$ son dos subárboles de P y de D, respectivamente. Se supondrá que la raíz $p[s]$ es un ∧-VLDC*, y que en la mejor sustitución-*VLDC *ese* ∧-VLDC*, $p[s]$, es sustituido por $d[t]$ y por los* k *subárboles hijos de $d[t]$ más a la izquierda. Por lo tanto, en este caso resulta que la distancia entre $P[l(i)..s]$ y $D[l(j)..t]$ se corresponde con la distancia entre $P[l(i)..s-1]$ y $D[l(i_{k+1}..t-1]$.*

Es evidente que la distancia $distBosque(l(i)..s-1, l(j)..t-1)$ no ofrece ese valor, ya que los bosques que se consideran en el cálculo de distBosque *comienzan todos en la hoja más a la izquierda de $D[l(j)..t]$,es decir, en $d[l(j)]$.*

Dado que los k subárboles sombreados en la figura 6.8 cumplen las condiciones de ser completos, consecutivos, situados a la izquierda y tener el mismo padre, se corresponden con el conjunto de subárboles que no son tenidos en cuenta al calcular $dbs(l(i)..s-1, l(j)..t-1)$. Por lo tanto, la distancia que deseamos se obtiene a partir de esa distancia entre sufijos:

$$distBosque(l(j)..s-1, l(t_{k+1})..t-1) = dbs(l(i)..s-1, l(j)..t-1)$$

Resultando que $distArbol(s,t) = dbs(l(i)..s-1, l(j)..t-1)$.

[8] En inglés, *suffix forest distance.*

[9] Un subárbol es completo con respecto a un bosque B, si ese subárbol no es, a su vez, subárbol de ningún otro subárbol contenido en B.

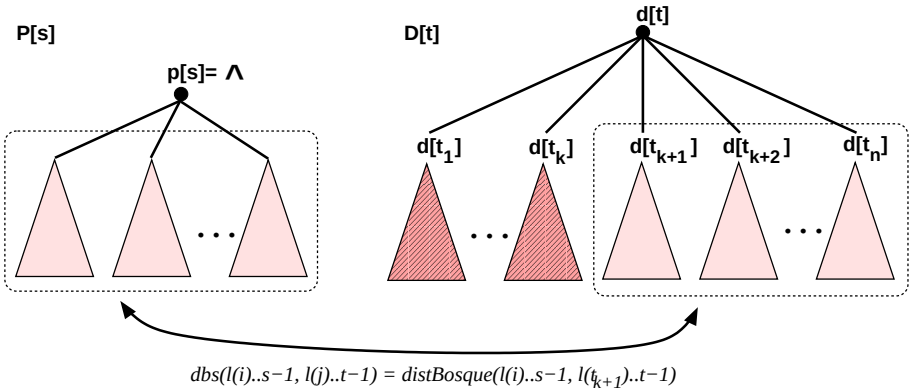


Figura 6.8: Diferencia entre *distBosque* y *dbs*.

La aproximación más simple para calcular *dbs* sería aplicar directamente la definición y probar todas y cada una de las posibles combinaciones de subárboles que se puedan eliminar, calcular las distancias *distBosque* resultantes y tomar la menor de todas ellas. Sin embargo, es posible plantear un esquema de cálculo análogo al utilizado para obtener las distancias *distBosque* con menor coste computacional y con la ventaja de poder calcular los valores de *distBosque* y *dbs* al mismo tiempo, efectuando un único recorrido sobre el árbol.

Cálculo de la distancia auxiliar entre sufijos

De forma similar a como ocurre en el cálculo de los valores *distBosque*, el cálculo de las distancias entre sufijos, *dbs*, se basa en los resultados de los siguientes lemas. El primero de ellos define como se realiza la inicialización de los valores *dbs* y los otros dos definen los cálculos a realizar para el caso de distancias entre árboles y entre bosques, respectivamente.

Lema 6.5.
Sean P y D árboles, y sean $i \in P$ y $j \in D$ dos de sus nodos. Siendo $p[s] \in desc(i)$ y $d[t] \in desc(j)$, entonces:

1. $dbs(\emptyset, \emptyset) = 0$.
2. $dbs(P[l(i)..s], \emptyset) = distBosque(P[l(i)..s], \emptyset)$.
3. $dbs(\emptyset, D[l(j)..t]) = \begin{cases} 0 & \textbf{sii } l(t) = l(j) \textbf{ ó } l(padre(t)) = l(j) \\ dbs(\emptyset, D[l(j)..t-1]) + \gamma(\varepsilon \rightarrow d[t]) & \textbf{en otro caso} \end{cases}$

Demostración.

- El caso *1* es trivial.
- El caso *2* se deriva del hecho de que no hay ningún árbol que eliminar en un bosque vacío y, por lo tanto, *dbs* es equivalente a *distBosque*.
- En el caso *3*, hay dos subcasos:
  (a) $l(t) = l(j)$
  En este caso, $D[l(j)..t]$ es un árbol y la opción de menor coste será eliminarlo completamente, resultando $dbs(\emptyset, D[l(j)..t]) = 0$.

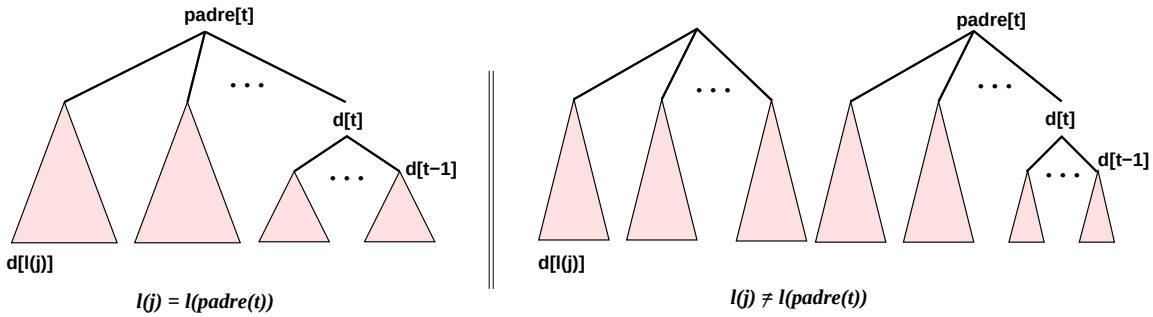


Figura 6.9: Topologías del bosque $D[l(j)..t]$.

(b) $\underline{l(t) \neq l(j)}$
Son posibles las dos situaciones que se ilustran en la figura 6.9:

(I) $\underline{l(padre(t)) = l(j)}$
Tal y como se muestra en el esquema izquierdo de la figura 6.9, todos los subárboles del bosque $D[l(j)..t]$ tienen el mismo padre, el nodo $D[padre(t)]$. Por lo tanto la opción de menor coste es eliminar todo el bosque, resultando

$$dbs(\emptyset, D[l(j)..t]) = 0$$

(II) $\underline{l(padre(t)) \neq l(j)}$
En este caso, mostrado en el esquema derecho de la misma figura, cualquier posible conjunto consecutivo de subárboles completos que se pueda construir en el bosque $D[l(j)..t-1]$ y que sea susceptible de ser eliminado, no puede incluir a ningún subárbol que sea hijo de $d[t]$. Si así fuera, de acuerdo con la definición de *dbs*, todos los nodos a eliminar deberían de ser hijos de $d[t]$, lo cual implicaría que $l(padre(t)) = l(j)$. Dado que no es así, el mejor conjunto de subárboles a eliminar del bosque $D[l(j)..t]$ coincide con el mejor conjunto de $D[l(j)..t-1]$. Por lo tanto,

$$dbs(\emptyset, D[l(j)..t]) = dbs(\emptyset, D[l(j)..t-1]) + \gamma(\varepsilon \rightarrow d[t])$$

□

El cálculo de *dbs* en el caso en el que el bosque dato sólo contiene un árbol, viene dado por el siguiente lema.

Lema 6.6.
Sean P y D árboles, sean $p[i]$ y $d[j]$ dos de sus nodos, y sean $p[s] \in desc(p[i])$ y $d[t] \in desc(d[j])$. Si $l(t) = l(j)$, entonces:

$$dbs(l(i)..s, l(j)..t) = \ min \left\{ \begin{array}{l} distBosque(l(i)..s, \emptyset), \\ distBosque(l(i)..s, l(j)..t) \end{array} \right.$$

Demostración.
Dado que $l(t) = l(j)$, $D[l(j)..t]$ es un árbol, pueden distinguirse dos situaciones:

- Caso 1: Es menos costoso eliminar $D[l(j)..t]$, resultando que:

$$dbs(l(i)..s, l(j)..t) = distBosque(l(i)..s, \emptyset)$$

- Caso 2: Es menos costoso transformar $D[l(i)..s]$ en $D[l(j)..t]$, resultando que:

$$dbs(l(i)..s, l(j)..t) = distBosque(l(i)..s, l(j)..t)$$

Por lo tanto, resulta que, para obtener la distancia deseada, bastará con tomar el menor de ambos valores. □

El valor de la distancia *dbs* para bosques dato con más de un árbol se calcula a partir de los resultados enunciados en el siguiente lema.

Lema 6.7.
Sean P y D árboles, sean $p[i]$ y $d[j]$ dos de sus nodos, y sean $p[s] \in desc(p[i])$ y $d[t] \in desc(d[j])$. Si $l(t) \neq l(j)$, entonces:

$$dbs(l(i)..s, l(j)..t) = \ min \begin{cases} dbs(l(i)..s-1, l(j)..t) + \gamma(p[s] \rightarrow \varepsilon), \\ dbs(l(i)..s, l(j)..t-1) + \gamma(\varepsilon \rightarrow d[t]), \\ dbs(l(i)..s-1, l(j)..t-1) + distArbol(s,t) \end{cases}$$

Demostración.
Consideraremos la situación en la que $p[s] \neq |$ y $p[s] \neq \wedge$. El caso en el que $p[s] = |$ y $p[s] = \wedge$ se demuestra de forma análoga al caso *2* de la demostración del lema 6.2.

Como en casos anteriores, partimos de que M es una correspondencia de coste mínimo. En este caso, M se establece entre el bosque $P[l(i)..s]$ y el resultado de eliminar el mejor subbosque B del bosque $D[l(j)..t]$. Ese subbosque B estará situado en el extremo izquierdo de $D[l(j)..t]$ y estará formado por un conjunto de árboles completos, consecutivos y con el mismo padre. Hay dos situaciones posibles, idénticas a las del caso *3(b)* consideradas en la demostración del lema anterior, ilustrados en la figura 6.9.

- Caso 1: $l(padre(t)) = l(j)$

  El bosque $D[l(j)..t]$ es como el mostrado en el esquema de la izquierda de la figura 6.9. Tenemos cuatro subcasos que considerar:

  (a) El subárbol $D[t]$ no es eliminado y el nodo $p[s]$ no está afectado por M.
  Siguiendo un razonamiento análogo al efectuado en la demostración del lema 6.2 se llega a la siguiente expresión:

$$dbs(l(i)..s, l(j)..t) = dbs(l(i)..s-1, l(j)..t) + \gamma(p[s] \rightarrow \varepsilon)$$

  (b) El subárbol $D[t]$ no es eliminado y el nodo $d[t]$ no está afectado por M.
  Para aplicar un razonamiento simétrico al empleado en el caso anterior es necesario demostrar un resultado previo.

  Proposición 6.1.
  En el caso de que $l(padre(t)) = l(j)$, el mejor subbosque B a eliminar del bosque $D[l(j)..t]$ es el mismo que en el bosque $D[l(j)..t]$. Siendo cierto, además, que la mejor correspondencia, M, entre $P[l(i)..s]$ y $D[l(j)..t] \setminus B$ es la misma que entre $P[l(i)..s]$ y $D[l(j)..t-1] \setminus B$.

  Demostración.
  Supongamos que no sea así. Entonces B_{t-1} será el mejor subbosque a eliminar en $D[l(j)..t-1]$ y M_{t-1} será la correspondencia asociada. Todos los árboles que se eliminan, los pertenecientes a B_{t-1}, tendrán a $padre(t)$ como padre, tal y como se deduce del esquema en el lado izquierdo de la figura 6.9. Además, B_{t-1} no podrá incluir a ningún subárbol hijo de $d[t]$. De ser este el caso, se podrían eliminar de $D[l(j)..t]$ los árboles de B_{t-1} y utilizar la correspondencia

M_{t-1} entre $P[l(i)..s]$ y $D[l(j)..t-1] \setminus B_{t-1}$. El coste resultante sería menor que el de la correspondencia M entre $P[l(i)..s]$ y$D[l(j)..t] \setminus B$, contradiciendo la suposición inicial de que B era el mejor conjunto de bosques a eliminar; y M una correspondencia de coste mínimo. Resultando, entonces, que la proposición es correcta. □

Utilizando la proposición anterior y tomando como base la demostración del lema 6.2 se llega a la expresión:

$$dbs(l(i)..s, l(j)..t) = dbs(l(i)..s, l(j)..t-1) + \gamma(\varepsilon \rightarrow d[t])$$

(c) El subárbol $D[t]$ no es eliminado y los nodos $p[s]$ y $d[t]$ están afectados por M.
De nuevo, se sigue el esquema de la demostración del lema 6.2 para llegar a,

$$dbs(l(i)..s, l(j)..t) = dbs(l(i)..l(s)-1, l(j)..l(t)-1) + distArbol(s,t)$$

(d) El bosque $D[l(j)..t]$ es eliminado completamente.
En este caso, $dbs(l(i)..s, l(j)..t) = distBosque(l(s)..i, \emptyset)$. Aplicando el lema 6.1 se deduce la igualdad,

$$dbs(l(s)..i, l(j)..t) = distBosque(l(i)..s-1, \emptyset) + \gamma p[s] \rightarrow \varepsilon$$

Como por la definición de *dbs* se cumple que $dbs(l(i)..s-1, l(j)..t) \leq distBosque(l(i)..s-1, \emptyset)$, este subcaso estará cubierto por el subcaso *1(a)*.

- Caso 2: $l(padre(t)) \neq l(j)$

El bosque $D[l(j)..t]$ es como el mostrado en el esquema a la derecha de la figura 6.9. En este caso tenemos sólo tres subcasos, puesto que, como se deduce de la figura no es posible eliminar el bosque completo y no se puede dar la situación del subcaso (d) anterior.

(a) El nodo $p[s]$ no está afectado por M.
Al igual que en el subcaso *1(a)* llegamos a,

$$dbs(l(i)..s, l(j)..t) = dbs(l(i)..s-1, l(j)..t) + \gamma(p[s] \rightarrow \varepsilon)$$

(b) El nodo $d[t]$ no está afectado por M.
Del esquema de la derecha de la figura 6.9 se llega a que el subárbol $D[t]$ no puede ser uno de los árboles a eliminar incluidos en el subbosque B. Del mismo modo, tampoco pueden ser eliminados los demás hijos de $d[padre(t)]$, ya que de ser así los de B no cumplirían la condición de tener todos el mismo padre. Por lo tanto se llega a una conclusión análoga a la de la proposición anterior, con lo que puede deducir la siguiente expresión:

$$dbs(l(i)..s, l(j)..t) = dbs(l(i)..s, l(j)..t-1) + \gamma(\varepsilon \rightarrow d[t])$$

(c) Los nodos $p[s]$ y $d[t]$ están afectados por M.
Al igual que en el subcaso *1(c)* resulta la expresión

$$dbs(l(s)..i, l(j)..t) = distBosque(l(i)..s-1, \emptyset) + \gamma p[s] \rightarrow \varepsilon$$

Combinando los subcasos anteriores y teniendo en cuenta que el subcaso *1(d)* está incluido en el *1(a)* se obtiene la expresión mostrada en el lema. □

Tal como hemos visto en los dos lemas anteriores, el cálculo de la distancia auxiliar *dbs* es similar al de la distancia *distBosque*. En ambos casos se diferencian dos tipos de cálculos dependiendo de la forma de los árboles, resultando un conjunto de fórmulas para distancias entre bosques y otro para distancias entre árboles.

Distancia en patrones con ∧-VLDC

En el apartado anterior, introducíamos la necesidad de utilizar la distancia auxiliar *dbs* para manejar las sustituciones-VLDC que impliquen símbolos ∧-VLDC y estudiamos como se realiza el cálculo de esas distancias. Veremos ahora como se integran las dos distancias, *distBosque* y *dbs*, para efectuar el cálculo de la distancia de edición entre subárboles para los casos en los que la raíz del subárbol patrón está etiquetada con un ∧-VLDC. El siguiente lema indica como calcular esa distancia.

Lema 6.8.
Sean P y D árboles, sean $p[i]$ y $d[j]$ dos de sus nodos, y sean $p[s] \in des(i)$ y $d[t] \in des(j)$. Suponiendo que $l(s) = l(i)$ y $l(t) = l(j)$, y que $p[s] = \wedge$, entonces:

$$distBosque(l(i)..s, l(j)..t) = \ min \begin{cases} distBosque(l(i)..s-1, l(j)..t) + \gamma(p[s] \to \varepsilon), \\ distBosque(l(i)..s, l(j)..t-1) + \gamma(\varepsilon \to d[t]), \\ distBosque(l(i)..s-1, l(j)..t-1) + \gamma(p[s] \to d[t]), \\ min_{t_k}\{dbs(l(i)..s-1, l(j)..t_k) | 1 \le k \le n_t\}, \\ min_{t_k}\{distArbol(s, t_k) | 1 \le k \le n_t\} \end{cases}$$

donde los nodos $d[t_k]$, con $1 \le k \le n_t$, son los n_t hijos del nodo $d[t]$[10].

Demostración.
Comenzaremos examinando el caso trivial, cuando $s = l(i)$, es decir $p[s] = \wedge$ es el único nodo de $P[l(s)..s]$. En este caso la mejor sustitución-VLDC es reemplazar la hoja $p[s]$ por todo el árbol $D[l(t)..t]$, resultando en un coste de 0. Dicho valor se obtendría de la expresión $min_{t_k}\{distArbol(s, t_k) | 1 \le k \le n_t\}$, puesto que cualquier $D[t_k]$ sería un subárbol hijo de $d[t]$ cuya distancia también sería 0.

Consideraremos entonces, el caso en el que $p[s]$ no sea una hoja, es decir $l(s) \neq s$. Como hemos venido haciendo hasta el momento, M será la correspondencia de coste mínimo entre el bosque $P[l(i)..s]$ y el bosque $D[l(j)..t]$, teniendo en cuenta las mejores sustituciones para los símbolos VLDC que pudieran estar incluidos en $P[l(i)..s]$. Existen tres posibilidades:

1. $p[s]$ es reemplazado por un conjunto vacío de nodos .
2. $p[s]$ es reemplazado por un conjunto no vacío de nodos y el nodo $d[t]$ no está afectado por M.
3. $p[s]$ es reemplazado por un conjunto no vacío de nodos y el nodo $d[t]$ está afectado por M.

En los casos *1* y *2*, el razonamiento es el mismo que el empleado en la demostración del lema 6.2 y las fórmulas obtenidas son idénticas. Para el caso *3* hay dos situaciones posibles:

(a) El nodo $d[t]$ es una hoja, es decir $l(t) = t$

En este caso, en $D[l(i)..t]$ sólo hay un nodo, por lo que el ∧-VLDC $p[s]$ es reemplazado por $d[t]$, puesto que $\gamma(\wedge \to d[i]) = 0$, se puede utilizar la siguiente expresión:

$$distBosque(l(i)..l(j)..t, =) distBosque(l(i)..s-1, l(j)..t-1) + \gamma(p[s] \to d[t])$$

(b) El nodo $d[t]$ tiene hijos, es decir $l(t) \neq t$

Puesto que el árbol $D[l(i)..t]$ contiene más de un nodo, $p[s]$ debe ser reemplazado un conjunto de nodos de acuerdo a la definición de ∧-VLDC. Además, el bosque $P[l(i)..s-1]$ debe ser transformado en subbosque incluido en $D[l(i)..t-1]$ y formado por un conjunto de árboles consecutivos hijos todos ellos de un nodo, $d[d]$, del árbol $D[l(t)..t]$. Supondremos que $d[t]$ tiene n_t hijos, $d[t_1], d[t_2], \ldots, d[t_{n_t}]$. Tenemos dos posibilidades a la hora de ubicar ese nodo $d[d]$ en el árbol:

[10] En el caso de que $d[t]$ sea una hoja, es decir, $t = l(t) = l(j)$, sólo serán aplicables las tres primeras expresiones.

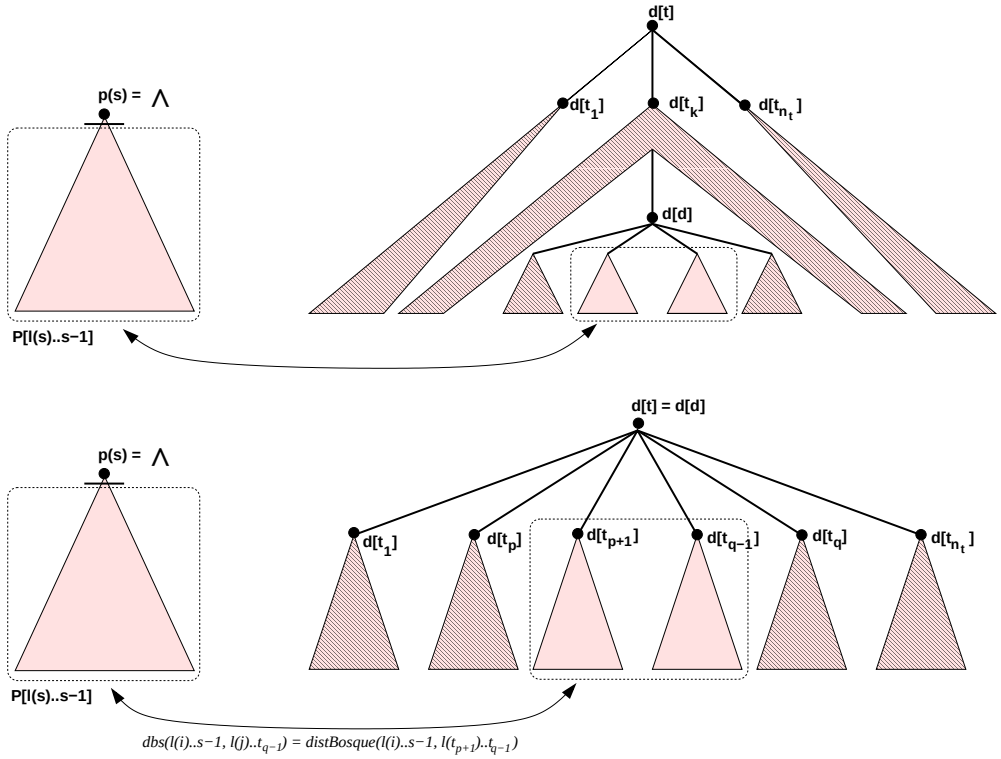


Figura 6.10: Subcasos *(i)* y *(ii)* del lema 6.8.

(I) $\underline{d = t}$

En este caso el nodo $d[t]$ será el padre de los nodos que se emparejarán con el bosque $P[l(i)..s-1]$, tal y como se ilustra en el esquema superior de la figura 6.10, donde los nodos que reemplazan al ∧-VLDC están señalados en color más oscuro. De este modo, el nodo $p[s]$ será reemplazado por el nodo $d[t]$ y por los subárboles $D[t_1], \ldots, D[t_l]$ y $D[t_k], \ldots, D[t_{n_t}]$, con $1 \leq p, \leq k \leq n_t$. Los árboles $D[t_1], \ldots, D[t_l]$ forman un bosque que cumple las condiciones exigidas en la definición de la distancia *dbs*. Por lo tanto, tenemos que:

$$distBosque(l(i)..s, l(i)..t,) = distBosque(l(i)..s-1, l(t_{k+1})..t_{k-1}) = dbs(l(i)..s-1, l(j)..t_{k-1})$$

Para encontrar la mejor sustitución deberemos tomar al hijo $d[q]$, con $1 \leq q \leq t_n$, que minimice la expresión anterior, resultando que:

$$distBosque(l(i)..s, l(i)..t,) = min_{t_k}\{dbs(l(i)..s-1, l(j)..t_k) | 1 \leq k \leq n_t\}$$

La fórmula anterior también cubre el caso de que $p[s]$ sea sustituido por la totalidad del árbol $D[t]$. En ese caso tendríamos que

$$distBosque(l(i)..s, l(j)..t, =) distBosque(l(i)..s-1, \emptyset) \geq dbs(l(i)..s, l(j)..t_k)\}$$

para cualquier t_k entre 1 y n_t. Por lo el mínimo obtenido en la expresión anterior continuaría siendo válido.

(II) $\underline{d \neq t}$

Este caso se muestra en la parte inferior de la figura 6.10, donde la porción del árbol que reemplaza al nodo $p[s]$ se señala en tono más oscuro. En este caso, el nodo $d[d]$ no es uno

de los hijos de $d[t]$, sino que está en niveles inferiores. A partir de la figura se deduce que $distBosque(l(i)..s, l(j)..t) = distArbol(s, d)$. Además, el cálculo de dicho valor, $distArbol(s, d)$, se habría realizado en una situación como la descrita en el subcaso anterior.

Supondremos que el nodo $d[t_k]$ es el hijo de $d[t]$ en el camino entre $d[t]$ hasta $d[d]$. Dado que todos los nodos desde $d[t_k]$ hasta $d[d]$, junto con sus subárboles asociados, forman parte de la sustitución y no afectan a la distancia, tenemos que:

$$distArbol(s, t_k) = distArbol(s, d)$$

De nuevo, para encontrar el valor deseado se toma el mínimo, con k entre 1 y n_t, resultando:

$$distBosque(l(i)..s, l(i)..t) = min_{t_k}\{distArbol(s, t_k) | 1 \leq k \leq n_t\}$$

Como en ocasiones anteriores, puesto que estos subcasos cubren todas las posibles formas de obtener la distancia deseada, bastará con elegir el de menor coste, quedando demostrado el lema. □

6.2.4. Algoritmo para el cálculo de las distancias

Hemos presentado en las secciones precedentes los resultados teóricos que formalizan el cálculo de la distancia de edición en presencia de símbolos VLDC y se ha demostrado su corrección. Para concluir con el estudio de este tipo de reconocimiento aproximado de patrones en árboles, el algoritmo 6.1 muestra como se integran estos lemas y teoremas para calcular la distancia de edición entre un árbol patrón con símbolos VLDC P y un árbol dato D. Para simplificar el algoritmo y facilitar su comprensión se presenta únicamente la estructura del proceso de cálculo de las distancias y se emplean funciones auxiliares que ocultan los detalles del calculo de las distancias en los distintos casos que se pueden presentar. Las fórmulas empleadas en cada uno de esos casos particulares se muestran en las funciones el algoritmo 6.2 y, como se puede apreciar, se corresponden con los resultados de los lemas y teoremas presentados en las secciones anteriores.

El esquema que se sigue en este algoritmo es idéntico al empleado en el algoritmo 5.1 para el cálculo de distancias sin VLDC mostrado en el capítulo 5. Como en ese algoritmo, se recorren los conjuntos de *raíces_I* de los dos árboles. Para cada par de nodos $i \in raíz\_I(P)$ y $j \in raíz\_I(D)$ se calcula la distancia $distArbol(i, j)$, que a su vez supone calcular de modo ascendente las distancias entre todos los pares de subbosques extraídos de esos dos subárboles. En el caso del algoritmo 6.1 hay dos diferencias notables. En primer lugar, se distinguen cuatro tipos de cálculos distintos y no dos como en el algoritmo sin VLDC. Se utilizan fórmulas específicas para distancias entre bosques, distancias entre árboles con $\wedge$-VLDC, distancias entre árboles con $|$-VLDC y para distancias entre árboles sin símbolos VLDC.

La segunda diferencia con la que nos encontramos, es que en este algoritmo es necesario utilizar una segunda estructura de almacenamiento temporal. Se trata de una nueva matriz temporal cuya finalidad es mantener los valores de las distancias *dbs* necesarias en el cálculo de distancias que impliquen un $\wedge$-VLDC. Dicha matriz temporal se maneja de forma análoga a la matriz de distancias *distBosque*. De este modo, se añaden las fórmulas necesarias para inicializarla al comienzo del procesamiento de cada par de *raíces_I*. Además, en cada iteración, se efectúa el cálculo de la distancia *dbs* inmediatamente después de haber calculado, empleando uno de los cuatro conjuntos de fórmulas anteriores, la distancia *distBosque* correspondiente. La razón de hacerlo en ese orden es que en el cálculo de la distancia *dbs* se utilizan los valores de *distBosque*, de modo que es necesario asegurar que dichos valores estén disponibles en el momento necesario. Para el cálculo de los valores *dbs* se utilizan dos conjuntos de fórmulas, uno para el calculo de distancias *dbs* que afecten a un árbol y otro para distancias *dbs* que afecten a bosques.

Algoritmo 6.1 Algoritmo de cálculo de distancias con VLDC

Entrada: árbol patrón P y árbol dato D
$raíz\_I(P)$ y $raíz\_I(D)$ ordenados de menor a mayor
Salida: distancias $distArbol(s,t)$ para todos los subárboles $p[s] \in P$ y $d[t] \in D$

```
/* recorrer conjunto de raíces_I */

for i = 1 to |raíz_I(P)| do
  for j = 1 to |raíz_I(D)| do

    /*cálculo de distArbol(i,j)*/

    inicialización_distBosque()
    inicialización_dbs()

    for s = l(i) to i do
      for t = l(j) to j do
        if (l(s) = l(i)) and (l(t) = l(j)) then

          /* distancia árbol-árbol */

          if (p[s] = ∧)
            distancia_árbol_∧-VLDC(l(i), s, l(j), t)
          else if (p[s] =|)
            distancia_árbol_|-VLDC(l(i), s, l(j), t)
          else
            distancia_árbol(l(i), s, l(j), t)
          end if
          distArbol(s,t) = distBosque(l(i)..s, l(j)..t) /* guardar valor intermedio */
        else
          distancia_bosque(l(i), s, l(j), t)
        end if
        /* cálculo distancia auxiliar entre sufijos */

        if (l(t) = l(j)) then
          distancia_dbs_árbol(l(i), s, l(j), t)
        else
          distancia_dbs_bosque(l(i), s, l(j), t)
        end if
      end for
    end for
  end for
end for
```

Algoritmo 6.2 Funciones auxiliares

function *inicialización_distBosque*() `/* Lema 6.1 */`

$distBosque(\emptyset, \emptyset) = 0$

$distBosque(l(i)..s, \emptyset)) = distBosque(l(i)..s-1, \emptyset) + \gamma(p[s] \rightarrow \varepsilon)$, $\forall s$ con $l(i) \leq s \leq i$

$distBosque(\emptyset, l(j)..t) = distBosque(\emptyset, l(j)..t-1) + \gamma(\varepsilon \rightarrow d[t])$, $\forall t$ con $l(j) \leq t \leq j$

function *inicialización_dbs*() `/* Lema 6.5 */`

$dbs(\emptyset, \emptyset) = 0$

$dbs(l(i)..s, \emptyset)) = distBosque(l(i)..s, \emptyset)$, $\forall s$ con $l(i) \leq s \leq i$

$$dbs(\emptyset, l(j)..t) = \begin{cases} 0 \ \textbf{si} \ l(t) = l(j) \ \textbf{or} \ l(padre(t)) = l(j) \\ distBosque(\emptyset, l(j)..t-1) + \gamma(\varepsilon \rightarrow d[t]) \ \textbf{en otro caso} \end{cases}, \forall t \text{ con } l(j) \leq t \leq j$$

function *distancia_árbol*$(l(i), s, l(j), t)$ `/* Lema 6.3 */`

$$distBosque(l(i)..s, l(j)..t) = \min \begin{cases} distBosque(l(i)..s-1, l(j)..t) \ + \ \gamma(p[s] \rightarrow \varepsilon), \\ distBosque(l(i)..s, l(j)..t-1) \ + \ \gamma(\varepsilon \rightarrow d[t]), \\ distBosque(l(i)..s-1, l(j)..t-1) \ + \ \gamma(p[s] \rightarrow d[t])\} \end{cases}$$

function *distancia_árbol_*$\wedge$-VLDC$(l(i), s, l(j), t)$ `/* Lema 6.4 */`

$$distBosque(l(i)..s, l(j)..t) = min \begin{cases} distBosque(l(i)..s-1, l(j)..t) + \gamma(p[s] \rightarrow \varepsilon), \\ distBosque(l(i)..s, l(j)..t-1) + \gamma(\varepsilon \rightarrow d[t]), \\ distBosque(l(i)..s-1, l(j)..t-1) + \gamma(p[s] \rightarrow d[t]), \\ min_{t_k}\{dbs(l(i)..s-1, l(j)..t_k) | 1 \leq k \leq n_t\}, \\ min_{t_k}\{distArbol(s, t_k) | 1 \leq k \leq n_t\} \end{cases}$$

function *distancia_árbol_*|-VLDC$(l(i), s, l(j), t)$ `/* Lema 6.8 */`

$$distBosque(l(i)..s, l(j)..t) = min \begin{cases} distBosque(l(i)..s-1, l(j)..t) + \gamma(p[s] \rightarrow \varepsilon), \\ distBosque(l(i)..s, l(j)..t-1) + \gamma(\varepsilon \rightarrow d[t]), \\ distBosque(l(i)..s-1, l(j)..t-1) + \gamma(p[s] \rightarrow d[t]), \\ distBosque(\emptyset, l(j)..t-1) + \\ \quad \min_{t_k}\{distArbol(s, t_k) - distArbol(\emptyset, t_k) \mid 1 \leq k \leq n_t\} \end{cases}$$

function *distancia_bosque*$(l(i), s, l(j), t)$ `/* Lema 6.2 */`

$$distBosque(l(i)..s, l(j), t) = \min \begin{cases} distBosque(l(i)..s-1, l(j)..t) \ + \ \gamma(p[s] \rightarrow \varepsilon), \\ distBosque(l(i)..s], l(j)..t-1) \ + \ \gamma(\varepsilon \rightarrow d[t]), \\ distBosque(l(i)..l(s)-1], l(j)..l(t)-1) \ + \ distArbol(s, t)\} \end{cases}$$

function *distancia_dbs_árbol*$(l(i), s, l(j), t)$ `/* Lema 6.6 */`

$$dbs(l(i)..s, l(j)..t) = min \begin{cases} distBosque(l(i)..s, \emptyset), \\ distBosque(l(i)..s, l(j)..t) \end{cases}$$

function *distancia_dbs_bosque*$(l(i), s, l(j), t)$ `/* Lema 6.7 */`

$$dbs(l(i)..s, l(j)..t) = min \begin{cases} dbs(l(i)..s-1, l(j)..t) + \gamma(p[s] \rightarrow \varepsilon), \\ dbs(l(i)..s, l(j)..t-1) + \gamma(\varepsilon \rightarrow d[t]), \\ dbs(l(i)..s-1, l(j)..t-1) + distArbol(s, t) \end{cases}$$

Parte III

Reconocimiento de Patrones en Bosques Compartidos

Capítulo 7

Bosques de análisis compartidos

Antes de abordar la integración de algoritmos de reconocimiento de patrones en árboles y las técnicas de análisis sintáctico, es conveniente revisar el problema del análisis sintáctico para conocer como son las estructuras sintácticas con las que alimentaremos esos algoritmos.

El presente capítulo comienza presentando una serie de generalidades sobre el problema del análisis sintáctico y una descripción de las técnicas de análisis sintáctico existentes, prestando especial atención a la utilización de los *autómata con pila*(APs). A continuación, se presentará el sistema generador de analizadores sintácticos ICE [56], basado en la interpretación tabular de APs, y con el que se ha realizado nuestro trabajo sobre integración de las técnicas de comparación de árboles. Se revisarán los formalismos de representación de estructuras sintácticas, centrándonos en el uso de gramáticas formales para representar análisis ambiguos. Terminaremos explicando como se lleva a cabo la generación de estas representaciones en ICE.

7.1. Análisis sintáctico

El *problema del análisis sintáctico* consiste en diseñar un algoritmo tal que dadas una gramática y una sentencia de entrada cualesquiera, compruebe si dicha sentencia pertenece al lenguaje generado por la gramática y, en caso afirmativo, genere una representación apropiada de la estructura sintáctica de la sentencia de entrada. Estos algoritmos se conocen como *analizadores sintácticos*. El análisis sintáctico incluye el *problema del reconocimiento* que hace referencia al desarrollo de *reconocedores sintácticos*. En el caso de los reconocedores, éstos se limitan a determinar si la sentencia de entrada pertenece o no al lenguaje generado por la gramática, sin generar estructuras sintácticas de salida.

La representación generada por los analizadores sintácticos indicará la secuencia de reglas que deben ser aplicadas para construir las derivaciones de la sentencia de entrada a partir del símbolo inicial de la gramática. La forma habitual de realizar estás representaciones supone la construcción de un árbol durante el proceso de análisis de la sentencia donde se reflejan esas derivaciones.

7.1.1. Técnicas de análisis

En esencia, tanto el problema del análisis como el del reconocimiento, pueden plantearse como problemas de búsqueda. El objetivo de esos procesos de búsqueda será encontrar la secuencia de derivaciones apropiada para una sentencia de entrada, en función de la gramática. Desde este punto de vista, las diferentes alternativas disponibles a la hora de definir el funcionamiento de los algoritmos de análisis sintáctico determinarán la estrategia de la búsqueda a realizar. Atendiendo a este criterio [1, 12] se puede plantear una primera clasificación de los algoritmos de análisis sintáctico:

- Los *analizadores sintácticos descendentes* inician la búsqueda en el axioma de la gramática y determinan cuales son reglas a usar en las derivaciones para llegar hasta los no terminales que componen la sentencia de entrada.

- Los *analizadores sintácticos ascendentes* utilizan la estrategia opuesta, partiendo de los símbolos terminales de la sentencia, deciden que reglas se deben utilizar para llegar al axioma de la gramática.
- Los *analizadores sintácticos mixtos* combinan aspectos de ambos enfoques. Por lo general resultan ser los más potentes y son los más utilizados en la práctica.

En muchos casos el proceso de análisis o reconocimiento no será determinista y existirán múltiples alternativas para continuar el proceso de búsqueda. Una de las causas más importantes de ese no determinismo es la ambigüedad de las gramáticas y lenguajes. Dicha ambigüedad implica que pueden existir diferentes análisis para una misma cadena de entrada, lo cual se traducirá en diferentes árboles de análisis. Dependiendo de la finalidad que se vaya a dar al análisis sintáctico efectuado, será necesario encontrar todos esos posibles análisis y representarlos de forma adecuada para su posterior procesamiento. Esto último es especialmente importante en los sistemas que procesen lenguajes naturales, dada la ambigüedad presente en las sentencia de este tipo de lenguajes. Atendiendo a las estrategias utilizadas para tratar el no determinismo puede hacerse otra clasificación de los algoritmos de análisis y reconocimiento sintáctico:

- *Basados en retroceso.*

  En este caso, el no determinismo se simula mediante un mecanismo de retroceso [1, 25]. Una vez llegados a un punto en que es necesario explorar diversas alternativas, se escoge únicamente una de ellas para continuar el análisis. Si en algún momento no es posible llegar a la solución, se retrocede hasta el último punto de no determinismo y se escoge una alternativa diferente, continuando el proceso y desechando las alternativas exploradas sin éxito.

  Aunque este enfoque es sencillo, presenta varios problemas importantes:

  - *Repetición de cálculos.* Los cálculos realizados tras la elección de una alternativa que resulte infructuosa se desechan. Como consecuencia, en caso de que esos mismos cálculos sean necesarios en alternativas futuras, se han de calcular de nuevo.
  - *Realización de cálculos innecesarios.* La elección de alternativas incorrectas provoca la realización de cálculos que no contribuyen a la solución final.
  - *Dificultad para localizar todas las soluciones existentes.* Cuando existen varios análisis diferentes debidos a la ambigüedad de la gramática suele ser necesario recuperarlos todos. En este caso se hace necesario forzar un retroceso para cada solución encontrada con lo que se agravan los problemas anteriores.

- *Basados en programación dinámica.*

  Mediante técnicas de programación dinámica [5, 59, 60, 10, 64, 17] se almacenan los cálculos intermedios de manera que no sea necesario repetirlos en caso de que sean necesarios de nuevo. Esta característica permite compartir cálculos entre las diferentes alternativas de análisis derivadas del no determinismo, solucionando los problemas derivados del uso de técnicas de retroceso. Esta será la aproximación seguida por el analizador utilizado en nuestro trabajo.

Una última clasificación para los algoritmos de análisis, se puede establecer en función de la dependencia que estos tengan respecto de la estructura gramatical durante el proceso de análisis sintáctico. Tenemos dos tipos de aproximaciones:

- *Guiados por la gramática.*

  La elección de alternativas se realiza únicamente examinando la información contenida en las reglas de la gramática.

- *Guiados por control finito.*

  Estos algoritmo incluyen una fase de preprocesamiento previa. Durante esa fase se construye un mecanismo de control a partir de las reglas gramaticales. Este mecanismo será usado durante el análisis para guiar la elección de alternativas, evitando la exploración de alternativas infructuosas.

7.1.2. Autómatas con pila

Nos centraremos ahora en el estudio de los APs, y su aplicación en la construcción de reconocedores y analizadores sintácticos para GICS. Estos autómatas son construcciones matemáticas que permiten realizar el reconocimiento de los lenguajes independientes del contexto. Tienen, además, la ventaja de que ofrecen un formalismo simple y uniforme dentro del cual se pueden enmarcar y evaluar distintas estrategias de búsqueda en el análisis sintáctico. Bastará, simplemente, con modificar la forma en que se definan los componentes del AP, en función de la estrategia de análisis deseada.

Definición 7.1 (autómata con pila)**.**
Un autómata com pila *es una 7-tupla* $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, \$_0, Q_F)$*, donde:*

- Q *es un conjunto finito de* estados*.*
- Σ *es un conjunto finito de* símbolos de entrada*.*
- Γ *es un conjunto finito de* símbolos de pila*.*
- $q_0 \in Q$ *es el* estado inicial*.*
- $\$_0 \in \Gamma$ *es el* símbolo inicial de la pila*.*
- $Q_F \subseteq Q$ *es el conjunto de* estados finales*.*
- δ *es un el conjunto de las* transiciones *del autómata:*

$$\begin{array}{rccc} \delta: & Q \times \Gamma \times (\Sigma \cup \{\varepsilon\}) & \longrightarrow & Q \times \Gamma^* \\ & (q, Z, a) & \rightsquigarrow & (q', \beta) \end{array}$$

Intuitivamente, un AP conforma un mecanismo que permite la definición de un lenguaje a través del reconocimiento de las cadenas que forman parte de él. Las transiciones representan los pasos intermedios que es posible realizar durante ese reconocimiento. Así, si el autómata se encuentra en el estado q, el siguiente símbolo de la cadena de entrada es a y el símbolo en la cima de la pila es Z, entonces, aplicando la transición $\delta(q, Z, a) = (q', \beta)$, el autómata pasa al estado q' y sustituye la cima de la pila por β. A partir de esta nueva situación del autómata, podrán tener lugar nuevas transiciones hasta que se llegue al reconocimiento de la sentencia de entrada o hasta que se determine que se trata de una sentencia no analizable.

Para formalizar este proceso de reconocimiento se define el concepto de *configuración*. Una configuración de un AP representa uno de los posibles estados intermedios durante el proceso de análisis. Por lo tanto, existirá una *configuración inicial* que represente el comienzo del reconocimiento y una o más *configuraciones finales* que indiquen la finalización exitosa del proceso de reconocimiento.

Definición 7.2 (configuración)**.**
Dados un AP $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, \$_0, Q_F)$ *y una cadena de entrada* w*, una* configuración *del autómata es una tupla* (q, α, x)*, donde* q *es el estado en que se encuentra o* estado actual*,* α *es el contenido de la pila, y* x *es la parte de la cadena de entrada que resta por analizar.*
La configuración inicial *se define como* $(q_0, \$_0, w)$*. Representando que el autómata se encuentra en su estado inicial, la pila está vacía y ninguna porción de la cadena ha sido reconocida.*
La aplicación de una transición $(q', \beta) \in \delta(q, Z, a)$ *a una configuración* (q, α, ax) *produce una nueva configuración* $(q', \alpha\beta, x)$*, que se notará como:*

$$(q, \alpha, ax) \vdash (q', \alpha\beta, x)$$

Por extensión, se notará el cierre transitivo de $\vdash$ *como* $\overset{+}{\vdash}$*, y el cierre reflexivo y transitivo como* $\overset{*}{\vdash}$*.*

A la hora de determinar el final del análisis, esto es, las configuraciones finales, hay dos alternativas. En el primer caso, en la configuración final, el autómata ha alcanzado uno de sus estados finales y se ha reconocido la totalidad de la cadena de entrada. En el segundo, la configuración final será aquella en que la pila se haya vaciado, habiendo consumido por completo la cadena de entrada. Estos dos casos dan lugar a dos definiciones del *lenguaje aceptado por el autómata*.

Definición 7.3 (lenguaje aceptado por un autómata)**.**
Dado un AP $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, \$_0, Q_F)$*, el* lenguaje aceptado por estado final *es el conjunto de cadenas:*

$$\mathcal{L}(\mathcal{A}) = \left\{ w \in \Sigma^* \mid (q_0, \$_0, w) \overset{*}{\vdash} (p, \alpha, \epsilon),\ p \in Q_F,\ \alpha \in \Gamma^* \right\}$$

Respectivamente, el lenguaje aceptado por pila vacía *es el conjunto de cadenas:*

$$\mathcal{L}(\overline{\mathcal{A}}) = \left\{ w \in \Sigma^* \mid (q_0, \$_0, w) \overset{*}{\vdash} (p, \epsilon, \epsilon),\ \forall p \in Q \right\}$$

Como se demuestra en [12] y en [14], las definiciones de lenguaje aceptado por el autómata dadas son equivalentes, puesto que dado un AP que reconoce un lenguaje por estado final es inmediato construir otro AP que reconozca el mismo lenguaje por pila vacía, y viceversa.

De cara a facilitar su utilización en ICE [56] es necesario redefinir el concepto de AP manejado hasta el momento. Tal y como se muestra en [29] el conjunto de estados del autómata es prescindible, ya que el estado está implícito en la secuencia de símbolos almacenados en la pila[1]. Asimismo, se puede restringir el tipo de transiciones, permitiendo únicamente transiciones canónicas que operen sobre parte superior de la pila y no sobre su totalidad. En concreto, en [14] se demuestra que para definir cualquier AP basta con utilizar transiciones que tomen en consideración, como máximo, los dos símbolos superiores de la pila. Así, se puede modificar la definición clásica e introducir una definición canónica de AP que no incluya el conjunto de estados y que restringa el conjunto de transiciones. Este será el tipo de AP utilizado en los analizadores construidos por ICE, sobre los que hemos basado nuestro trabajo. La gran ventaja de este tipo de APs es que, con estas modificaciones, se obtiene una representación más simple y uniforme de las configuraciones del AP lo que posibilita una procesamiento más eficiente.

Definición 7.4 (autómata con pila canónico)**.**
Un autómata con pila canónico *es una tupla* $\mathcal{A} = (\Sigma, \Gamma, \delta, \$_0, \$_f)$ *, donde:*

- Σ *es un conjunto finito de símbolos de entrada.*
- Γ *es un conjunto finito de símbolos de pila.*
- $\$_0$ *es el símbolo inicial de la pila.*
- $\$_f$ *es el* símbolo final de la pila.
- δ *es un conjunto de* transiciones

$$\begin{array}{rccc} \delta: & \Gamma^* \times (\Sigma \cup \{\varepsilon\}) & \longrightarrow & \Gamma^* \\ & (\alpha, a) & \rightsquigarrow & (\beta) \end{array}$$

Las transiciones podrán ser de tres tipos: HORIZONTAL*,* APILAR*, y* ELIMINAR*:*

- HORIZONTAL*:* $\delta(C, a) = (F)$*. Reemplaza el elemento* C *de la cima de la pila por* F*, y se lee* a *en la cadena de entrada.*

[1] Simplemente, bastaría con almacenar el estado como un elemento de la pila más.

- APILAR: $\delta(C, a) = (CF)$. *Apila un nuevo elemento F en la cima de la pila y se lee a de la cadena de entrada.*
- ELIMINAR: $\delta(CF, a) = (G)$. *Se eliminan los elementos C y F de la cima de la pila, se apila G, y se lee a.*

Al igual que en los autómatas con pila clásicos, los pasos intermedios del proceso de análisis vendrán determinados por las configuraciones del autómata. La definición es análoga a la anterior, con la salvedad de que no se incluyen estados en las configuraciones.

Definición 7.5 (configuración)**.**
Dados un AP $\mathcal{A} = (\Sigma, \Gamma, \delta, \$_0, \$_f)$ *y una cadena de entrada* w*, una* configuración *del AP se define como un par (ξ, x), donde ξ es el contenido de la pila y* x *es la parte de la cadena de entrada que resta por analizar.*
La configuración inicial *viene dada por $(\$_0, w)$, siendo* w *la cadena a reconocer. Mientras que la* configuración final *es de la forma $(\$_0\$_f, \varepsilon)$.*
Diremos que una configuración (ξ, ax) deriva en un único paso *otra configuración (ξ', x), si y sólo si existe una transición que aplicada a ξ produce ξ' y lee a de la cadena de entrada:*

$$\begin{array}{ll} (\xi C, ax) \vdash (\xi F, x) & \textit{Transición } \text{HORIZONTAL} \\ (\xi C, ax) \vdash (\xi CF, x) & \textit{Transición } \text{APILAR} \\ (\xi CF, ax) \vdash (\xi G, x) & \textit{Transición } \text{ELIMINAR} \end{array}$$

Si $(\xi, aw) \overset{*}{\vdash} (\xi', w)$ decimos que (ξ, aw) deriva *(ξ', w) en* n *pasos.*

A diferencia de la definición de APs clásica, no existe reconocimiento por estado final. Las configuraciones finales los serán siempre por pila vacía.

Definición 7.6 (lenguaje aceptado por un autómata)**.**
Dado un AP $\mathcal{A} = (\Sigma, \Gamma, \delta, \$_0, \$_f)$*, el* lenguaje aceptado por el autómata *es el conjunto de cadenas $w \in \Sigma^*$:*

$$\mathcal{L}(\mathcal{A}) = \left\{ w \mid (\$_0, w) \overset{*}{\vdash} (\$_0\$_f, \epsilon) \right\}$$

7.2. Análisis sintáctico en ICE

Presentaremos en esta sección la utilización de técnicas de tabulación en el caso de los analizadores sintácticos basados en APs y veremos como permiten superar de forma eficiente los problemas de no determinismo, ambigüedad y repetición de cálculos. En concreto se describe el sistema ICE [56], de generación de analizadores sintácticos para GICs sin restricciones, sobre el que desarrollarán los trabajos posteriores.

7.2.1. Conceptos generales sobre tabulación

La aproximación seguida en ICE se basa en la simulación de las transiciones del AP empleando técnicas de *tabulación*. Las técnicas de tabulación tienen su origen en los trabajos de programación dinámica [5], cuyos principios han sido aplicadas al análisis sintáctico [9, 17, 45] y a la programación lógica [44, 4, 27], bajo diversas formas y nombres: *memoization* [44, 36], *chart parsing* [19, 46], *magic sets* [4], etc, ...

Dado un problema que pueda descomponerse en subproblemas más sencillos, la idea base de las técnicas basadas en tabulación consiste en almacenar la solución de cada uno de estos subproblemas. Se pretende que, de esta forma, no sea necesario resolver dos veces el mismo subproblema. El almacenamiento de estas soluciones, desde el punto de vista lógico, se puede describir como una tabla que almacena las soluciones y a la que se accede a través de unos índices que se corresponden con los subproblemas. Desde este punto de vista, son tres los elementos que caracterizan un algoritmo basado en tabulación:

Granularidad. Es necesario definir el tipo y tamaño de los subproblemas a considerar. Cuanto menores sean estos, mayores serán las posibilidades de reutilización de las soluciones almacenadas, aunque a costa de incrementar el tamaño de la tabla.

Indexación. Es necesario un mecanismo de indexación eficiente que permita acceder al contenido de la tabla a partir de la descripción de un subproblema, para obtener su solución. Una restricción obvia es que el coste del acceso a la solución de un subproblema debe ser menor que el coste de calcular de nuevo esa solución.

Comprobación de redundancia. Es necesario un mecanismo que evite la inclusión de elementos redundantes en la tabla. En este caso, interesará definir una relación de subsumción entre los subproblemas que evite incluir en la tabla soluciones a subproblemas que sean casos particulares de otros más generales cuya solución ya haya sido almacenada.

7.2.2. Interpretación tabular de autómatas con pila

En el caso de los APs aplicados al análisis sintáctico, el problema a tabular se corresponde con el análisis de una cadena de entrada y los subproblemas en que se descompone, son los análisis de distintas porciones de dicha cadena. La forma más directa de representar esas soluciones intermedias del análisis es mediante las configuraciones del autómata. En el caso del sistema ICE se ha pretendido maximizar la reutilización de cálculos y se ha optimizado la representación de las configuraciones de forma que esta información pueda ser compartida y reutilizada al máximo.

La idea en que se basa ICE es la de diferenciar las configuraciones únicamente en función de la parte superior de la pila, permitiendo la compartición del resto. Esta idea se adapta de forma natural a la definición de AP presentada anteriormente, puesto que las transiciones operan únicamente sobre los elementos de la parte superior de la pila. Para aplicar esta idea, se define una relación de equivalencia que permita agrupar todas las configuraciones cuyos n elementos en la cima de la pila coincidan. El representante canónico de cada clase de equivalencia es una imagen compacta de la configuración de la pila asociada. A continuación, será necesario definir una operación que transforme las transiciones originales del autómata, de manera que sean capaces de operar directamente con las clases de equivalencia. Todo estas ideas se recogen en el concepto de *entorno dinámico* [58].

Definición 7.7 (entorno dinámico)**.**
Dado un AP*, $\mathcal{A} = (\Sigma, \Gamma, \delta, \$_0, \$_f)$, un entorno dinámico es un par $(\Re, Op)$, donde:*

- *$\Re$ es una relación de equivalencia sobre las configuraciones del autómata. Las clases de equivalencia se denominarán* ítems*. Notaremos:*
  - *La clase de ξ como $\overline{\xi}$.*
  - *El conjunto de ítems como $It_{\mathcal{A},\Re}$, o simplemente It, cuando no haya ambigüedad.*
- *Op es un operador de la forma:*

$$\begin{array}{rccl} Op: & \delta & \rightarrow & \{It^+ \rightarrow It^+\} \\ & \tau & \rightsquigarrow & Op(\tau) : It^n \rightarrow It^m \end{array}$$

Notaremos $Op(\tau)(It_0, \ldots, It_n) = It$ como $It_0, \ldots, It_n \overline{\vdash\!\!-} It$.

Además, para un entorno dinámico $(\Re, Op)$ *se definen las siguientes propiedades:*

- Compatibilidad*: Todas las computaciones en el autómata tienen su contrapartida en el entorno dinámico.*

$$\forall \tau \in \Theta,\ \xi \in \mathrm{Dom}(\tau), \begin{cases} \exists I_{1\ldots n},\ t.q.\ (\overline{\xi}, I_1, \ldots, I_n) \in \mathrm{Dom}(Op(\tau)) \\ Op(\tau)(\overline{\xi}, I_1, \ldots, I_n) = \tau(\xi) \end{cases}$$

  donde $\mathrm{Dom}(\tau)$ *representa el dominio para* $\tau \in \Theta$*.*

- Completud*: Todas las configuraciones finales en el autómata tienen su contrapartida en el entorno dinámico.*

$$\forall \xi,\ t.q\ \overset{*}{\vdash\!\!\!-} \xi \quad \Rightarrow \quad \overline{\overset{*}{\vdash\!\!\!-}}\ \overline{\xi}$$

- Corrección*: Todas las configuraciones finales en el entorno dinámico tienen su contrapartida en el autómata.*

$$\forall I,\ t.q.\ \overline{\overset{*}{\vdash\!\!\!-}}\ I \quad \Rightarrow \quad \exists \xi,\ t.q.\ \overset{*}{\vdash\!\!\!-} \xi\ y\ \overline{\xi} = I$$

Intuitivamente, la relación de equivalencia permite construir una representación compacta o ítem para cada configuración del autómata, y $\overline{\vdash\!\!\!-}$ corresponde a la aplicación de una transición en el entorno dinámico. Los ítems serán los elementos susceptibles de ser tabulados. Verificándose la propiedad de que cuanto más pequeños sean esos elementos intermedios, mayores serán las posibilidades de compartición. Por su parte, las propiedades de *compatibilidad*, *completud* y *corrección* del entorno dinámico aseguran que los resultados obtenidos realizando un análisis directamente en el entorno dinámico son los mismos que se obtendrían con el autómata original.

Dada la definición de AP que estamos manejando, pueden definirse distintos tipos de entornos dinámicos. Estos pueden ir desde el más sencillo, en el cual no se realiza compactación de la pila y, por lo tanto, no permite compartición, hasta otros con más utilidad práctica, que representan a la pila tomando uno o dos símbolos de su cima.

Entorno dinámico S^T También llamado *entorno dinámico estándar*. En él cada configuración del autómata está representada por todos los elementos de la pila, sin realizar ningún tipo de compactación.

Intuitivamente, en S^T cada ítem equivale a una configuración del autómata, por lo tanto las propiedades de compatibilidad, completud y corrección son inmediatas. Como contrapartida, al no haber compactación, la compartición y reutilización son prácticamente nulas.

Entorno dinámico S^2 Definido por Lang en [27]. Para la representación de los ítems se emplean los dos elementos de la cima de la pila:

$$\langle A, A' \rangle = \{AA'\xi\}$$

y el operador Op se define como:

- $Op(C \longmapsto F)(\langle A, A' \rangle) = \langle F, A \rangle,$ si $C = A$
- $Op(C \longmapsto CF)(\langle A, A' \rangle) = \langle F, A \rangle,$ si $C = A$
- $Op(CF \longmapsto G)(\langle A, A' \rangle, \langle A', E \rangle) = \langle A, G \rangle,$ si $C = A'$ y $F = E$

La principal ventaja de este entorno es la sencillez a la hora de definirlo, y a la hora de demostrar sus propiedades de compatibilidad, completud, corrección. Esto es así dado que siempre podemos suponer una representación canónica del AP en la que las transiciones dependan, a lo sumo, de los

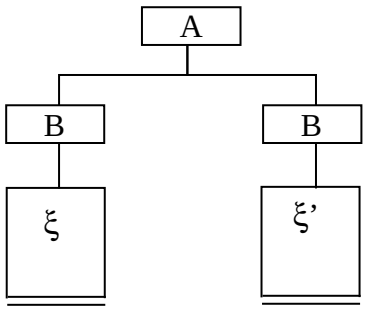


Figura 7.1: Aplicabilidad de las transiciones dinámicas.

dos últimos elementos de la pila [14]. Como desventaja, su capacidad de compartición es menor el entorno S^1, que se describe a continuación.

Entorno dinámico S^1 Propuesto por Vilares [56] para el caso de las GICs y por Clergerie [58] para el caso de los programas de cláusulas definidas. Los ítems se construyen a partir del elemento en la cima de la pila, con lo que se alcanza el mayor grado de compactación posible:

$$\langle A \rangle = \{A\xi\}$$

En este caso la construcción del operador Op es más compleja, puesto que exige la introducción del concepto de *transición dinámica* para tratar las transiciones de tipo ELIMINAR:

- $Op(C \longmapsto F)(\langle A \rangle) = \langle F \rangle$, si $C = A$
- $Op(C \longmapsto CF)(\langle A \rangle) = \langle F \rangle$, si $C = A$
- $Op(CF \longmapsto G)(\langle A \rangle) = (C \longmapsto G)$, si $F = A$

Las transiciones dinámicas son transiciones de tipo HORIZONTAL que se crean de forma dinámica. Su misión es la de completar el segundo paso de una transición de tipo ELIMINAR $CF \longmapsto G$ sobre el ítem $\langle A \rangle$. La eliminación del primer símbolo, F, de la cima de la pila es inmediata. Sin embargo, el segundo paso, eliminar C de la cima de la pila sustituyéndolo por G, no es posible realizarlo en ese instante, ya que el ítem no contiene información sobre el resto de la pila. En este caso se creará una nueva transición que se aplicará a todos los ítems en el contexto de $\langle A \rangle$. La figura 7.1 muestra esta idea. La transición ELIMINAR$(BA \longmapsto D)(\langle A \rangle)$ genera la transición dinámica $B \longmapsto D$ que se aplicará sobre los dos ítems $\langle B \rangle$, independientemente del orden en que se creen.

La principal desventaja de este entorno dinámico reside en que la corrección no está garantizada [58, 56].

La figura 7.2 muestra un ejemplo del uso de estos tres entornos dinámicos durante un análisis no determinista. En este caso tal como muestra la figura, en determinado momento del análisis surgieron tres alternativas distintas. La figura de la izquierda muestra la representación empleando un entorno S^T. Como se puede apreciar, la representación de las configuraciones del AP utiliza la totalidad de las pilas asociadas a los posibles análisis, por lo que existe compactación de las pilas. En el centro de la figura su muestra la representación de los posibles análisis utilizando el entorno S^2. Como se puede apreciar, sí se comparten las partes comunes a las tres pilas, que en este caso es únicamente un ítem. Por último, el esquema de la derecha muestra la compartición para el caso del entorno dinámico S^1. Donde se pone de manifiesto que las posibilidades de compartición son mayores que con el entorno S^2.

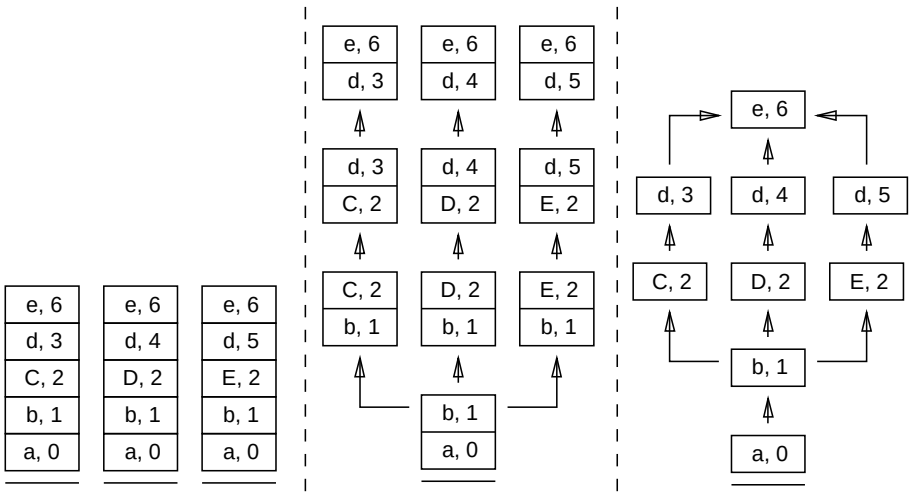


Figura 7.2: Compartición en los entornos dinámicos S^T, S^2 y S^1.

7.2.3. Tabulación en ICE

El sistema ICE, por *Incremental Context-free Environment*, es descrito por Vilares en [56]. ICE es un generador de analizadores sintácticos incrementales para GICs, sin restricciones. Ante la presencia de una ambigüedad en el proceso de análisis, procesa cada una de las alternativas de forma concurrente, lanzando para cada una de ellas una *rama de análisis* diferente. Para reducir el número de ramas de análisis necesario y mejorara la velocidad de tratamiento, se incorpora un control finito que guía a la estrategia de evaluación empleada. Al mismo tiempo se busca la máxima compartición de cálculos entre las distintas ramas aplicando técnicas de tabulación.

La base de la estrategia de evaluación en ICE son los autómatas de tipo LR(k), en concreto los LALR(1). Para la tabulación del autómata usado en ICE, se ha optado por el entorno dinámico S^1, por ser el que posibilita el mayor grado de compartición. Uno de los objetivos de ICE es mantener la complejidad temporal del algoritmo en $\mathcal{O}(n^3)$ para el peor de los casos y $\mathcal{O}(n)$ para las gramáticas LALR(1)[2]. Para conseguir esta complejidad se descompone cada paso de reducción del autómata LALR(1), que implica m antecedentes, en $m+1$ pasos con un máximo de 2 antecedentes. De esta forma se obtiene la complejidad deseada. La división del paso de reducción se realiza mediante la binarización implícita de las producciones [28]. Para ello se hace uso de un conjunto de nuevos símbolos de pila, los símbolos $\nabla_{r,i}$. Cada uno de estos símbolos representa el reconocimiento de los i primeros símbolos del lado derecho de la regla número r, cuando $i > 0$, o que aún no se ha iniciado el reconocimiento de la regla r, en el caso $i = 0$. Así, la reducción de una producción:

$$A_{r,0} \rightarrow A_{r,1}\, A_{r,2} \ldots A_{r,n_r}$$

se realiza de forma equivalente como la siguiente secuencia de reducciones:

$$\begin{aligned}
&A_{r,0} \rightarrow A_{r,1}\, \nabla_{r,1}\\
&\nabla_{r,1} \rightarrow A_{r,2}\, \nabla_{r,2}\\
&\vdots\\
&\nabla_{r,n_r-1} \rightarrow A_{r,n_r}\, \nabla_{r,n_r}\\
&\nabla_{r,n_r} \rightarrow \varepsilon
\end{aligned}$$

[2]Es decir, el analizador se comporta de forma determinista para el tipo de gramáticas analizables por el autómata elegido como control finito.

Como se ha indicado, el funcionamiento del analizador se basa en la generación de *ítems* de tipo S^1, que representan de forma compacta los pasos intermedios del análisis. Para realizar la tarea de tabulación, estos ítems se almacenan en *conjuntos de ítems*. De este modo, se asocia un conjunto de ítems S_i^w a cada símbolo terminal w_i de la cadena de entrada de longitud n, w, y se añade un conjunto de ítems S_0^w que se corresponde con el inicio del análisis. Los ítems almacenados siguen la definición de entorno dinámico S^1, incluyendo, además de la representación compacta de la pila, información sobre el estado del autómata LALR(1) e información sobre su posición en la cadena de entrada. Su definición es la siguiente:

Definición 7.8 (ítem en ICE)**.**
Un ítem en ICE *es una cuádrupla* $[p, X, S_i^w, S_j^w]$*, que representa de forma compacta la configuración actual del reconocedor* LALR(1)*, donde:*

- p *es el estado actual en el reconocedor* LALR(1)*.*
- X *es el último símbolo de pila reconocido desde* p*.*
- S_i^w *es una referencia al conjunto de ítems asociado a la posición* i *de la cadena de entrada* w *donde se encuentra el primer terminal que se deriva de* X*. Indica el inicio de la porción de la cadena de entrada reconocida por el ítem.*
- S_j^w *es una referencia al conjunto de ítems actual, en el que se almacena el ítem, asociado a la posición* j *que está siendo actualmente analizada de la cadena de entrada* w*.*

El analizador generado por ICE comenzará con un ítem inicial $[0, \varepsilon, S_0^w, S_0^w]$ que representan la configuración inicial del autómata. A partir de aquí, se generarán nuevos ítems aplicando transiciones sobre los ítems ya existentes hasta que no se puedan generar nuevos ítems. Esas transiciones sobre ítems se construyen a partir de las transiciones del analizador LALR(1) original utilizando el operador de transformación del entorno dinámico S_1.

Así, dada una transición $\tau \in \delta \ / \ \tau = \delta(p, X, a) \ni (q, Y)$ del AP $\mathcal{A} = (\Sigma, \Gamma, \delta, \$_0, \$_f)$. Se traducirá esta transición de S^T al entorno dinámico S^1 como $\tilde{\delta} = Op(\tau)$:

1.	$\tilde{\delta}([p, X, S_j^w, S_i^w], a)$	$\ni$	$[q, \varepsilon, S_i^w, S_i^w]$	**si** $Y = X$
2.	$\tilde{\delta}([p, X, S_j^w, S_i^w], a)$	$\ni$	$[p, Y, S_i^w, S_{i+1}^w]$	**si** $Y = a$
3.	$\tilde{\delta}([p, X, S_j^w, S_i^w], a)$	$\ni$	$[p, Y, S_i^w, S_i^w]$	**si** $Y \in N$
4.	$\tilde{\delta}([p, \varepsilon, S_j^w, S_i^w], a)$	$\ni$	$\tilde{\delta}_d([q, \varepsilon, S_l^w, S_j^w], a) \ni [q, \varepsilon, S_l^w, S_i^w]$	**si** $Y = \varepsilon$ $\forall q \in \delta$ tal que $\exists\ \delta(q, X, \varepsilon) \ni (p, X)$

con:

$$\tilde{\delta} : \mathrm{It} \times \Sigma \cup \{\varepsilon\} \longrightarrow \{\mathrm{It} \cup \tilde{\delta}_d\} \times \Pi^* \qquad \tilde{\delta}_d : \mathrm{It} \times \Sigma \cup \{\varepsilon\} \longrightarrow \mathrm{It}$$

donde It es el conjunto de todos los ítems generados durante el proceso de análisis y $\tilde{\delta}_d$ es el conjunto de *transiciones dinámicas*.

Intuitivamente, se pueden explicar los casos anteriores como sigue:

1. El cambio de estado desde p al estado q después del análisis de X en el analizador LALR(1), almacenando el nuevo ítem en el conjunto de ítems actual.
2. El apilamiento del terminal a en el estado p, almacenando el nuevo ítem en el siguiente conjunto de ítems.
3. El apilamiento de un no terminal Y en el estado p, almacenando el nuevo ítem en el conjunto de ítems actual.

4. La eliminación de la cima de la pila en el estado p, donde q es un estado ancestro de p bajo la transición en la fue reconocido el símbolo X. En este caso, no se genera un nuevo ítem, sino una *transición dinámica* $\tilde{\tau}_d$ para tratar la ausencia de información sobre el contenido del resto de la pila.

Como se ha adelantado, el uso de transiciones dinámicas es una necesidad derivada de la utilización de un entorno dinámico de tipo S^1. En este tipo de entornos sólo se almacena información sobre el elemento de la cima de la pila. En transiciones de eliminación como la descrita en el caso 4, dado que no se dispone de información sobre el siguiente elemento de la pila, es necesario generar un conjunto de transiciones dinámicas que representen la eliminación de cada uno de los posibles símbolos que pudieran haber dado origen a esa cima de pila. En el esquema de análisis usado en ICE, sólo se producen transiciones dinámicas durante la reducción. Una vez creada una transición dinámica, en principio, por cada nuevo ítem que generemos *a posteriori*, se deberá comprobar si la transición dinámica es aplicable.

Para asegurar la corrección y facilitar la implementación, se establece un mecanismo de *sincronización* del análisis. Como se ha indicado, el analizador construye, de forma secuencial, un conjunto de ítems por cada posición de la cadena de entrada. Cada uno de estos conjuntos sirve como punto de sincronía, al exigirse que todo conjunto de ítems se complete antes de pasar al siguiente. La inclusión del mecanismo de sincronización permite optimizar el manejo de las transiciones dinámicas en dos sentidos [56]:

- Su ámbito de aplicabilidad se restringe al conjunto de ítems en el que fueron generadas. De esta forma, una vez completada la construcción de un conjunto de ítems, es posible ignorar todas las transiciones dinámicas generadas hasta ese momento.

- La generación de transiciones dinámicas dentro de un conjunto de ítems sólo es necesaria bajo determinadas circunstancias, derivadas de la presencia de no determinismo en el AP LALR(1) de partida. En esos casos será posible generar dos ítems que compartan el mismo contexto sintáctico, lo cual es, a su vez, condición necesaria para que sea posible aplicar una transición dinámica.

En lo que respecta a la comprobación de redundancias dentro de la tabla de ítems, es suficiente establecer una condición de igualdad entre ítems. Esto es así, dado se está tratando sobre GICs y basta con considerar relaciones de subsumción triviales. El mecanismo de comprobación de redundancias implica que, como paso previo a la introducción de un ítem en la tabla, se compruebe que no existe otro ítem igual[3]. Para mejorar la eficiencia de este mecanismo, ICE utiliza una función de indexación de la tabla, que permita accesos directos. Es importante señalar que el hecho de incluir al estado en la comprobación de redundancias puede ocasionar una merma en las posibilidades de compartición. Puede suceder que la distinción entre estados diferentes del control finito evite la compartición de ítems cuyo contexto sintáctico es el mismo.

7.3. Bosques compartidos

La salida del proceso de análisis sintáctico debe ser algún tipo de representación que muestre como la sentencia de entrada puede ser construida de acuerdo a las reglas de la gramática. Generalmente esta representación toma la forma de árboles de análisis sintáctico. Al manejar sentencias sentencias ambiguas tendremos que es posible construir múltiples árboles de análisis que representen las distintas formas de generarlas. En estos casos, es deseable disponer de una representación compacta de todos esos posibles análisis en una única estructura.

En esta sección se revisarán una serie de generalidades sobre la representación de análisis sintácticos, centrándonos en el problema de la representación de análisis ambiguos y con estructuras compartidas. A continuación, se describirá el uso de GICs como formalismo de representación compacta de árboles de análisis para el caso de sentencias ambiguas, y que servirán como formalismo de representación en ICE.

[3]Es decir, mismo símbolo, mismo estado, misma posición de inicio y mismo conjunto de ítems.

(1)	S	$\rightarrow$	$SN\ SV$	(6)	SV	$\rightarrow$	$verbo\ SN$
(2)	SN	$\rightarrow$	nom	(7)	SV	$\rightarrow$	$verbo\ SN\ SP$
(3)	SN	$\rightarrow$	$det\ nom$	(8)	SP	$\rightarrow$	$prep\ SN$
(4)	SN	$\rightarrow$	$det\ nom\ SP$				
(5)	SN	$\rightarrow$	$det\ nom\ SP\ SP$				

Figura 7.3: Gramática G_{SN-SP}

7.3.1. Arboles y bosques de análisis

Nos centraremos, entonces, en la generación y uso de árboles de análisis sintáctico como formalismo de representación de la estructura sintáctica de las sentencias analizadas. El objetivo de todo algoritmo de análisis será construir una árbol que represente en que forma se puede construir la sentencia de entrada utilizando los elementos de la gramática: terminales, no terminales y reglas de producción. En el caso de sentencias o gramáticas ambiguas esa representación no será un único árbol de análisis, sino un conjunto de posibles árboles sintácticos. Ese conjunto de árboles conformará lo que denominaremos un *bosque de análisis*.

A la hora de representar los múltiples árboles de análisis asociados a una sentencia es deseable contar con un formalismo de representación lo más eficiente posible, tanto en lo relativo a sus necesidades de espacio de almacenamiento, como en las facilidades que ofrezca para su posterior procesamiento. El primer aspecto es crucial, máxime si se tiene cuenta que el número de posibles árboles de análisis puede ser muy grande, especialmente en el caso de sentencias muy largas. Además, en muchos casos, ese número de análisis puede crecer de forma exponencial con el tamaño de la entrada, y puede llegar a ser infinito si se manejan gramáticas con ciclos. Dado que en muchas aplicaciones prácticas es deseable disponer de todos los posibles árboles de análisis, como por ejemplo en PLN, es conveniente representar simultáneamente y de forma compacta todos esos árboles en una única estructura que permita la compartición de las partes que sean comunes a más de un análisis. La principal ventaja de este tipo de compartición es el ahorro de espacio de almacenamiento, pero esta compartición de estructuras también puede permitir, en fases posteriores, la compartición del procesamiento efectuado sobre esas partes comunes.

Este tipo de representaciones compactas de todos los árboles de análisis, en las cuales se comparten estructuras comunes, se denominan *bosques de análisis compartidos*. Generalmente, esos bosques de árboles compartidos tienen la forma de *grafos* Y-O.

Definición 7.9 (grafo Y-O)**.**
Un grafo Y-O $G = (A, R)$ *es un grafo dirigido en cual se verifican las siguientes condiciones:*

- $A = A_{\mathrm{O}} \cup A_{\mathrm{Y}}$, *con* $A_{\mathrm{O}} \cap A_{\mathrm{Y}} = \emptyset$.
- $\forall (a_i, a_j) \in R$, $a_i \in A_{\mathrm{O}}$ *y* $a_j \in A_{\mathrm{Y}}$ *ó* $a_i \in A_{\mathrm{Y}}$ *y* $a_j \in A_{\mathrm{O}}$.

Intuitivamente un grafo Y-O es un grafo $G = (A, R)$ en el cual se distinguen dos tipos de nodos, nodos-O y nodos-Y. Es decir, el conjunto de nodos A se divide en dos subconjuntos de nodos disjuntos, A_{O} y A_{Y}. Así, los nodos $o \in A_{\mathrm{O}}$ se denominan *nodos*-O y los nodos $y \in A_{\mathrm{Y}}$ se denominan *nodos*-Y. Además, sólo se permiten enlaces entre nodos de distinto tipo. Es decir, los pares (a_i, a_j) incluidos en la relación R son únicamente de dos tipos. O bien. enlazan a un nodo-O con un nodo-Y, esto es $a_i \in A_{\mathrm{O}}$ y $a_j \in A_{\mathrm{Y}}$. O bien, enlazan a un nodo-O con un nodo-Y, esto es $a_i \in A_{\mathrm{Y}}$ y $a_j \in A_{\mathrm{O}}$. Una vez introducido el concepto de grafo Y-O se puede definir formalmente un *bosque de análisis compartido*.

Definición 7.10 (bosque de análisis compartido)**.**
Un bosque de análisis *para una GIC* $\mathcal{G} = (N, \Sigma, P, S)$ *es un grafo* Y-O *etiquetado* $G = (A, R)$, *que verifica:*

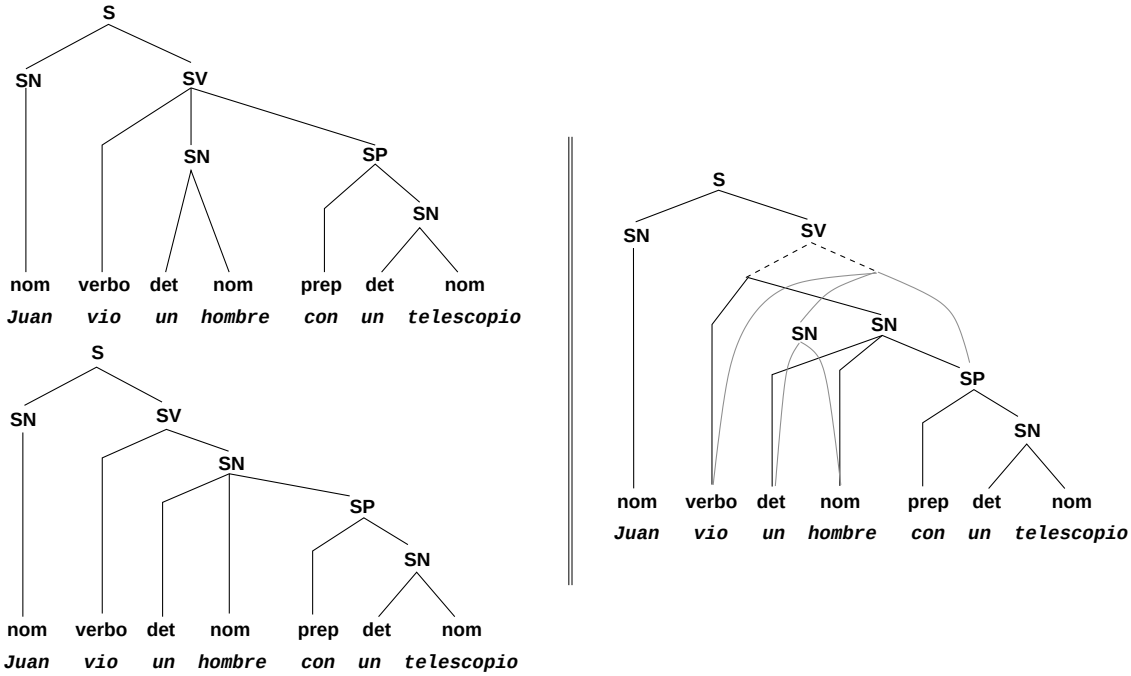


Figura 7.4: Ejemplo de bosques de análisis compartidos.

- *El único nodo de G cuyo grado de entrada es 0 está etiquetado como S y se denomina* raíz.
- *Si $X_1, X_2, \ldots, X_n$ son los subgrafos asociados a los nodos-*O *enlazados con S, y la raíz de X_i está etiquetada como S_i, entonces:*
  - *$S \rightarrow S_1, S_2, \ldots S_n$ es una producción de P.*
  - *Si $X_i \notin \Sigma$, X_i es un bosque de análisis compartido.*

Si se verifica que en G no existen nodos duplicados, se dirá que G es un bosque de análisis compartido.

Como se deduce de la definición, los bosques de análisis compartidos son una extensión de los árboles de análisis presentados en el capítulo 2, en la cual se utilizan grafos Y-O en lugar de árboles para representar los distintos análisis de la cadena de entrada. Un diferencia importante es que, en el caso de los bosques compartidos, se exige que no existan nodos duplicados. Esta propiedad asegurará que en el caso de análisis ambiguos las estructuras comunes a más de un análisis se representen una única vez.

En el caso de grafos Y-O empleados en el análisis sintáctico, los hijos de un nodo-Y representan la lista ordenada de constituyentes en los que se puede derivar el símbolo gramatical con el que se corresponde ese nodo, de acuerdo a una de las reglas de la gramática. Para el otro tipo de nodos, los nodos-O, cada uno de sus hijos representa una alternativa posible en el análisis del símbolo asociado a dicho nodo. Esto es, los nodos-O representan un conjunto de derivaciones posibles para un símbolo no terminal de la gramática.

Emplearemos la gramática $\mathcal{G}_{SN-SP}$, cuyas reglas se muestran en la figura 7.3, para mostrar un ejemplo de un bosque compartido. Los dos árboles de la parte izquierda de la figura 7.4 muestran los dos posibles análisis de la sentencia *"Juan vio un hombre con un telescopio"*. En la parte derecha de esa figura se muestra una representación en forma de grafo Y-O del bosque compartido obtenido a partir de esos dos análisis. En este grafo, los nodos-O se distinguen porque sus descendientes están enlazados mediante lineas discontinuas, representando que se trata de alternativas de análisis distintas. También se puede observar la compartición de estructuras comunes a los dos análisis, que se distinguen por ser subárboles cuyas raíces tienen más de un padre, como por ejemplo el subárbol con raíz en el nodo SP.

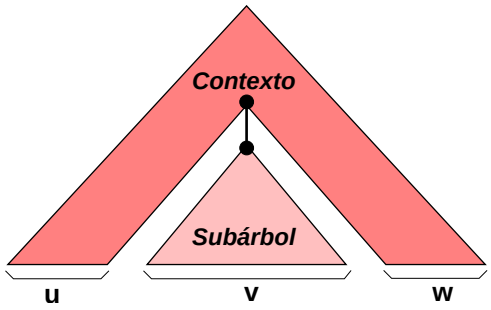


Figura 7.5: Componentes de los bosques compartidos.

7.3.2. Compartición en bosques de análisis

Veremos ahora las distintas posibilidades de compartición de estructuras sintácticas en este tipo de bosques compartidos. Como se muestra en la figura 7.5, supongamos que la parte compartida entre dos o más análisis de la cadena uvw se corresponde con el subárbol asociado a la subcadena v. Se puede aislar esa estructura simplemente eliminando el arco que une a la raíz de ese subárbol con su nodo padre. Esto da lugar a las dos estructuras que se muestran en esta figura, el *subárbol* y el *contexto*. Cualquiera de esas dos estructuras puede ser compartida en el bosque compartido que represente los múltiples análisis de esa cadena. Tenemos dos posibilidades:

- Cuando el *subárbol* es el mismo en dos o más análisis, tenemos la situación mostrada en la parte izquierda de la figura 7.6, lo que denominaremos *compartición de un subárbol*.
- Cuando es el *contexto* la parte que es común a dos o más análisis, tenemos un caso de *compartición del contexto*, mostrado en el esquema la parte derecha de la figura 7.6. En este caso el nodo padre del subárbol compartido será un nodo-O que puede tener dos o más subárboles descendientes alternativos, como se muestra en la figura.

El segundo tipo de compartición, la compartición del contexto, está originado por las ambigüedades presentes en la sentencia analizada. Esto es, tendremos un no terminal B, que estará asociado al nodo-O y que podremos derivar de dos o más formas para obtener la subcadena v. El análisis de las otras dos porciones de la cadena de entrada, u y w, es único, lo que dará lugar a la compartición del contexto donde se enmarca ese no terminal B, causante de la ambigüedad.

7.3.3. Gramáticas independientes del contexto y bosques Y-O

En [28, 6], Lang *et al.* ofrecen una visión novedosa de la relación entre GICs y bosques de análisis compartidos, en concreto de los grafos Y-O . La idea básica de su aportación es que tanto las GICs como los bosques compartidos pueden representarse como grafos Y-O. Estos autores ponen de manifiesto el hecho de que no existe diferencia entre un bosque compartido y una GIC, concluyendo que es posible la utilización de GICs para representar de forma eficiente los bosques compartidos generados durante el proceso de análisis sintáctico.

La idea de Lang en [28] se basa en establecer una correspondencia entre los elementos de las gramáticas: terminales, no terminales y reglas, y los nodos de un grafo Y-O. Así, la representación de una GIC es trivial. La figura 7.7 muestra un ejemplo utilizando la gramática $\mathcal{G}_{SN-SP}$. Los nodos-O se corresponden con los símbolos no terminales de la gramática. Los nodos-Y se corresponden con las reglas de la gramática y se representan mediante un número asociado a cada regla. Finalmente, en las hojas se sitúan los símbolos terminales de la gramática. Cada nodo-O, asociado a un no terminal B, se enlazará con los nodos-Y correspondientes a las reglas que tengan a B como su antecedente. Para simplificar la figura se ha evitado dibujar este enlace en los casos donde sólo exista una regla para el no terminal. Los hijos de un nodo-Y,

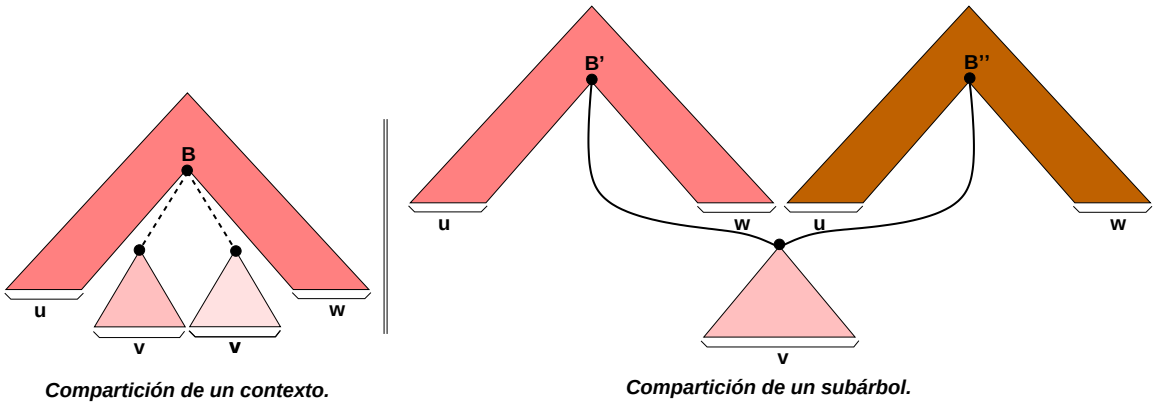


Figura 7.6: Tipos de compartición en bosques de análisis compartidos.

asociado a una regla i, serán una lista ordenada con los nodos-O correspondientes a los no terminales y/o las hojas asociadas a los terminales que aparezcan en el lado derecho de la regla i.

Como se puede apreciar en la figura 7.7, el grafo Y-O asociado a una gramática tiene la particularidad de que todos sus nodos son distintos. Esta propìedad puede interpretarse en sentido inverso. Así, se puede afirmar que cualquier grafo Y-O[4] en el cual todos sus nodos estén etiquetados de forma única puede interpretarse como una GIC. Los no terminales de esa gramática serán las etiquetas de los nodos-O, las etiquetas de las hojas serán sus terminales y las reglas se construirán a partir de los hijos de los nodos-Y. Cuando la gramática G_x así generada representa un grafo Y-O obtenido del análisis sintáctico de una cadena w, dicha gramática tiene la particularidad de que el lenguaje que genera, $\mathcal{L}(G_X)$, contiene una única cadena. Además, dicha cadena resulta ser la propia cadena w, es decir $\mathcal{L}(G_X) = \{w\}$. Esto es consecuencia del hecho de que todos los nodos del grafo sean distintos. De esta forma, las reglas que se extraigan únicamente podrán generar la cadena w cuyo análisis es representado por el grafo Y-O.

Hemos visto que cualquier bosque compartido representado como grafo Y-O puede transformarse en una GIC. Para lograrlo, simplemente es necesario asegurar que todos los nodos del grafo estén etiquetados de forma única. Una posible forma de asegurar que se cumple esta condición es numerar todos los nodos del grafo de forma que todos ellos sean distintos. Volvemos a considerar el grafo mostrado en la parte derecha de la figura 7.4 que representa los dos posibles análisis de la cadena *"Juan vio un hombre con un telescopio"* empleando la gramática G_{SN-SP}. Se puede asignar un número a cada nodo de ese grafo para hacerlos todos distintos, tal y como se muestra en la parte izquierda de la figura 7.8. De este modo, se puede representar el grafo resultante mediante las reglas de la gramática mostrada a su derecha en esa misma figura. Como se puede observar en la lista de reglas, el nodo-O etiquetado SV_4 da lugar a dos reglas con el símbolo SV_4 en su lado izquierdo, que representan a las dos alternativas de ese nodo-O. También se puede apreciar la compartición de estructuras que origina que un símbolo, como por ejemplo SP_{10}, aparezca en el lado derecho de dos o más reglas.

7.4. Generación de bosques compartidos en ICE

Una vez revisadas las técnicas de representación de estructuras sintácticas y presentado el uso de GIC para representar árboles compartidos, se describirá como se lleva a cabo la generación de la estructura sintáctica correspondiente al análisis de la cadena de entrada en ICE. Como se había adelantado, dicha estructura será un bosque de análisis, donde se reflejarán las ambigüedades y la compartición de cálculos. En concreto, la salida del análisis efectuado por el sistema ICE tiene la forma de una GIC que representa de forma compacta todos los posibles árboles de análisis. En esta sección se mostrará como se adapta el

[4]En nuestro caso será un grafo Y-O, representando todos los posibles análisis sintácticos de una sentencia.

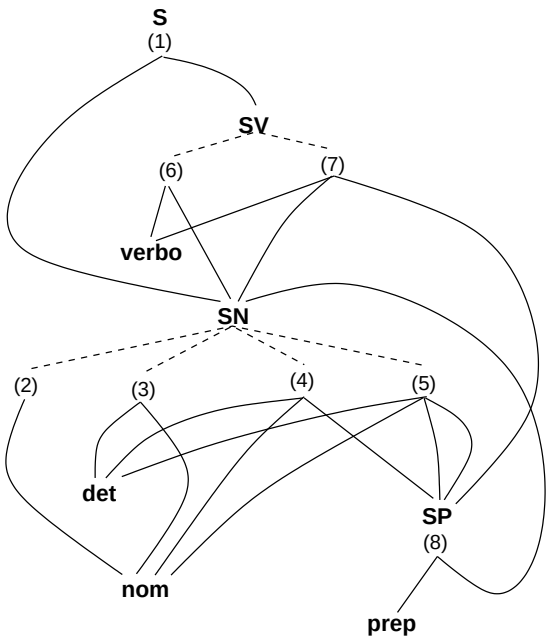


Figura 7.7: Representación de G_{SN-SP} como un grafo Y-O.

formalismo descrito con anterioridad para permitir la construcción de la estructura sintáctica resultante del análisis de una cadena. Se revisarán también las características específicas de los árboles generados por ICE , prestando atención especial a los aspectos relativos a la compartición de estructuras.

7.4.1. Traductores con pila

Para obtener el bosque sintáctico como resultado del análisis, es necesario sustituir los APs por el uso de *traductores con pila*, en adelante TPs, como formalismo operacional.

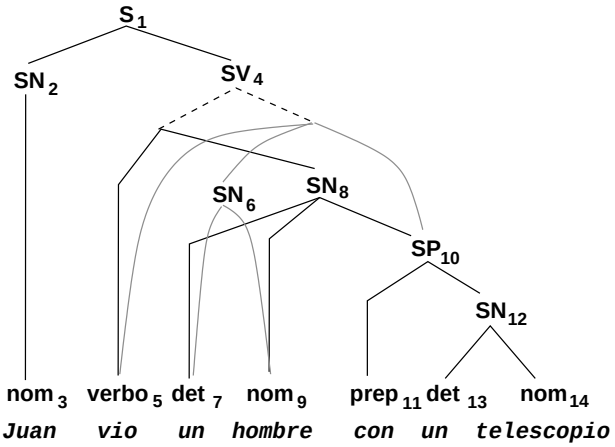
Definición 7.11 (traductor con pila)**.**
Un traductor con pila *es una tupla* $\mathcal{T} = (\Sigma, \Gamma, \Pi, \delta, \$_0, \$_f)$*, donde* Σ*,* Γ*,* $\$_0$*, y* $\$_f$ *tienen el mismo significado que en un AP, y:*

- Π *es un conjunto de símbolos de salida*
- δ *es un conjunto de* transiciones

$$\begin{array}{rccc} \delta: & \Gamma^* \times (\Sigma \cup \{\varepsilon\}) & \longrightarrow & \Gamma^* \times \Pi^* \\ & (\alpha, a) & \rightsquigarrow & (\beta, u) \end{array}$$

La característica distintiva de los TPs con respecto a los APs, es que este nuevo formalismo permite especificar la salida que debe ser generada tras la aplicación de una transición. En consecuencia, una *configuración del traductor*, será similar a la configuración de un autómata, con el añadido de un nuevo elemento u, donde se acumulará la salida generada por cada transición aplicada: (ξ, x, u).

En el caso de ICE, lo que el TP irá acumulando en sus configuraciones es el conjunto de reglas de producción que definen la *gramática de salida* $\mathcal{G}_o$. Esta gramática de salida representará de forma compacta el grafo Y-O correspondiente al bosque compartido que incluye a todos los posibles árboles de análisis de la cadena de entrada. La gramática de salida, $\mathcal{G}_o = (N_o, \Sigma_o, P_o, S_o)$, se construirá durante



Gramática ascociada:

$$
\begin{array}{lcl}
S_1 & \rightarrow & SN_2\ SV_4 \\
SN_2 & \rightarrow & nom_3 \\
SV_4 & \rightarrow & verbo_5\ SN_8 \\
SV_4 & \rightarrow & verbo_5\ SN_6\ SP_{10} \\
SN_6 & \rightarrow & det_7\ nom_9 \\
SN_8 & \rightarrow & det_7\ nom_9\ SP_{10} \\
SP_{10} & \rightarrow & prep_{11}\ SN_{12} \\
SN_{12} & \rightarrow & det_{13}\ nom_{14}
\end{array}
$$

Figura 7.8: Transformación de grafos Y-O en GICs.

el proceso de análisis de la siguiente forma. Los ítems generados a medida que avance el análisis de la entrada serán usados como no terminales, por lo que N_o se corresponderá con el conjunto de todos los ítems. El conjunto de terminales de G_o coincide con el conjunto de terminales de la gramática original $\mathcal{G}$. Las reglas de P_o son construidas por el algoritmo de análisis a medida que se van generando nuevos ítems. Se genera una regla de P_o cada vez que reconocemos una categoría sintáctica. La parte izquierda de dichas reglas es el ítem que representa dicho reconocimiento. En cuanto a la parte derecha, en caso de reconocimiento de un terminal mediante un salto, será el propio terminal reconocido. En el caso de que se reconozca un no terminal mediante una reducción, la parte derecha de la regla de salida estará formada por los ítems retirados de la cima de la pila. Finalmente, el símbolo inicial S_o será el último ítem producido por un reconocimiento satisfactorio de la cadena de entrada.

Formalmente, la interpretación tabular de los TPs en ICE y la generación de las reglas de la gramática de salida será de la siguiente forma, análoga al caso de los APs. Dado un TP $\mathcal{T} = (\Sigma, \Gamma, \Pi, \delta, \$_0, \$_f)$ y una transición $\tau \in \delta \ / \ \tau = \delta(p, X, a) \ni (q, Y, u)$. Se traduce esta transición de S^T al entorno dinámico S^1 como $\tilde{\delta} = Op(\tau)$:

$$
\begin{array}{lllll}
1. & \tilde{\delta}([p, X, S_j^w, S_i^w], a) & \ni & ([q, \varepsilon, S_i^w, S_i^w], & \varepsilon) \quad \textbf{si } Y = X \\
2. & \tilde{\delta}([p, X, S_j^w, S_i^w], a) & \ni & ([p, Y, S_i^w, S_{i+1}^w], & I_0 \rightarrow a) \quad \textbf{si } Y = a \\
3. & \tilde{\delta}([p, X, S_j^w, S_i^w], a) & \ni & ([p, Y, S_i^w, S_i^w], & I_1 \rightarrow I_2) \quad \textbf{si } Y \in N \\
4. & \tilde{\delta}([p, \varepsilon, S_j^w, S_i^w], a) & \ni & \tilde{\delta}_d([q, \varepsilon, S_l^w, S_j^w], a) \ni ([q, \varepsilon, S_l^w, S_i^w], & I_3 \rightarrow I_4 I_5) \quad \textbf{si } Y = \varepsilon \\
 & & & \forall q \in \delta \text{ tal que } \exists\ \delta(q, X, \varepsilon) \ni (p, X) &
\end{array}
$$

con:

$$\tilde{\delta} : \mathrm{It} \times \Sigma \cup \{\varepsilon\} \longrightarrow \{\mathrm{It} \cup \tilde{\delta}_d\} \times \Pi^* \qquad \tilde{\delta}_d : \mathrm{It} \times \Sigma \cup \{\varepsilon\} \longrightarrow \mathrm{It}$$

donde It es el conjunto de todos los ítems generados en el proceso de análisis y $\tilde{\delta}_d$ es el conjunto de *transiciones dinámicas*. Siendo los elementos de salida:

$$
\begin{array}{lll}
I_0 = [p, Y, S_i^w, S_{i+1}^w] & I_1 = [p, Y, S_i^w, S_i^w] & I_2 = [p, X, S_j^w, S_i^w] \\
I_3 = [q, \varepsilon, S_l^w, S_i^w] & I_4 = [q, X, S_l^w, S_j^w] & I_5 = [p, \varepsilon, S_j^w, S_i^w]
\end{array}
$$

Tal como se puede apreciar en la definición de las transiciones, se generará una regla de la gramática de salida cada vez que se produzca una acción de salto o reducción sobre la pila del TP. Es importante señalar que, dado que se realiza la binarización implícita de las reglas de la gramática original y las acciones realizadas sobre la pila del TP dependen de, a lo sumo, dos ítems, las reglas de la gramática de salida resultante tendrán también como máximo dos ítems en su parte derecha. La figura 7.9 muestra como serán los árboles resultantes de la aplicación de una acción de salto y una acción de reducción. Para

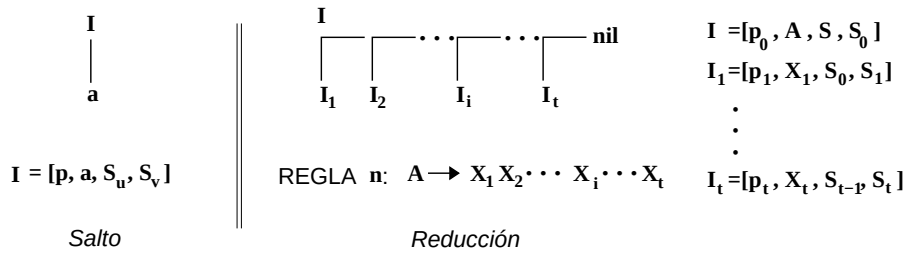


Figura 7.9: Generación de nodos en ICE.

simplificar la representación, en esta figura no se incluyen explícitamente los ítems correspondientes a los símbolos adicionales $\nabla_{r,i}$ originados en el proceso de binarización implícita de las reglas.

Tenemos pues, que cada transición del traductor con pila generará un nodo-Y del bosque de análisis, que se corresponderá con producciones de la gramática de salida, cuyos símbolos serán referencias a los ítems generados en el entorno dinámico S^1. Cuando se produzca la compartición de cálculos entre ítems provenientes de distintas ramas de análisis, la salida de dichos ítems se compactará en un nodo-O, que se corresponderá con la existencia de varias reglas en la gramática de salida con ese mismo ítem en su parte izquierda. El mecanismo de subsumción basado en la igualdad de los ítems asegura que no se genera el mismo ítem dos veces. Al emplear este tipo de comprobación de redundancias tenemos la seguridad de que, de ningún modo, puede haber dos nodos iguales en el grafo Y-O que representa el resultado del análisis y, por lo tanto, es posible representarlo en forma de GIC. Por lo tanto, la GIC de salida, $\mathcal{G}_o$, que genera ICE representa, efectivamente, un grafo Y-O que contiene los posibles análisis de la cadena de entrada.

7.4.2. Compartición de estructuras en ICE

La relación de subsumción asegura la máxima compartición entre los ítem, con el consiguiente ahorro de espacio. Esta compartición, junto con el hecho de que las reglas de la gramática de salida tienen dos elementos en su parte derecha como máximo hace posible obtener complejidad espacial $O(n^3)$, en el peor de los casos, tal y como se demuestra en [56]. Además de obtener esta complejidad espacial, la aproximación seguida en ICE tiene otras ventajas en lo relativo a la generación de la representación sintáctica.

- El empleo de un grafo Y-O permite representar mediante una estructura finita, un conjunto posiblemente infinito de árboles de análisis.
- Las estructuras de datos empleadas en la representación de las relaciones sintácticas son las mismas que se utilizan en el proceso de análisis.
  - Esta correspondencia nos permite establecer una relación directa entre la compartición de cálculos durante el análisis y la compartición del estructuras sintácticas. En efecto, los ítems de ICE juegan el papel de elementos base para el cálculo durante el proceso de análisis. Pero, a la vez, tienen una función estructural, al corresponderse con los nodos del grafo de salida. Así, el empleo de un grafo Y-O permite representar la compartición de estructuras sintácticas como nodos-O, al mismo tiempo que los nodos-Y son referencias a los ítems del analizador dados por las producciones de la gramática de salida.
  - Dado que las transiciones del TP dependen únicamente del primer elemento o de los dos primeros elementos de la cima de la pila, las producciones de la gramática de salida poseen como máximo dos elementos en su parte derecha. Esto tiene como consecuencia que sea posible

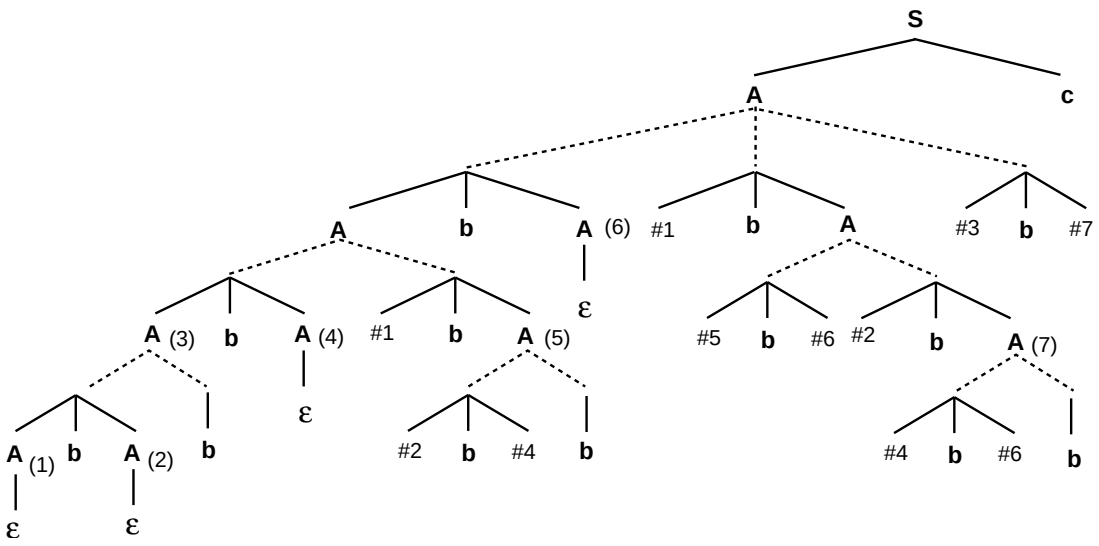


Figura 7.10: Compartición en representaciones arbóreas clásicas.

compartir la cola de la lista de hijos de un subárbol en lugar de la lista completa, aumentando las posibilidades de compartición. En efecto, este tipo de compartición no es posible con una representación clásica no binarizada, puesto que, en ese caso únicamente se podrían compartir subárboles completos, tal y como muestra el ejemplo 7.1.

Ejemplo 7.1.
La figura 7.10 muestra, empleando una representación clásica, el bosque correspondiente al análisis de la cadena $w = bbbc$ empleando una gramática definida por las siguientes reglas:

$$\begin{array}{lcl@{\qquad}lcl} S & \rightarrow & A\,c & A & \rightarrow & b \\ A & \rightarrow & A\,b\,A & A & \rightarrow & \varepsilon \end{array}$$

La figura 7.11 muestra el bosque obtenido por ICE *para el análisis de esa misma cadena de entrada. Como se puede apreciar, la representación empleada por* ICE *permite un mayor grado de compartición. En este caso es posible compartir la cola de la lista de hijos de un subárbol, como la que se muestra sombreada e identificada con el número* (7)*, en la figura 7.11. En la representación clásica no se presenta esta posibilidad, puesto que se exige la compartición de subárboles completos.*

Como hemos visto, las ventajas de la aproximación seguida en ICE en cuanto a la compartición de estructuras, son importantes. Sin embargo, el empleo del autómata LALR(1) para guiar el proceso de análisis en el TP y mejorar la eficiencia, presenta el problema de que puede dar lugar a que la compartición de estructuras sintácticas no sea la ideal. La razón de que, en determinados casos, la compartición no sea la ideal es la inclusión del estado del autómata LALR(1) en los ítems manejados por ICE. Al realizar la comprobación de redundancias basada en la relación de subsumción entre ítems, se tiene en cuenta el estado almacenado para decidir si dos ítems son iguales o no. Es posible que dos ítems representen el reconocimiento del mismo símbolo de la gramática, cubriendo ambos idéntica porción de la cadena de entrada y en un mismo contexto sintáctico, pero que sin embargo se diferencien por el estado que almacenan. De este modo, el estado del autómata LALR(1) puede diferenciar dos ítems que sean sintácticamente equivalentes, haciendo que no sea posible compartirlos en el grafo Y-O.

Vilares [56] muestra ejemplos en los que empleando la misma técnica de tabulación con otros tipos de autómatas como guía, se obtienen mejores resultados en cuanto la compartición de estructuras, aunque a costa de una merma en la eficiencia del proceso de análisis. La conclusión de ese estudio es que el empleo

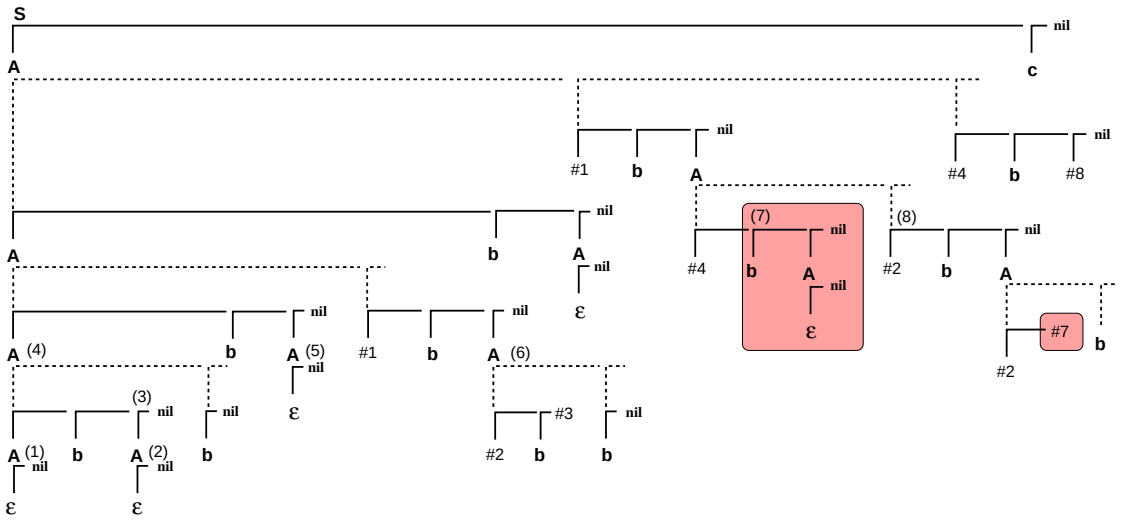


Figura 7.11: Compartición en ICE.

de autómatas LR, en concreto LALR(1), es la que ofrece el mejor compromiso entre eficiencia en el análisis y eficiencia en la representación.

Capítulo 8

Integración con el analizador sintáctico

Una vez que hemos presentado el entorno de análisis sintáctico ICE [56], y descrito el formalismo de representación de estructuras sintácticas que emplea, dedicaremos este capítulo a estudiar la integración del algoritmo de reconocimiento de patrones en árboles de Zhang y Shasha [65] con los analizadores generados por ICE. Dicha integración se centra en dos aspectos. En primer lugar, será necesario adaptar el algoritmo de cálculo de la distancia de edición propuesto por estos autores al tipo de estructuras sintácticas manejadas por ICE, esto es, al caso de bosques de análisis compartidos representados como grafos Y-O. En segundo lugar, se aprovechará la compartición de estructuras sintácticas ofrecida por ICE, para evitar repetir cálculos de distancias que afecten a subestructuras compartidas.

De este modo, comenzaremos el capítulo revisando una serie de generalidades que afectan al modo en que se efectuará la integración. Presentaremos después las modificaciones que es necesario introducir en el algoritmo presentado en el capítulo 5 para manejar los bosques de análisis generados por ICE. Tras revisar dichas modificaciones, nos centraremos en el estudio del uso de la compartición de estructuras sintácticas para ahorrar cálculos durante el proceso de reconocimiento de patrones. El capítulo finaliza con un ejemplo práctico que ilustra el empleo de la compartición de estructuras sintácticas para ahorrar cálculos innecesarios de distancias innecesarios.

8.1. Consideraciones previas

Nuestro objetivo es diseñar un algoritmo de reconocimiento de patrones en bosques compartidos e integrarlo en al analizador sintáctico ICE. Como vimos en el capítulo 7, los bosques compartidos generados por ICE tienen la forma de grafos Y-O. Esto grafos, a su vez, se corresponden con la gramática de salida construida a partir de los ítems del analizador y de las transiciones realizadas durante el proceso de análisis de las cadenas de entrada. En nuestro caso nos limitaremos exclusivamente al manejo de bosques dato compartidos generados por ICE, de tal forma que el árbol patrón sigue siendo un único árbol simple. Lo que pretenderemos será integrar en el proceso de análisis sintáctico un mecanismo que nos permita calcular de forma eficiente las distancias entre el árbol patrón que se recibirá como entrada y el conjunto de árboles de análisis representados de forma compacta en el seno del bosque compartido. De este modo, una vez que esté disponible el árbol patrón, el analizador ICE se encargará de alimentar al algoritmo de comparación con los fragmentos del bosque de análisis que vaya construyendo.

Las aplicaciones de la integración de este tipo de técnicas reconocimiento de patrones y de análisis sintáctico son muy diversas y se examinarán con más detalle en la última parte de esta memoria. No obstante, podemos citar las posibilidades que ofrecen como mecanismo para dar soporte a la indexación y consulta en sistemas de RI o para su uso en entornos de *extracción de información*(EI). El hecho de integrar el análisis sintáctico y el reconocimiento de patrones en paralelo, permitirá definir mecanismos que permitan mejorar la eficiencia del proceso conjunto mediante el empleo de las distancias parciales

que se vayan calculando durante el análisis. Así, sería posible definir heurísticas que permitieran eliminar alternativas de análisis en función de las distancias parciales que tengan asociadas. Del mismo modo se podrán definir heurísticas que permitan abortar un análisis cuando las distancias obtenidas indiquen que su resultado no estará próximo al patrón utilizado.

Como señalábamos, nos limitaremos a estudiar el cálculo de las distancias de un árbol patrón determinista con respecto a un bosque compartido. Si bien podría plantearse la posibilidad de extender las técnicas expuestas en este capítulo al caso más general, esto es, al cálculo de distancias entre pares de bosques compartidos, su coste computacional sería extremadamente alto, mientras que su utilidad sería muy relativa, especialmente en el caso del tipo de aplicaciones citadas anteriormente. En efecto, en dichos entornos de indexación y consulta no nos interesará realizar una comparación entre dos grafos Y-O, sino evaluar la similaridad entre uno o más árboles patrón deterministas y el árbol dato que les sea más próximo de entre todos los que forman parte del bosque compartido considerado. De este modo, pasamos a definir formalmente el tipo de *distancia respecto a bosques compartidos* que vamos a estudiar.

Definición 8.1 (distancia respecto a bosques compartidos)**.**
Dados un árbol patrón P etiquetado y ordenado, y un bosque de análisis compartido D, representado en forma de grafo Y-O*. Si $B_D = \langle \hat{D}_1, \hat{D}_2, \ldots, \hat{D}_n \rangle$ es el conjunto de árboles contenidos en el bosque D, definiremos la* distancia de P respecto al bosque compartido D*, $d(P, D)$, como*

$$d(P, D) = min\{d(P, \hat{D}_i) |\ \hat{D}_i \in B_D\}$$

Intuitivamente, ante un árbol patrón P y un bosque dato D, lo que deseamos obtener es el valor de la distancia entre P y el árbol $\hat{D}$ contenido en el bosque D que sea más próximo al patrón P. El modo más directo de obtener dicha distancia sería el calcular, una a una, todas las posibles distancias entre el patrón P y cada uno de los árboles representados en el bosque compartido y tomar la menor de ellas. Sin embargo, el número de árboles presentes en dicho bosque puede ser muy alto o incluso infinito. Es por ello por lo que es necesario un mecanismo de cálculo de distancias que saque provecho, en la medida de lo posible, de la compartición de estructuras sintácticas que ofrece el bosque compartido para evitar la repetición de cálculos innecesarios.

Esta reutilización de cálculos basada en la compartición de estructuras es un punto clave en nuestra aportación. Para poder llevarla a cabo necesitamos un conocimiento en profundidad de los mecanismos que originan la compartición de estructuras sintácticas. Esto nos permitirá adaptar los algoritmos de reconocimiento de patrones en árboles descritos con anterioridad para el caso de las distancias sobre bosques compartidos. Dado que nuestro trabajo se centra en la integración con el analizador ICE [56], comenzaremos estudiando las peculiaridades que éste ofrece en cuanto a la representación de los bosques compartidos. Por ello, será necesario modificar el algoritmo original de Zhang y Shasha [65] para manejar el tipo de grafos Y-O utilizados en ICE. Una vez modificado el algoritmo original, estudiaremos como aprovechar la compartición que ofrece ICE para ahorrar cálculos.

8.2. Modificaciones del algoritmo básico

La integración de un analizador sintáctico que proporcione al sistema los bosques dato en los cuales localizar los árboles patrón, determina ciertos aspectos que afectan al modo de realizar el proceso de reconocimiento y determinan la eficiencia del sistema en conjunto. En este punto, es necesario tener en cuenta que ambos aspectos, análisis sintáctico y reconocimiento de patrones, están topológicamente relacionados, en particular en lo referente a los mecanismos que provocan la multiplicación y compartición de estructuras sintácticas.

Comenzaremos, entonces, presentando los aspectos del algoritmo de Zhang y Shasha [65] que se ven afectados por las peculiaridades de las representaciones sintácticas utilizadas en ICE [56] como se muestra en [47, 51, 35, 49, 50]. En primer lugar, veremos como la topología de los grafos Y-O de ICE

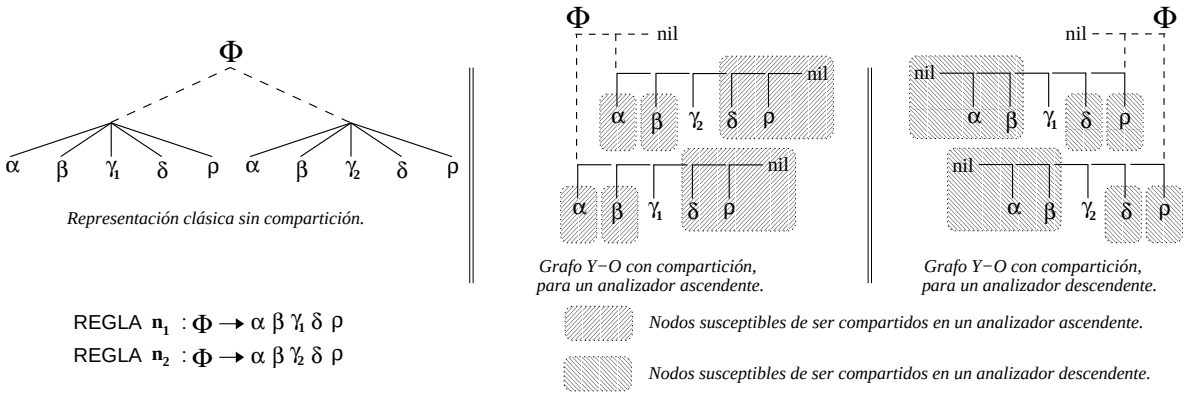


Figura 8.1: Compartición de estructuras en grafos Y-O.

aconsejan cambiar la estructura general del algoritmo original para modificar el orden en el que se realiza el tratamiento de los subbosques inducidos. Los otros aspectos que requieren modificación son los relativos al procesamiento de dos características específicas de ICE, el procesamientos de nodos-O y la propagación de distancias en árboles binarizados.

8.2.1. Cambio en la ordenación de los árboles

Comenzamos exponiendo las causas que nos llevan a modificar la estrategia de procesamiento de los nodos del bosque compartido. Como hemos mencionado, el primer factor determinante al abordar este tipo de integración es el propio tipo de representación sintáctica utilizada. En nuestro caso, los analizadores generados por ICE representan el resultado del análisis mediante un grafo Y-O, como mostramos en el capítulo 7. En este grafo, los nodos-Y se corresponden con los nodos usuales de un árbol de análisis, mientras que los nodos-O representan las ambigüedades. También explicamos como las subestructuras compartidas puede corresponderse con árboles completos, pero también con la compartición de una parte de los descendientes de un nodo, lo que denominábamos compartición de colas de hijos en el capítulo 7, y que muestra la figura 8.1.

El tipo de compartición sobre esta parte de la descendencia depende, a su vez, del tipo de estrategia de análisis considerada. De este modo las técnicas ascendentes provocan la compartición de los constituyentes situados más a la derecha, mientras que las descendentes lo hacen con los situados más a la izquierda, tal y como también se muestra en la misma figura 8.1. Otra característica interesante de ICE es que identifica los nodos del grafo Y-O de salida con los elementos mismos de cálculo del analizador, esto es, los ítems tabulados de ICE.

El algoritmo de Zhang y Shasha considera un postorden transversal en la numeración de los nodos. Para poder sacar provecho de las posibilidades de reutilización de cálculos derivadas de la compartición estructural, en concreto de la compartición de colas de hijos, esta ordenación requeriría de un analizador sintáctico descendente. Como se muestra en segundo esquema de la parte derecha de la figura 8.1, el ordenamiento en postorden permite reutilizar de un modo directo los los cálculos asociados a los nodos de una cola de hijos compartida. En efecto, como muestra el otro esquema de la misma figura, con un ordenamiento en postorden y un analizador ascendente, los nodos de una cola de hijos compartida se encontrarían enmarcados en distintos contextos izquierdos y no sería posible reutilizar las distancias asociadas. Mientras que, con la misma ordenación y un analizador descendente dichos nodos comparten un mismo contexto izquierdo y las distancias resultantes son las mismas en los dos análisis posibles.

Sin embargo, el uso de este tipo de analizadores descendentes supone, en general, una merma de de eficacia en relación a estrategias ascendentes más eficientes computacionalmente, como es el caso de ICE. En consecuencia, se impone una adaptación del algoritmo original para su integración práctica con un analizador sintáctico ascendente. Para hacer esto hemos llegado a la conclusión de que es necesario

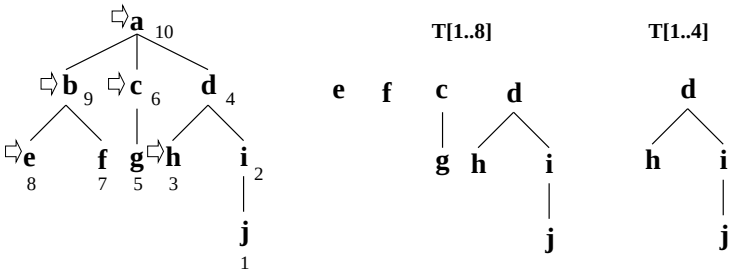


Figura 8.2: Distancia entre bosques usando una numeración en postorden inverso

cambiar la numeración de los nodos, empleando un postorden inverso, que ordena a los hijos de un nodo comenzando por la derecha, en lugar del postorden clásico usado por Zhang y Shasha [65], que ordena de izquierda a derecha.

Definición 8.2 (postorden inverso)**.**
Dado un nodo u de un árbol $T = (V, E, raíz(T))$, se le asigna un índice *o* numeración en postorden inverso*, $post(u)$, de acuerdo a las siguientes reglas:*

- *La numeración en postorden inverso de la hoja más a la derecha de T es* 1*.*
- *Si u es un nodo interno, y p es el mayor índice en preorden inverso asignado a un nodo en $T[u]$*[1]*, entonces $post(u) = p + 1$.*
- *Siendo v la hoja más a la derecha del subárbol con raíz en el siguiente hermano a la izquierda de u, entonces $post(v) = post(u) + 1$.*

Este tipo alternativo de numeración de los nodos es análogo al postorden clásico presentado en el capítulo 2. Al igual que en aquel, se asignan los índices de modo ascendente. Se comienza por la hoja más a la derecha y se va numerando de forma consecutiva a sus hermanos izquierdos antes de numerar a sus padres, hasta terminar asignando un índice al nodo raíz.

Hay otro argumento, relacionado con la topología de los árboles, a favor del empleo de un postorden inverso. Como vimos en el capítulo 7 ICE maneja transiciones canónicas que dependen a lo sumo de dos ítems. Esto hace que las reglas de la gramática de salida que codifica el grafo Y-O contengan, como máximo, dos ítems en su parte derecha. Como consecuencia de este comportamiento, los nodos del grafo resultante tienen como máximo dos hijos. Por otro lado, el empleo como control finito de un autómata ascendente LALR(1) y la binarización implícita de las reglas durante las operaciones de reducción determina la topología de las listas de hijos de un nodo. Dicha binarización y el empleo de derivaciones por la derecha da lugar a que, a medida que avanza el análisis y se reconocen nuevas porciones de la cadena de entrada, los nuevos subárboles hijos construidos se vayan acumulando por el extremo derecho de la lista de constituyentes de su nodo padre.

Suponiendo una ordenación en postorden clásico, esta topología da lugar a que los ítems asociados a los símbolos auxiliares $\nabla_{r,i}$, introducidos durante la binarización, sean *raíces_I*. El problema que surge con este comportamiento es que, a medida que se avanza en el análisis de la cadena, los subárboles asociados a estas *raíces_I* se van haciendo cada vez más grandes y van incluyendo a otras *raíces_I*, también etiquetadas con símbolos $\nabla_{r,i}$, y ya procesadas con anterioridad. Como parece obvio, este comportamiento es poco eficiente. La figura 8.3 ilustra este problema. En ella, el esquema central muestra el grafo Y-O[2] construido por ICE para una cadena de entrada $w = abcd$, empleando la gramática que se

[1] Ese nodo será el hijo más a la izquierda de u.
[2] En este caso no hay nodos-O con más de una alternativa.

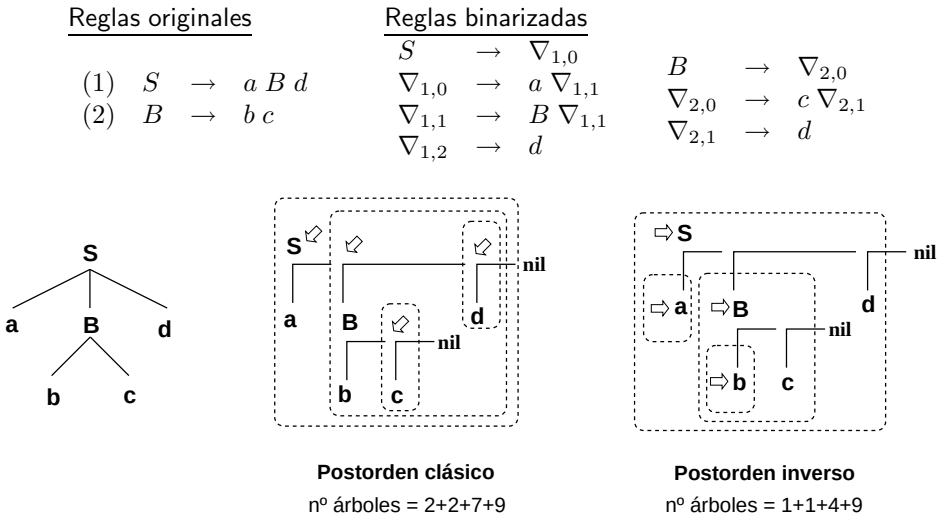


Figura 8.3: Binarización y tipos de postorden.

muestra en esa misma figura. En dicho grafo se señalan las *raíces_I* y sus subárboles asociados. Se indica también el número de bosques inducidos que se pueden crear dentro de las *raíces_I* presentes en el árbol. Además de incrementar el tamaño de las *raíces_I* a procesar, el postorden clásico presenta la desventaja de que no se puede procesar un subárbol *raíz_I* en el momento en que se completa el reconocimiento de sus constituyentes. Empleando esta ordenación, y como consecuencia de la binarización realizada, será necesario esperar hasta que se complete el reconocimiento de la totalidad de la parte derecha de la regla en la cual está incluida dicha *raíz_I*, antes de poder calcular las distancias asociadas a sus descendientes. Sin embargo, como veremos al redefinir el algoritmo de Zhang y Shasha [65] empleando un postorden inverso, esta nueva ordenación si permite realizar el cálculo de distancias de un subárbol sin tener que esperar a completar el reconocimiento de su subárboles hermanos.

Por lo tanto, la posibilidad de compartir colas de hijos y la propia topología de los bosques construidos en base a la binarización implícita de las reglas de la gramática, aconsejan utilizar una numeración en postorden inverso de los nodos de los árboles manejados, tal y como se muestra en la figura 8.2. De este modo, siendo T un árbol, definimos $r(i)$ como el índice de la hoja más a la derecha de las que descienden del subárbol con raíz en $t[i]$. De forma análoga a como se procedía en el algoritmo original, se define el conjunto *raíces_D*(T), indicado mediante flechas en ejemplo de la figura 8.2, como el conjunto de los nodos de un árbol T con un hermano a la derecha, más la raíz, *raiz*(T), de T. La definición formal de este conjunto es simétrica a la de *raíces_I*.

Definición 8.3 (conjunto *raíces_D*(T))**.**
Sea T un árbol, se define el conjunto raíces_D(T) de la siguiente forma:

$$raíces\_D(T) = \{k|\ 1 \leq k \leq |T|\ \ y\ \nexists k' > k\ \ tal\ que\ r(k) = r(k')\}$$

Retomando la figura 8.3, el esquema de la derecha muestra las *raíces_D* y sus subárboles, en el caso de considerar un postorden inverso. Como se puede apreciar el número de subbosques inducidos a procesar es menor. Además presenta la ventaja de que no es necesario esperar a terminar el reconocimiento de la totalidad de las reglas para poder empezar a procesar las *raíces_D* de niveles inferiores. En cuanto se enlaza un nodo con su hermano derecho, se sabe que dicho nodo es una *raíz_D* y puede procederse a calcular sus distancias en ese preciso instante. Lo cual evita el tener que esperar a completar el

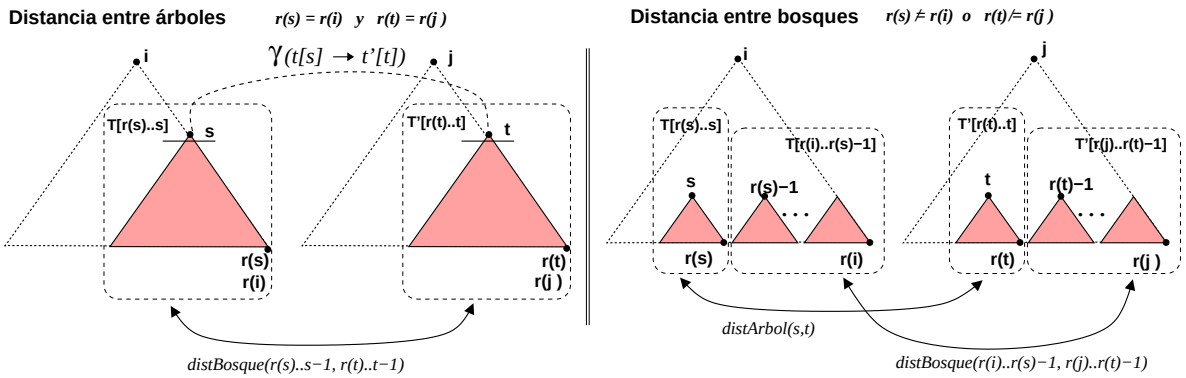


Figura 8.4: La distancia entre bosques en nuestra propuesta.

reconocimiento de los demás nodos hermanos que pertenezcan al lado derecho de la misma regla que ésta $raíz\_D$, y que aun no hubieran sido analizados.

A partir de aquí, la construcción alternativa para la distancia de edición entre bosques, utilizando la nueva ordenación, es análoga a la del algoritmo original. Mostramos a continuación las fórmulas resultantes para el cálculo de distancias en árboles y bosques numerados según el postorden inverso.

$$distBosque(\emptyset, \emptyset) = 0$$
$$distBosque(r(i)..s, \emptyset) = distBosque(r(i)..s-1, \emptyset) + \gamma(t[s] \rightarrow \varepsilon)$$
$$distBosque(\emptyset, r(j)..t) = distBosque(\emptyset, r(j)..t-1) + \gamma(\varepsilon \rightarrow t'[t])$$

$$distBosque(r(i)..s, r(j)..t) = \begin{cases} \min \left\{ \begin{array}{lcl} distBosque(r(i)..s-1, r(j)..t) & + & \gamma(p[s] \rightarrow \varepsilon), \\ distBosque(r(i)..s, r(j)..t-1) & + & \gamma(\varepsilon \rightarrow d[t]), \\ distBosque(r(i)..s-1, r(j)..t-1) & + & \gamma(p[s] \rightarrow d[t]) \end{array} \right\} \\ \textbf{sii } r(s) = r(i) \textbf{ y } r(t) = r(j) \\ \\ \min \left\{ \begin{array}{lcl} distBosque((i)..s-1, r(j)..t) & + & \gamma(p[s] \rightarrow \varepsilon), \\ distBosque(r(i)..s, r(j)..t-1) & + & \gamma(\varepsilon \rightarrow d[t]), \\ distBosque(r(i)..r(s)-1, r(j)..r(t)-1) & + & distArbol(s,t) \end{array} \right\} \\ \textbf{en otro caso} \end{cases}$$

De nuevo, para calcular $distArbol(P, D)$ será suficiente tener en cuenta que

$$distArbol(P, D) = distBosque(r(raíz(P))..raíz(P), r(raíz(D))..raíz(D))$$

En cuanto a los límites espaciales y temporales del mecanismo de cálculo de distancias de edición empleando un postorden inverso, estos son los mismos que los del algoritmo original de Zhang y Shasha. Esto es, dados dos árboles T y T', la complejidad temporal del algoritmo de cálculo de distancias de edición sigue siendo $O(|T| \times |T'| \times min\{prof(T),\ hojas(T)\} \times min\{prof(T'),\ hojas(T')\})$, donde $prof$ es la profundidad máxima del árbol y $hojas$ el número de hojas, y la complejidad espacial es $O(|T| \times |T'|)$, como en el algoritmo original. El razonamiento en este caso es simétrico al realizado en la sección 5.2.3, simplemente es necesario tener en cuenta la nueva ordenación y, en base a ella, adaptar las definiciones y resultados que empleábamos en dicha sección.

8.2.2. Distancias en presencia de nodos-o

Revisaremos ahora como debemos adaptar el comportamiento del algoritmo del Zhang y Shasha [65] para manejar grafos con nodos-o. La presencia de nodos-o en un bosque compartido tiene una serie

de implicaciones importantes en lo referente al cálculo de las distancias de edición. En primer lugar, la existencia en un bosque de un nodo-O con dos o más alternativas dará lugar a que para determinados nodos localizados después de dicho nodo-O, sea necesario manejar múltiples valores simultáneos de distancias con respecto a bosques del árbol patrón. En concreto, cada una de las distancias que se calculen para los nodos procesados después de un nodo-O va a tener tantos valores posibles como alternativas tenga dicho nodo. En el caso de que en un mismo bosque haya más de un nodo-O, el número de distancias a manejar simultáneamente se iría multiplicando a medida que surgieran nuevas alternativas.

La segunda particularidad, es que la presencia de un nodo-O puede dar lugar a la compartición de subestructuras comunes entre los descendientes de las distintas alternativas de dicho nodo. Como veremos más tarde, esta compartición permitirá evitar en ciertos casos la repetición de cálculos de distancias redundantes. Por el momento nos centraremos en cómo manejar de forma eficiente los nodos-O y cómo dar soporte a distancias con múltiples valores simultáneos dentro del marco del algoritmo original.

Como mencionábamos, la existencia de un nodo-O en un grafo implica que este representa a un bosque formado por tantos árboles distintos como alternativas tenga dicho nodo. Esto nos lleva a que la numeración en postorden inverso que utilizábamos para identificar unívocamente a los nodos de un árbol deje de ser válida. En efecto, al incluir nodos-O un mismo nodo tendrá tantas posiciones en postorden inverso distintas como el número de árboles diferentes en los que esté incluido. Por lo tanto será necesario definir un nuevo tipo de numeración en postorden para el caso de grafos Y-O.

Definición 8.4 (postorden inverso en grafos Y-O)**.**
Dado un grafo Y-O D *que contiene al conjunto de árboles* $B_D = \langle \hat{D}_1, \hat{D}_2, \ldots, \hat{D}_n \rangle$*, y dado un nodo* u *perteneciente a dicho grafo, definiremos la numeración en postorden inverso de* u*,* $post_{\text{Y-O}}(u)$*, del siguiente modo:*

$$post_{\text{Y-O}}(u) = \{i |\ i = post(u)\ en\ \hat{D}_j\}\ \forall \hat{D}_j \in B_D\ tales\ que\ u \in \hat{D}_j$$

De ahora en adelante emplearemos $post(u)$ en lugar de $post_{\text{Y-O}}(u)$ cuando no haya ambigüedad. Intuitivamente, $post_{\text{Y-O}}(u)$ se corresponde con el conjunto de todos los índices que tiene el nodo u en los distintos árboles incluidos en el seno del grafo D. Seguiremos empleando la notación utilizada en los capítulos anteriores, pero con la salvedad de que ahora un nodo puede tener asociado más de un índice en postorden inverso. Así, en un grafo Y-O D, $d[i]$ representa al nodo en la posición i-ésima de algún árbol de D, que a su vez puede ser el nodo en una posición j diferente en otro de los árboles incluidos en D. En el caso de la notación $D[i..s]$, como ambos nodos i y s pueden pertenecer a varios árboles, extenderemos dicha notación para que represente al bosque Y-O comprendido entre ambos nodos. Es decir, $D[i..s]$ representa al conjunto de todos los subbosques incluidos en D que comienzan en el nodo i-ésimo de un árbol de D dado y terminan en el nodo s de ese mismo árbol.

Con estas ideas en mente podemos definir, para el caso de bosques Y-O, los conceptos relacionados con árboles que fueron introducidos en el capítulo5. En nuestro caso, dado un grafo Y-O, D, y un nodo $d[j]$, la notación $r(j)$ representará a la hoja u hojas más a la derecha de entre las que descienden del nodo $d[j]$ en todos los árboles en los que está incluido dicho nodo. El conjunto $raíces\_D(D)$ contendrá a todos los nodos que son $raíz\_D$ en alguno de los árboles de D. Más concretamente, $raíces\_D(D)$ contiene a las raíces de los árboles presentes en D, así como, a todos los nodos que en algún árbol de D tengan hermanos por la derecha. Para el caso de la distancia $distBosque(P[r(i)..s], D[r(j)..t])$ debemos tener en cuenta que el bosque $D[j..t]$ puede estar representando en realidad a un conjunto de bosques. Así, partiendo de la definición de distancia respecto a compartidos presentada en la sección 8.1, tenemos la siguiente definición de *distBosque*.

Definición 8.5 (*distBosque* respecto a bosques compartidos)**.**
Siendo P *un árbol patrón etiquetado y ordenado y* D *un grafo* Y-O*. Dados los nodos* $s \in P$ *y* $t \in D$ *y las* $raíces\_D$ $i \in anc(s)$ *en* P *y* $j \in anc(t)$ *en* D*, la distancia* $distBosque(P[r(i)..s], D[r(j)..t])$ *se define*

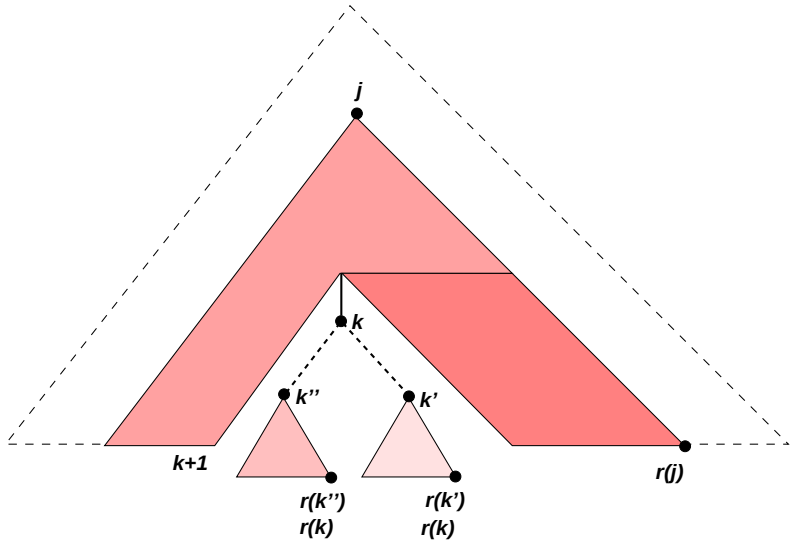


Figura 8.5: Cálculo de distancias con nodos-o.

como:

$$distBosque(P[r(i)..s], D[r(j)..t]) = min\{distBosque(P[r(i)..s], \hat{D}[r(j)..t])|\ \hat{D}[r(j)..t] \in D[r(j)..t]\}$$

Lo que deseamos calcular en este caso es la distancia entre el bosque patrón $P[r(i)..s]$ y el bosque contenido en $D[r(j)..t]$ que sea más próximo a ese patrón. La forma más simple de hacerlo sería extraer del grafo $D[r(j)..t]$ todos los bosques que contiene, evaluar una a una sus distancias respecto al bosque $P[r(i)..s]$ y tomar el menor de dichos valores. Sin embargo, veremos como podemos aprovechar las estructuras compartidas presentes en $D[r(j)..t]$ para proponer un mecanismo de cálculo menos costoso. El primer paso consistirá en definir cómo obtener el valor de la distancia en el caso de un bosque o un árbol cuyo último nodo sea un nodo-o.

Lema 8.1.
Sean P un árbol patrón y D un grafo y-o*. Dados un nodo $i \in raíces\_D(P)$, un nodo $s \in desc(i)$ y un nodo $j \in raíces\_D(D)$. Si tenemos un nodo-*o *$k \in desc(j)$ con dos alternativas numeradas k' y k'', se verifica que:*

$$distBosque(P[r(i)..s], D[r(j)..k]) = min\{\ distBosque(P[r(i)..s], D[r(j)..k']),\\ distBosque(P[r(i)..s], D[r(j)..k'])\}$$

En el caso de que k tuviera n alternativas, $k'_1, k'_2, \ldots, k'_n$; la expresión resultante sería $distBosque(P[r(i)..s], D[r(j)..k]) = min_{1 \leq i \leq n}\{distBosque(P[r(i)..s], D[r(j)..k'_i])\}$.

Demostración.
El resultado de este lema se deriva directamente de la definición de distancia respecto a bosques compartidos que estamos manejando. En este caso, en el bosque $D[r(j)..k]$ hay dos bosques distintos, $D[r(i)..k']$ y $D[r(j)..k'']$. Aplicando la definición 8.5 de *distBosque* respecto a bosques compartidos tendremos que tomar el bosque $\hat{D}[r(j)..t]$ más cercano a $P[r(i)..s]$ de entre los incluidos en $D[r(j)..t]$, con lo que resulta la expresión anterior. □

El lema anterior indica que para obtener la distancia en un nodo-o hay que tomar de entre sus alternativas aquella que ofrezca la menor distancia. En el caso de que las alternativas incluyeran a otros nodos-o, estos serían tratados de igual modo durante el cálculo de las distancias que afectasen a las distintas

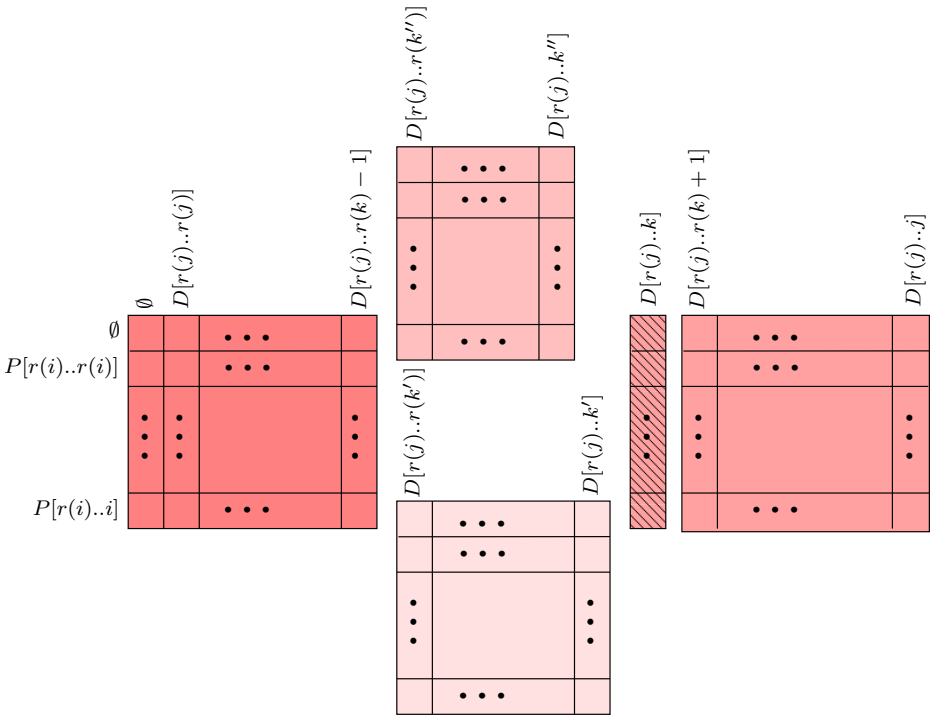


Figura 8.6: Compartición de distancias en los nodos-o.

alternativas. La consecuencia más importante de este lema es que permite convertir un conjunto de valores de distancias en un valor único al procesar los nodos-o. Esta propiedad será una de las claves para poder llevar a cabo la reutilización de distancias.

La figura 8.5 muestra la estructura general de un árbol que incluye a un nodo-o. Dicho esquema se corresponde con el caso de compartición del contexto que mostramos en la sección 7.3.2. En este caso tenemos que el nodo j es una *raíz_D* y que el nodo k es un nodo-o con dos alternativas, cuyas respectivas raíces son k' y k''. La región comprendida entre el nodo $r(j)$ y el nodo $r(k)-1$, inmediatamente anterior al nodo-o k, representa al contexto derecho dentro del cual se enmarca cada una de las dos alternativas del nodo-o, k' y k''. Dado que consideramos un postorden inverso, las distancias calculadas para ese subbosque son comunes a los dos subárboles y no es necesario evaluarlas dos veces. Las distancias que incluyan a nodos pertenecientes a los dos subárboles, k' y k'', sí deben calcularse de forma separada, puesto que se corresponden con bosques diferentes. Una vez que se completa el procesamiento de los subárboles de las dos alternativas, se puede aplicar el lema 8.1 para obtener valores únicos para las distancias que impliquen al nodo-o k. En una primera aproximación, las distancias asociadas a los nodos situados después de k deberán tener en cuenta los distintos valores calculados para las diferentes alternativas de k. Sin embargo, el lema 8.1 permite transformar esos múltiples valores en una única distancia que se asocia al nodo-o k. De este modo, los cálculos con bosques que terminen en nodos con un índice mayor que k tendrán que consultar un único valor, no siendo necesario, por lo tanto, duplicar cálculos a causa de las alternativas del nodo-o. Conseguimos con esto, simplificar la obtención de estas distancias, que deberán ser calculadas una única vez, considerando en dicho cálculo sólo una de las alternativas del nodo-o, aquella que ofreciera la menor distancia.

La figura 8.6 ilustra la compartición del contexto anterior y posterior a un nodo-o. En esta figura se representan la compartición de las tablas de distancias *distBosque* para un árbol como el mostrado en la figura 8.5, con un único nodo-o, k, que tiene dos alternativas, k' yk''. Como se puede apreciar, no se evalúan dos veces las distancias que involucran a nodos situados antes de las dos alternativas del

nodo k. Esto es, las distancias para los bosques entre el nodo j y el nodo $r(k)-1$. Dichos bosques y sus distancias asociadas son comunes para las dos alternativas de k y basta con calcularlas una vez. En el procesamiento de las dos alternativas sí es necesario realizar dos cálculos distintos, puesto que, aunque el contexto derecho sea el mismo, los dos bosques resultantes son distintos. Una vez procesadas las dos alternativas, se extraen los mejores valores que darán lugar a las distancias asociadas al nodo-O, k. Dichas distancias se almacenan en la columna rayada, etiquetada como $D[r(j)..k]$ en la figura 8.6. Puesto que los mejores valores de las dos alternativas se han asociado al nodo-O k, el procesamiento para los demás nodos, entre $k+1$ y j, se puede realizar una única vez. Al procesar estos bosques simplemente se toman los valores de la columna $D[r(j)..k]$ que sean necesarios. De esta forma, se asegura que en cada momento se considera la alternativa que origina la distancia más próxima al correspondiente subbosque patrón.

8.2.3. Distancias en árboles binarizados

Una vez mostrado el manejo de los nodos-O presentes en el grafo, es necesario definir cómo será el comportamiento del algoritmo de cálculo de distancias al tratar con los nodos auxiliares que surgen como resultado de la binarización implícita de las reglas de la gramática de entrada.

Recordemos que, para mantener la complejidad espacial y temporal $O(n^3)$, siendo n la longitud de la cadena de entrada, ICE realiza una binarización implícita de las reglas de la gramática, como explicamos en el capítulo 7. Esta binarización, junto con el uso de transiciones que consideren como máximo dos ítems, determina la topología de los grafos Y-O generados por ICE, como también mostramos anteriormente. Para realizar la binarización implícita se se hace uso de un conjunto de nuevos símbolos de pila, los símbolos $\nabla_{r,i}$. Cada uno de estos símbolos representa el reconocimiento de las i primeras categorías del lado derecho de la regla número r, cuando $i > 0$, o que aún no se ha iniciado el reconocimiento de la regla r, en el caso $i = 0$. De este modo, la reducción de una producción $A_{r,0} \rightarrow A_{r,1}\, A_{r,2} \ldots A_{r,n_r}$, se transforma en la secuencia de reducciones:

$$
\begin{aligned}
&A_{r,0} \rightarrow A_{r,1}\, \nabla_{r,1}\\
&\nabla_{r,1} \rightarrow A_{r,2}\, \nabla_{r,2}\\
&\vdots\\
&\nabla_{r,n_r-1} \rightarrow A_{r,n_r}\, \nabla_{r,n_r}\\
&\nabla_{r,n_r} \rightarrow \varepsilon
\end{aligned}
$$

Antes de comenzar a exponer la problemática derivada del proceso de binarización, revisaremos una serie de propiedades que se derivan de la utilización de este tipo de árboles binarios y de la ordenación en postorden inverso. Siendo D un grafo Y-O binarizado y $d[t]$ uno de sus nodos, se verifica lo siguiente:

- El hermano a la derecha de $d[t]$ es el nodo numerado como $r(t)-1$.
- El hijo izquierdo de $d[t]$ está numerado como $t-1$. Si el nodo $d[t]$ tiene asociado un símbolo auxiliar $\nabla_{r,i}$ siempre se verificará que, de existir, su hijo izquierdo $d[t-1]$ estará etiquetado con un símbolo terminal o no terminal de la gramática.
- El hijo derecho de $d[t]$ es el nodo numerado como $r(t-1)-1$. Si $d[t]$ está etiquetado con un símbolo auxiliar $\nabla_{r,i}$, su hijo derecho $d[r(t-1)-1]$, si este existe, también tiene asociado uno de esos símbolos auxiliares.

Si empleáramos directamente los cálculos alternativos para distancias en postorden inverso de la sección 8.2.1 sobre el grafo Y-O, sin tener en cuenta la presencia de estos símbolos auxiliares, los valores obtenidos serían distintos de los que resultarían si se empleara una representación clásica de esos mismos árboles. Será necesario, por lo tanto, modificar el método de cálculo para que la inclusión de estos símbolos de pila auxiliares en los ítems que forman el grafo Y-O no afecte a la distancia final resultante.

Lo primero que debemos definir es como será el coste de las operaciones de edición realizadas sobre estos símbolos. Como deseamos que la distancia final no se vea afectada, deberemos asignar un coste cero

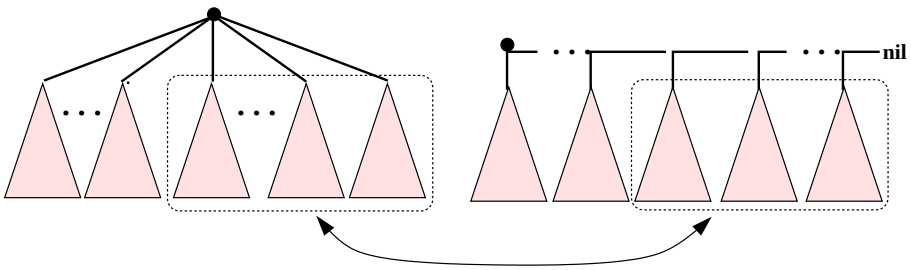


Figura 8.7: Distancias entre bosques binarizados.

a cualquier operación de edición en la que esté involucrado uno de estos símbolos. Así, añadiremos una restricción adicional sobre la función de coste γ, de modo que:

$$\gamma(a \rightarrow b) = 0 \text{ si } a = \nabla_{r,i} \text{ ó } b = \nabla_{r,i}, \forall \text{ símbolo auxiliar } \nabla_{r,i}$$

Si bien esta modificación nos permite aplicar libremente operaciones de edición sobre estos símbolos sin afectar a las distancias, no asegura que la distancia final resultante no se vea perturbada. En efecto, debemos tener en cuenta el significado de los símbolos $\nabla_{r,i}$ para poder modificar el modo en que se calculan las distancias. Los símbolos $\nabla_{r,i}$ representan el reconocimiento parcial de la parte derecha de una regla. Este reconocimiento parcial puede interpretarse en el árbol de análisis correspondiente como la representación de un fragmento del bosque formado por parte de los descendientes del lado derecho de dicha regla. Sin embargo, como consecuencia de la ordenación en postorden inverso, los nodos etiquetados con símbolos $\nabla_{r,i}$ estarán siempre involucrados en el cálculo de distancias entre árboles. Esto es así por el modo en que se realiza la binarización y por como se construyen los grafos Y-O en ICE. Tenemos que los nodos $\nabla_{r,i}$ serán siempre los hijos más a la derecha de sus padres y, en consecuencia, siempre compartirán su hoja más a la derecha con todas las *raíces_D* de las que desciendan.

Por lo tanto, los símbolos $\nabla_{r,i}$ se asocian a bosques parciales, pero para el algoritmo de cálculo estarán implicados en distancias entre subárboles. Será necesario, entonces, modificar las fórmulas que afectan a este tipo de nodos para que reflejen realmente su verdadero significado y permitan obtener unos valores de distancias correctos. Si nos fijamos en el modo en que se realiza la binarización, el hijo izquierdo de un símbolo $\nabla_{r,i}$ nunca podrá ser uno de estos símbolos auxiliares, sino que se corresponderá con el último subárbol del bosque parcial que representa dicho símbolo auxiliar. El hijo derecho, de existir, sí estará etiquetado con otro símbolo $\nabla_{r,i}$, y representará al resto de componentes del bosque parcial asociado a su padre. De este modo, podemos definir la distancia de edición en presencia de los símbolos auxiliares $\nabla_{r,i}$.

Definición 8.6 (distancias con símbolos $\nabla_{r,i}$)**.**
Siendo $i \in P$ y $j \in D$ raíces_D y dados los nodos $s \in desc(i)$ y $t \in desc(j)$, las distancias que impliquen a nodos etiquetados con símbolos $\nabla_{r,i}$ se definirán del siguiente modo:

$$distBosque(P[r(i)..s], D[r(j)..t]) := \begin{cases} distBosque(P[r(i)..s-1], D[r(j)..t]) & \textbf{si } p[s] = \nabla_{r,i} \\ distBosque(P[r(i)..s], D[r(j)..t-1]) & \textbf{si } d[t] = \nabla_{r,i} \end{cases}$$
$$\forall \textit{ símbolo auxiliar } \nabla_{r,i}$$

Intuitivamente, lo que se hace en estos casos, tal y como se muestra en la figura 8.7 es recuperar las distancias asociadas al hijo derecho del símbolo $\nabla_{r,i}$ en el árbol patrón o en el árbol dato, según corresponda. Dicho hijo se corresponde con el último nodo del subbosque parcial que representa $\nabla_{r,i}$. De este modo, la distancia que asociemos al nodo $\nabla_{r,i}$, será realmente una distancia entre bosques. Además se garantiza que dicha distancia entre bosques será la misma que la que se obtendría sobre el correspondiente árbol no binarizado. Para ello simplemente basta con tener en cuenta que todos los símbolos $\nabla_{r,i}$ de niveles

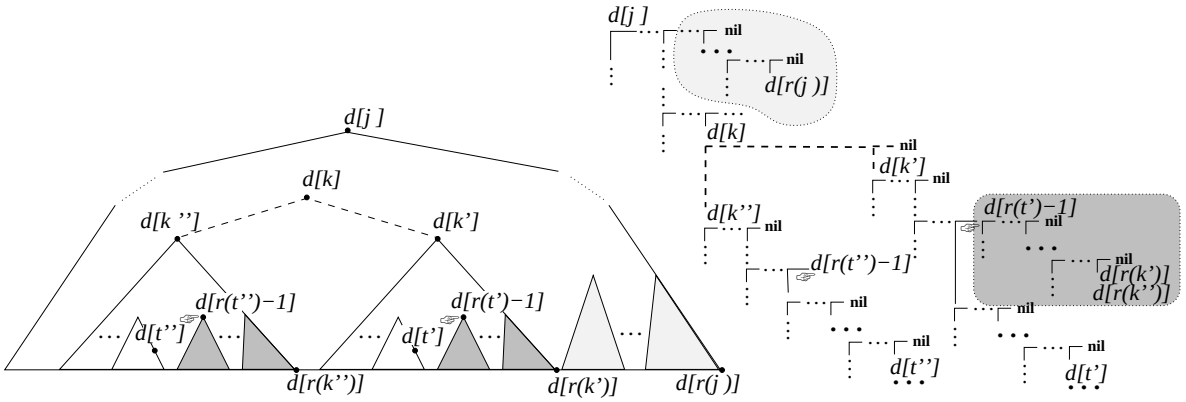


Figura 8.8: Compartiendo en una misma *raíz_D*.

inferiores que hubieran sido introducidos en el proceso de binarización del árbol habrían sido tratados de este mismo modo. Resultando, entonces, que todos los cálculos anteriores habrían empleado y propagado valores de distancias entre bosques parciales idénticos a los que se habrían obtenido en caso de haber manejado los árboles no binarizados correspondientes.

8.3. Reutilización de cálculos en bosques compartidos

Ofrecemos ahora una explicación que muestra como ambos entornos, de análisis sintáctico y de reconocimiento aproximado de patrones, pueden ser integrados de forma eficaz, y que complementa a las presentadas en [47, 51, 35, 49, 50]. Como hemos hecho anteriormente, la aproximación que seguimos se basa en la utilización de las estructuras sintácticas compartidas generadas por nuestro analizador, para evitar la repetición cálculos de distancias redundantes. En este caso se trata de una estudio de unos tipos de compartición estructural más complejos que los descritos en secciones anteriores.

Para comenzar, sea P un árbol ordenado y etiquetado, y sea D un grafo Y-O, ambos construidos conforme al formalismo de representación empleado en nuestro entorno de análisis sintáctico ascendente. Identificaremos a P con una interrogación y a D con una parte de la representación sintáctica de una cadena de entrada con cierto grado de ambigüedad. Como en secciones previas, nuestro objetivo será obtener el valor de la distancia entre P y el árbol contenido en D que sea más próximo. Presentaremos el modo en el que calculamos la distancia entre un árbol patrón y el conjunto de árboles dato representados en el grafo Y-O, así como la forma de mejorar el rendimiento computacional sobre la base de la compartición de estructuras presentes en el resultado del análisis sintáctico.

Sea $p[s]$ el nodo actual en el postorden inverso de P y sea $i \in \text{anc}(s)$ una *raíz_D*. Dado un nodo-O en D podemos distinguir dos casos, dependiendo de la situación de dicho nodo-O en relación a las *raíces_D* de D: la *compartición dentro de una misma raíz_D* y la *compartición entre distintas raíces_D*.

8.3.1. Compartición dentro de una misma *raíz_D*

Esta primera posibilidad es una extensión del mecanismo de compartición de las distancias asociadas a los contextos de las alternativas de los nodos-O que se muestra en la sección 8.2.2. En este caso tenemos una situación en la cual las alternativas de un nodo-O comparten una subestructura común. Veremos como se puede utilizar dicha compartición de estructuras para evitar la duplicación del cálculo de determinadas distancias.

Esta situación se muestra en la figura 8.8, usando una representación clásica y un grafo Y-O. Tenemos que $d[t']$ y $d[t'']$ son los nodos tratados en paralelo sobre dos ramas alternativas del nodo-O $d[k]$, identificadas como $d[k']$ y $d[k'']$. Tenemos que $j \in anc(t') \cap anc(t'')$, esto es, el subárbol con raíz en

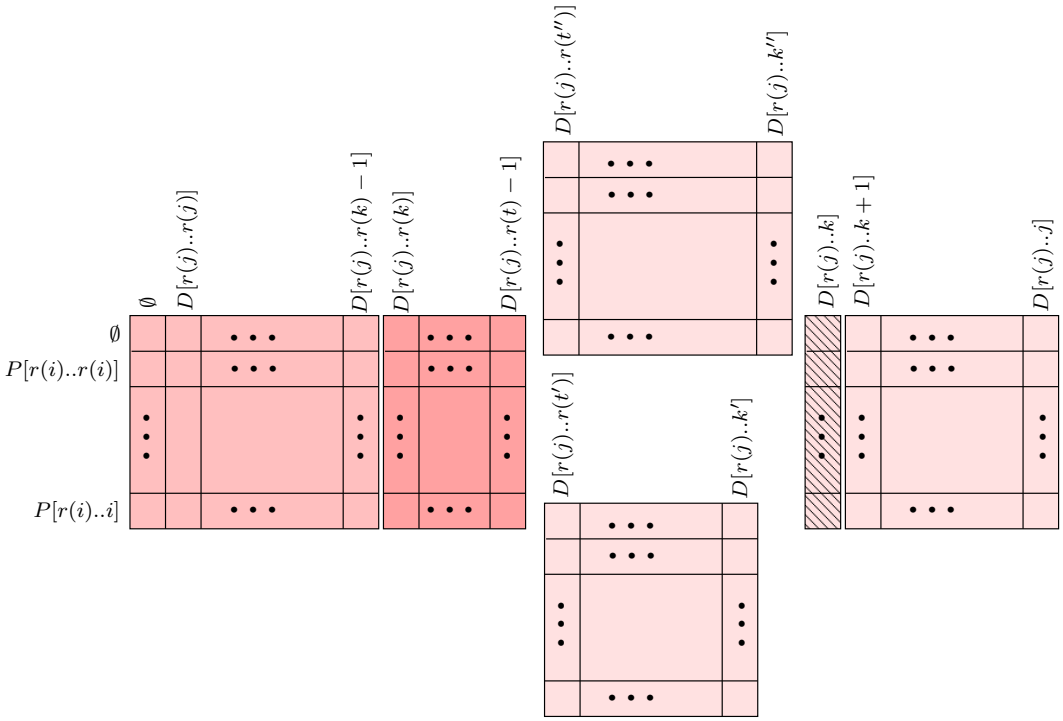


Figura 8.9: Compartición de tablas de distancias en una misma *raíz_D*.

la *raíz_D* $d[j]$ incluye a las dos alternativas del nodo-O, $d[k']$ y $d[k'']$. En esta figura la parte suavemente sombreada se refiere a los nodos cuyas distancias han sido calculadas siguiendo el postorden inverso, antes del nodo-O $d[k]$, y que forman el contexto derecho de sus dos alternativas. La parte fuertemente sombreada representa a la estructura compartida por la dos alternativas del nodo-O. La notación "• • •" en el grafo Y-O expresa el descenso a lo largo de la rama más a la derecha del correspondiente subárbol.

Asumiremos que los nodos $d[r(t') - 1]$ y $d[r(t'') - 1]$ son el mismo, esto es, sus correspondientes subárboles son compartidos. Así, supondremos que $d[r(t')]$ (resp. $d[r(t'')]$) es el siguiente nodo en $d[k']$ (resp. $d[k'']$) a ser tratado, una vez que la distancia sobre la estructura compartida haya sido calculada.

En este punto, nuestro objetivo es calcular los valores para $distBosque(r(i)..s, r(j)..\hat{t})$, $\hat{t} \in \{t',\ t''\}$, probando que podemos traducir la compartición de estructuras durante el análisis sintáctico en compartición de cálculos para estas distancias. En este caso tenemos que $r(j) \neq r(\hat{t})$, puesto que asumimos que existe una región compartida en el bosque de análisis entre $d[r(\hat{t})]$ y $d[r(j)]$. Formalmente, los valores para $distBosque(r(i)..s, r(j)..\hat{t})$, $\hat{t} \in \{t',\ t''\}$ vienen dados por las siguientes fórmulas:

$$distBosque(r(i)..s, r(j)..\hat{t})) = \min \left\{ \begin{array}{lcl} distBosque(r(i)..s-1, r(j)..\hat{t}) & + & \gamma(p[s] \rightarrow \varepsilon), \\ distBosque(r(i)..s, r(j)..\hat{t}-1) & + & \gamma(\varepsilon \rightarrow d[\hat{t}]), \\ distBosque(r(i)..r(s)-1, r(j)..r(\hat{t})-1) & + & distArbol(s, \hat{t}) \end{array} \right\}$$

Así, centrándonos en este cálculo de distancias, podemos pasar a estudiar como se ven afectadas las fórmulas anteriores por el tipo de compartición mostrada en la figura 8.8

1. Los valores $distBosque(r(i)..s-1, r(j)..\hat{t})$, $\hat{t} \in \{t',\ t''\}$ han sido calculados por el algoritmo en un paso previo. De este modo, la compartición proporcionada por el analizador sintáctico no tiene consecuencias en el cálculo de las distancias actuales.

2. Respecto al segundo caso, son posibles dos situaciones en relación a la naturaleza de los nodos $d[\hat{t}]$, $\hat{t} \in \{t',\ t''\}$, éstas son:

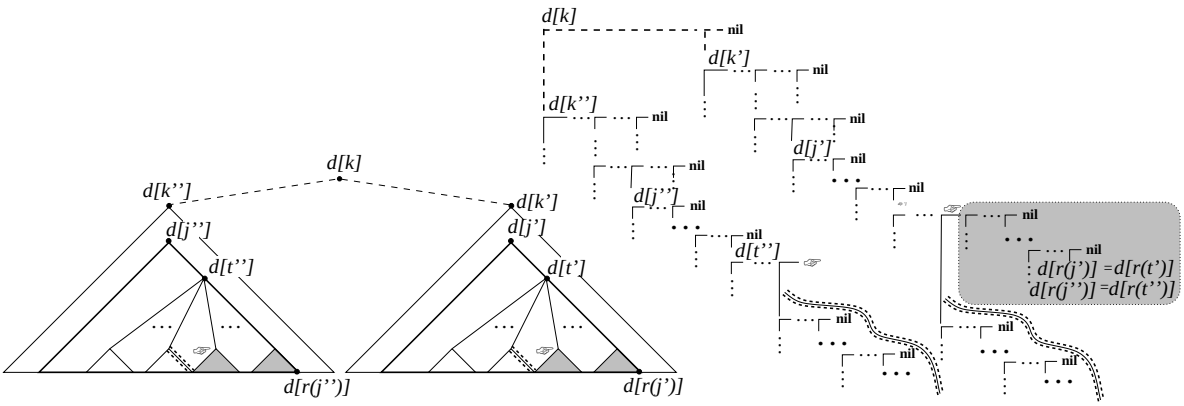


Figura 8.10: Compartición entre distintas *raíces_D* (primer caso).

- Si ambos nodos son hojas, entonces $r(\hat{t}) = \hat{t}$. En consecuencia tenemos que

$$d[t' - 1] = \quad d[r(t') - 1] = \quad d[r(t'') - 1] = \quad d[t'' - 1]$$

  puesto que $r(t')-$ y $r(t'')-$ se refieren al mismo nodo. Por lo tanto, para $\hat{t} \in \{t',\ t''\}$ los valores $distBosque(r(i)..s, r(j)..\hat{t} - 1) = distBosque(r(i)..s, r(j)..r(\hat{t}) - 1)$, son también los mismos.

- De otro modo, siguiendo el postorden inverso, en pasos previos del algoritmo se habrían calculado las distancias que afectasen a las hojas más a la derecha en las derivaciones de $d[t']$ y $d[t'']$, donde se podría aplicar el razonamiento del caso anterior.

3. Los valores para las distancias $distBosque(r(i)..r(s) - 1, r(j)..r(\hat{t})) - 1)$, $\hat{t} \in \{t',\ t''\}$ son idénticos, dado que los nodos $d[r(\hat{t}) - 1]$, $\hat{t} \in \{t',\ t''\}$ son compartidos por el analizador sintáctico.

Finalmente, la figura 8.9 refleja el razonamiento realizado para esta situación y muestra como es la compartición de las tablas de distancias *distBosque*. Se puede apreciar como este tipo de compartición de cálculos es una extensión del mostrado en la figura 8.5, con el añadido de que en este caso las alternativas del nodo-o comparten una subestructura común. Podemos ver como, si esta estructura compartida está situada en el extremo más a la derecha de ambas alternativas, es posible reutilizar las distancias asociadas a esos nodos compartidos sin tener que calcularlas dos veces. También se puede apreciar que, en el caso de que la estructura compartida no estuviera situada a la derecha de ambas alternativas, no será posible reutilizar las distancias calculadas para esos nodos compartidos, puesto que sus respectivos contextos derechos serían distintos.

8.3.2. Compartición entre distintas *raíces_D*

El otro tipo de compartición posible, la compartición entre distintas *raíces_D*, es menos restrictivo. En este caso, tenemos dos *raíces_D*, $j' \in anc(t')$ y $j'' \in anc(t'')$, donde $j' \neq j''$. Tenemos también un nodo-o $d[k]$ que es ancestro común de ambas *raíces_D*. Supondremos que estas dos *raíces_D* se encuentran en ramas de análisis diferentes, esto es, existe una *raíz_D*, $d[j']$ (resp. $d[j'']$), en la rama etiquetada $d[k']$ (resp. $d[k'']$).

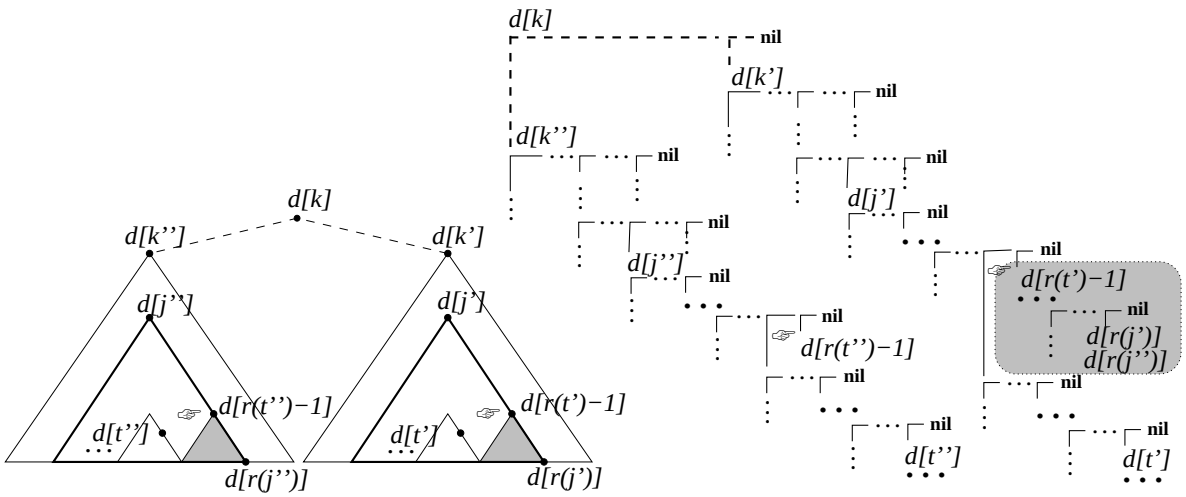


Figura 8.11: Compartición entre distintas *raíces_D* (segundo caso).

Nuestro objetivo será calcular los valores para las distancias $distBosque(r(i)..s, r(\hat{\jmath})..\hat{t})$, donde los pares $(\hat{\jmath}, \hat{t})$ están en $\{(j',\ t'),\ (j'', t'')\}$. Estos valores vienen dados por los siguientes conjuntos de fórmulas:

$$distBosque(r(i)..s, r(\hat{\jmath})..\hat{t}) = \begin{cases} \min \left\{ \begin{array}{lcl} distBosque(r(i)..s-1, r(\hat{\jmath})..\hat{t}) & + & \gamma(p[s] \to \varepsilon), \\ distBosque(r(i)..s, r(\hat{\jmath})..\hat{t}-1) & + & \gamma(\varepsilon \to d[\hat{t}]), \\ distBosque(r(i)..s-1, r(\hat{\jmath})..\hat{t}-1) & + & \gamma(p[s] \to d[\hat{t}]) \end{array} \right\} \\ \textbf{sii } r(s) = r(i) \textbf{ y } r(\hat{t}) = r(\hat{\jmath}) \\ \\ \min \left\{ \begin{array}{lcl} distBosque(r(i)..s-1, r(\hat{\jmath})..\hat{t}) & + & \gamma(p[s] \to \varepsilon), \\ distBosque(r(i)..s, r(\hat{\jmath})..\hat{t}-1) & + & \gamma(\varepsilon \to d[\hat{t}]), \\ distBosque(r(i)..r(s)-1, r(\hat{\jmath})..r(\hat{t})-1) & + & distArbol(s, \hat{t}) \end{array} \right\} \\ \textbf{en otro caso} \end{cases}$$

Esta situación, tal y como se muestra en la figura 8.10, hace posible $r(s) = r(i)$ y $r(\hat{t}) = r(\hat{\jmath})$. En este primer caso, tenemos que los nodos $d[\hat{t}]$ que estamos procesando comparten su hoja más a la derecha con las dos *raíces_D* $d[\hat{\jmath}]$. Además, asumimos que que en ambas *raíces_D*, una parte de la descendencia de $d[t']$ y $d[t'']$, situada en su extremo derecho, se corresponde con una estructura compartida que es común a ambos nodos. Podemos también asumir que estos constituyentes compartidos son un subconjunto propio de la descendencia de dichos nodos dado que, de otro modo, nuestro analizador garantizaría que $d[\hat{t}],\ \hat{t} \in \{t',\ t''\}$ fuesen también compartidos. Teniendo en cuenta la estrategia de análisis sintáctico que hemos adoptado, que identifica estructuras sintácticas y cálculos, concluimos que las distancias $distBosque(r(i)..s, r(\hat{\jmath})..\hat{t})$, con $(\hat{\jmath}, \hat{t}) \in \{(j',\ t'),\ (j'', t'')\}$, no dependen de cálculos previos sobre la parte compartida, tal y como se muestra en la figura 8.10. Pese a que dicha compartición de estructuras no es relevante para estas distancias, si lo fue durante el cálculo de las distancias correspondientes a los nodos situados en la rama más a la derecha del subárbol inmediatamente a la izquierda de los constituyentes compartidos, a los que denotamos mediante una línea doblemente punteada en el diagrama. Como veremos a continuación, al procesar dichos nodos si es posible reutilizar los valores de las distancias asociadas a nodos pertenecientes a la subestructura compartida y evitar repetir determinados cálculos.

Consideramos ahora el segundo caso, esto es, el cálculo de la distancia entre bosques cuando $r(\hat{\jmath}) \neq r(\hat{t})$, tal y como aparece en la figura 8.11. Así, en relación a los valores alternativos posibles para el cálculo de la distancia mínima, tenemos un razonamiento similar al realizado en el caso de la compartición dentro de una misma *raíz_D*:

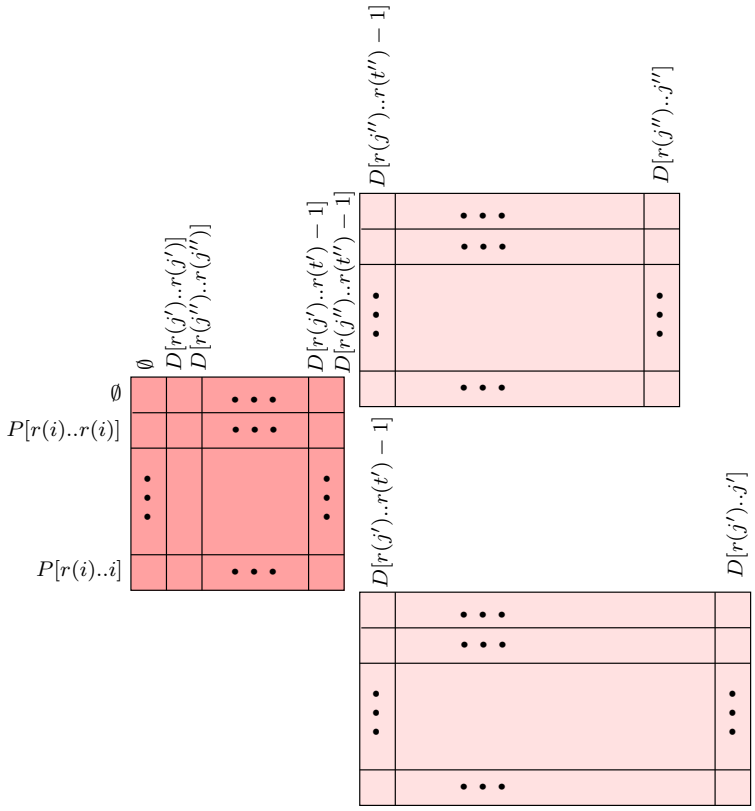


Figura 8.12: Compartición de tablas de distancias entre diferentes *raíces_D*.

1. Los valores $distBosque(r(i)..s-1, r(\hat{\jmath})..\hat{t})$, $(\hat{\jmath}, \hat{t}) \in \{(j',\ t'),\ (j'', t'')\}$ ya han sido calculados en un paso previo del algoritmo, durante el cual la compartición sintáctica ya fue tenida en cuenta al efectuar el cálculo de dichas distancias.

2. Para las distancias $distBosque(r(i)..s, r(\hat{\jmath})..\hat{t}-1)$, $(\hat{\jmath}, \hat{t}) \in \{(j',\ t'),\ (j'', t'')\}$ distinguimos dos casos en relación a la naturaleza de los nodos $d[\hat{t}]$. Aplicaremos un razonamiento análogo al considerado cuando sólo teníamos una *raíz_D*:

   - Si ambos nodos son hojas, entonces $r(\hat{t}) = \hat{t}$. En consecuencia tenemos que

     $$d[t'-1] = \quad d[r(t')-1] = \quad d[r(t'')-1] = \quad d[t''-1]$$

     y para $\hat{t} \in \{t',\ t''\}$, las distancias $distBosque(r(i)..s, r(j)..\hat{t}-1)$ son iguales.
   - En otro caso, siguiendo el postorden inverso, alcanzamos las hojas más a la derecha en las derivaciones de $d[t']$ y $d[t'']$, donde se aplicaría el razonamiento del caso anterior.

3. Los valores para las distancias $distBosque(r(i)..r(s)-1, r(\hat{j})..r(\hat{t}-1))$, con $\hat{t} \in \{t',\ t''\}$ son idénticos, dado que los árboles con raíces en $d[r(\hat{t})-1]$, $\hat{t} \in \{t',\ t''\}$ son compartidos por el analizador sintáctico.

En la figura 8.12 se refleja el razonamiento realizado en este caso y muestra la compartición de las tablas de distancias *distBosque*. Como podemos ver, en la compartición entre distintas *raíces_D* tenemos dos o más *raíces_D* que comparten una subestructura situada en sus extremos más a la derecha. Por lo tanto, en esta situación sí será posible reutilizar los valores de las distancias correspondientes a los nodos compartidos. De modo análogo a lo que sucedía en la compartición dentro de una misma *raíz_D*, no es

posible tal compartición de cálculos si las estructuras comunes no están ubicadas en el extremo derecho. En ese caso, cada región compartida estaría insertada en un contexto derecho distinto, por lo que los valores de sus distancias serán diferentes y requerirán ser cálculos por separado.

8.4. Ejemplo práctico

Para ilustrar el funcionamiento de las modificaciones presentadas en los apartados anteriores, mostraremos a continuación un ejemplo práctico que muestra como se realiza el cálculo de distancias en bosques compartidos. En concreto, nos centraremos en el caso de la compartición de cálculos en una misma *raíz_D*. Para este ejemplo tomaremos como base el lenguaje de las expresiones aritméticas simplificado. Consideraremos una gramática no determinista, G_N, que lo genera, definida por las reglas:

$$\begin{array}{lcl} S & \rightarrow & S \ + \ S \\ S & \rightarrow & a_1 \mid a_2 \mid a_3 \mid a_4 \end{array}$$

donde los terminales $a_1, a_2, a_3, a_4 \in \mathbb{N}$ representan números naturales. Dada esta gramática, utilizamos nuestro analizador para construir el bosque de análisis compartido para la sentencia de entrada $w = a_1 + a_2 + a_3 + a_4$. En la figura 8.13 se muestra parte de dicho bosque compartido, junto con el árbol patrón que utilizaremos en este ejemplo. Para facilitar la comprensión del ejemplo se incluye la representación clásica de ambos árboles junto con las representaciones como grafos Y-O empleadas en nuestro analizador.

Como en el caso general descrito anteriormente, $d[t']$ y $d[t'']$ son los nodos de D que estamos tratando en paralelo y $d[j]$ es la *raíz_D* en la cual están enmarcados. Del mismo modo los nodos $d[k']$ y $d[k'']$ son las raíces de las dos alternativas del nodo-O $d[k]$. La región sombreada en color más claro se corresponde con los nodos cuya distancia fue calculada en pasos previos del algoritmo antes de alcanzar el nodo $d[k]$ de acuerdo al recorrido en postorden inverso. Es decir, son los nodos que forman el bosque $D[r(j)..r(k)-1]$. Las regiones sombreadas en color oscuro representan a una estructura compartida que, en este caso, se corresponde con los bosques $D[r(\hat{k})..r(\hat{t})-1]$, con $\hat{k} \in \{k', k''\}$ y $\hat{t} \in \{t', t''\}$.

En lo que respecta al cálculo de las distancias de edición, la figura 8.14 muestra un conjunto de tablas con los valores de las distancias entre los subbosques del árbol patrón y los subbosques inducidos construidos a partir de los descendientes de la *raíz_D* $d[j]$. Estas distancias han sido calculadas considerando una métrica discreta para asignar el coste de las operaciones de edición: las operaciones de borrado e inserción tienen coste uno; el coste del cambio de etiquetas es cero si ambas etiquetas coinciden, o uno en caso contrario.

Cada fila de las tablas de la figura 8.14 se corresponde con cada uno de los subbosques inducidos que pueden ser construidos en el árbol dato, comenzando en el nodo etiquetado "a_3". En el caso de las columnas, tenemos una por cada posible subbosque inducido que comience en el nodo "a_4", en cada uno de los árboles incluidos en el grafo Y-O dato. La tabla situada más a la izquierda muestra los cálculos de las distancias que son comunes a todos los árboles incluidos en el grafo Y-O. Esta tabla tiene dos partes, la primera almacena los valores da las distancias para los subbosques situados justo antes de las alternativas del nodo-O. La segunda parte, resaltada en negrita, contiene las columnas con los valores de las distancias asociadas a los nodos de la estructura compartida del grafo dato, comprendida entre los nodos $r(\hat{k})$ y $r(\hat{t}-1)$. Las otras dos tablas de la parte derecha de la figura 8.14 almacenan las distancias calculadas para los nodos pertenecientes a las dos alternativas del nodo-O $d[k]$. Finalmente, en el extremo más a la derecha de la figura se muestran las distancias para el nodo-O y para los demás nodos que lo siguen hasta llegar a la *raíz_D* $d[j]$.

Centraremos nuestro ejemplo en el cálculo de las distancias relativas al nodo del árbol patrón etiquetado como "a_2", esto es, cuando el índice s apunta a dicho nodo. Por lo tanto, en este caso estamos calculando distancias con respecto al subbosque patrón que comienza en el nodo etiquetado como "a_3" y termina en el nodo etiquetado "a_2". Lo que se corresponde con distancias almacenadas en las filas etiquetadas como $P[a_3..a_2]$ de la figura 8.14. Retomando las explicaciones presentadas en secciones anteriores respecto a la compartición de cálculos dentro de una misma *raíz_D*, necesitamos los valores de las siguientes distancias para calcular $distBosque(r(i)..s, r(j)..\hat{t})$, con $\hat{t} \in \{t', t''\}$:

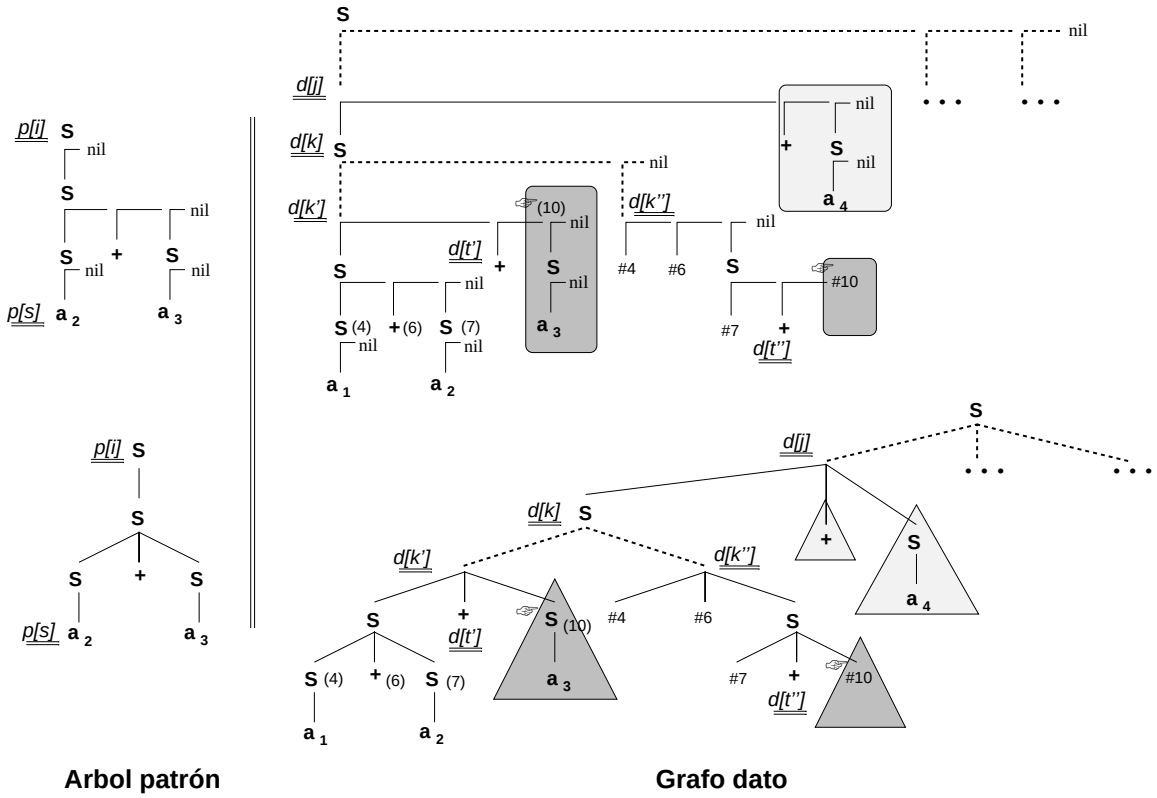


Figura 8.13: Ejemplo de compartición en una misma *raíz_D*.

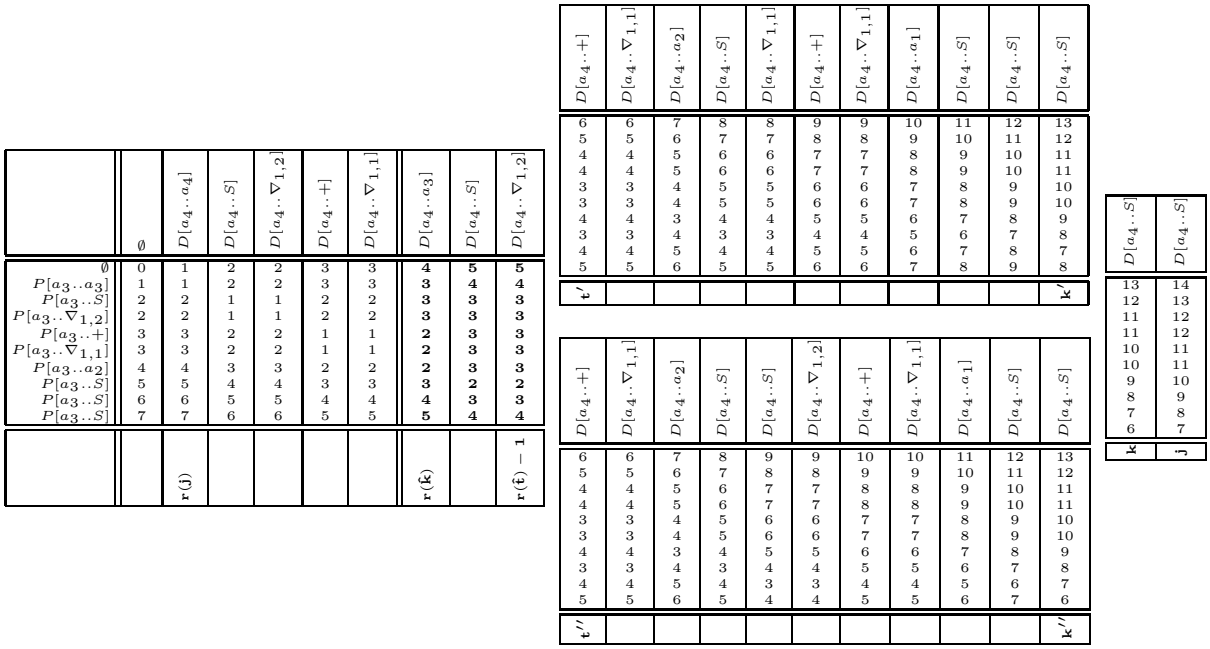
1. *Los valores* $distBosque(r(i)..s-1, r(j)..\hat{t})$. Estas distancias fueron calculadas en un paso previo del algoritmo, cuando fue procesado el nodo $p[s-1]$ del árbol patrón, que está etiquetado con el símbolo auxiliar $\nabla_{1,1}$. En el ejemplo de la figura 8.14, esto significa que se tomarán los valores almacenados en la fila $P[a_3..\nabla_{1,1}]$ de las columnas marcadas como t' y t''. Por la definición de distancias en presencia de símbolos $\nabla_{r.i}$, dichos valores coinciden con los de las distancias almacenadas en la fila $P[a_3..+]$ Para t' tomaríamos el valor 4 y le sumaríamos el coste $\gamma(a_2 \to \varepsilon) = 1$. Para t'' sumaríamos $\gamma(a_2 \to \varepsilon) = 1$ al valor 4 tomado de la columna $P[a_3..\nabla_{1,1}]$.

2. *Los valores* $distBosque(r(i).s, r(j)..\hat{t}-1)$. En nuestro ejemplo, dado que los nodos t' y t'' son ambos hojas, tenemos que $r(\hat{t}) = \hat{t}$. Por lo tanto las dos distancias $distBosque(r(i).s, r(j)..\hat{t}-1)$ son, en realidad, la misma, que está asociada al último nodo de la estructura compartida. En la figura 8.14, esto supone tomar el valor almacenado en la fila $P[a_3..a_2]$ de la columna etiquetada como $r(\hat{t})-1$, que se corresponde con la última columna de la submatriz asociada a la estructura compartida. De este modo tomamos el valor 3 de la fila $P[a_3..a_2]$ y le añadimos el coste $\gamma(\varepsilon \to +)$ para el caso de t' y el de $\gamma(\varepsilon \to +)$ para t''.

3. *Los valores* $distBosque(r(i)..r(s)-1, r(j)..r(\hat{t})-1)$. En este caso los índices $r(t')-1$ y $r(t'')-1$ se refieren al mismo nodo, que es la raíz de la estructura compartida. En nuestro ejemplo tenemos que el índice $r(s)-1$ apunta al nodo del árbol patrón etiquetado con el símbolo auxiliar $\nabla_{1,1}$. Por lo tanto, debemos tomar las distancias almacenadas en la fila $P[a_3..\nabla_{1,1}]$[3] de la columna $r(\hat{t})-1$ de la figura 8.14.

[3]Que coinciden con las almacenadas en la fila $P[a_3..+]$.

En lo que respecta a las distancias $distArbol(s, \hat{t})$, estas ya habrán sido calculadas en pasos previos del algoritmo y almacenadas en las entradas correspondientes de la matriz *distArbol*. Para el caso de t' este valor es 1 y para t'' también es 1. Para obtener la distancia deseada se suma cada uno de esos valores a la distancia almacenada en la fila $P[a_3..+]$ de la columna $r(\hat{t}) - 1$, cuyo valor es 3.



	$\emptyset$	$D[a_4..a_4]$	$D[a_4..S]$	$D[a_4..\nabla_{1,2}]$	$D[a_4..+]$	$D[a_4..\nabla_{1,1}]$	$D[a_4..a_3]$	$D[a_4..S]$	$D[a_4..\nabla_{1,2}]$
$\emptyset$	0	1	2	2	3	3	**4**	**5**	**5**
$P[a_3..a_3]$	1	1	2	2	3	3	**3**	**4**	**4**
$P[a_3..S]$	2	2	1	1	2	2	**3**	**3**	**3**
$P[a_3..\nabla_{1,2}]$	2	2	1	1	2	2	**3**	**3**	**3**
$P[a_3..+]$	3	3	2	2	1	1	**2**	**3**	**3**
$P[a_3..\nabla_{1,1}]$	3	3	2	2	1	1	**2**	**3**	**3**
$P[a_3..a_2]$	4	4	3	3	2	2	**2**	**3**	**3**
$P[a_3..S]$	5	5	4	4	3	3	**3**	**2**	**2**
$P[a_3..S]$	6	6	5	5	4	4	**4**	**3**	**3**
$P[a_3..S]$	7	7	6	6	5	5	**5**	**4**	**4**
		r(j)					**r(k)**		**r(t̂) − 1**

$D[a_4..+]$	$D[a_4..\nabla_{1,1}]$	$D[a_4..a_2]$	$D[a_4..S]$	$D[a_4..\nabla_{1,1}]$	$D[a_4..+]$	$D[a_4..\nabla_{1,1}]$	$D[a_4..a_1]$	$D[a_4..S]$	$D[a_4..S]$	$D[a_4..S]$
6	6	7	8	8	9	9	10	11	12	13
5	5	6	7	7	8	8	9	10	11	12
4	4	5	6	6	7	7	8	9	10	11
4	4	5	6	6	7	7	8	9	10	11
3	3	4	5	5	6	6	7	8	9	10
3	3	4	5	5	6	6	7	8	9	10
4	4	3	4	4	5	5	6	7	8	9
3	3	4	3	3	4	4	5	6	7	8
4	4	5	4	4	5	5	6	7	8	7
5	5	6	5	5	6	6	7	8	9	8
t′										**k′**

$D[a_4..S]$	$D[a_4..S]$
13	14
12	13
11	12
11	12
10	11
10	11
9	10
8	9
7	8
6	7
k	**j**

$D[a_4..+]$	$D[a_4..\nabla_{1,1}]$	$D[a_4..a_2]$	$D[a_4..S]$	$D[a_4..S]$	$D[a_4..\nabla_{1,2}]$	$D[a_4..+]$	$D[a_4..\nabla_{1,1}]$	$D[a_4..a_1]$	$D[a_4..S]$	$D[a_4..S]$
6	6	7	8	9	9	10	10	11	12	13
5	5	6	7	8	8	9	9	10	11	12
4	4	5	6	7	7	8	8	9	10	11
4	4	5	6	7	7	8	8	9	10	11
3	3	4	5	6	6	7	7	8	9	10
3	3	4	5	6	6	7	7	8	9	10
4	4	3	4	5	5	6	6	7	8	9
3	3	4	3	4	4	5	5	6	7	8
4	4	5	4	3	3	4	4	5	6	7
5	5	6	5	4	4	5	5	6	7	6
t″										**k″**

Figura 8.14: Matriz *distBosque* compartida para los árboles de ejemplo.

Finalmente, tal como indica el algoritmo debemos obtener el mínimo de estos tres valores calculados para t' y t'' respectivamente. De este modo, se llega a que $distBosque(r(i)..s, r(j)..t') = 4$ y $distBosque(r(i)..s, r(j)..t'') = 4$ y se almacenan ambos valores en las celdas de las tablas de distancias *distBosque* que correspondan.

8.5. Símbolos VLDC en bosques compartidos

Dedicaremos esta sección a presentar el modo en que se pueden extender nuestras propuestas respecto al reconocimiento eficiente de patrones en bosques compartidos, para permitir el uso de árboles patrón con símbolos VLDC. Esto es, veremos como integrar las técnicas de reconocimiento aproximado de patrones presentadas en el capítulo 6 con los bosques de análisis compartidos generados por ICE. Justificaremos cómo las modificaciones y optimizaciones descritas anteriormente siguen siendo válidas para el caso del cálculo de distancias de edición donde estén implicados símbolos VLDC. Y describiremos como afecta la utilización de bosques compartidos a este tipo de reconocimiento aproximado.

Comenzamos presentando las formulas para el cálculo de las distancias con VLDC empleando la ordenación en postorden inverso. En este caso, la presencia o ausencia de estos símbolos en el árbol patrón no es relevante en lo que respecta al razonamiento efectuado en la sección 8.2.1 a cerca de la conveniencia de emplear esta nueva ordenación. En efecto, las razones que aconsejan este cambio de orden se centran únicamente en las posibilidades de compartición específicas de los bosques compartidos generados por ICE, así como de su topología, derivadas ambas de las particularidades del formalismo de representación empleado.

En este punto, tampoco es relevante el hecho de que en este tipo de reconocimiento aproximado necesitemos calcular las distancias auxiliares entre sufijos, *dbs*. El mecanismo de cálculo seguido en este

caso es análogo al empleado para obtener las distancias *distBosque*, puesto que ambos diferencian entre distancias entre subbosques y distancias entre subárboles y emplean, en el algoritmo original, el concepto de *raíz_I*. Por ello, las consideraciones realizadas para el caso de *distBosque* son también válidas para las distancias *dbs*. Así, tenemos los siguientes conjuntos de fórmulas para los cálculos de distancias entre subárboles donde estén implicados símbolos VLDC:

Si $r(s) = r(i)$ **y** $r(t) = r(j)$**:**

$$distBosque(r(i)..s, r(j)..t) = \begin{cases} \min \begin{cases} distBosque(r(i)..s-1, r(j)..t) + \gamma(p[s] \rightarrow \varepsilon), \\ distBosque(r(i)..s, r(j)..t-1) + \gamma(\varepsilon \rightarrow d[t]), \\ distBosque(r(i)..s-1, r(j)..t-1) + \gamma(p[s] \rightarrow d[t]), \\ distBosque(\emptyset, r(j)..t-1) + \\ \qquad \min_k \{distArbol(s,k) - distArbol(\emptyset,k) \mid 1 \leq k \leq n_t\} \end{cases} \\ \textbf{si } p(s) = \text{“ | “} \\ \\ \min \begin{cases} distBosque(r(i)..s-1, r(j)..t) + \gamma(p[s] \rightarrow \varepsilon), \\ distBosque(r(i)..s, r(j)..t-1) + \gamma(\varepsilon \rightarrow d[t]), \\ distBosque(r(i)..s-1, r(j)..t-1) + \gamma(p[s] \rightarrow d[t]), \\ min_k\{dbs(r(i)..s-1, r(j)..k) | 1 \leq k \leq n_t\}, \\ min_k\{distArbol(s,k) | 1 \leq k \leq n_t\} \end{cases} \\ \textbf{si } p(s) = \text{“} \wedge \text{“} \end{cases}$$

Al igual que sucedía en la versión original de Zhang y Shasha [66], las fórmulas para obtener las distancias entre subbosques y las distancias entre subárboles sin símbolos VLDC coinciden con las mostradas en la sección 8.2.1. En lo que respecta al cálculo de las distancias auxiliares *dbs* los conjuntos de fórmulas alternativos son los siguientes:

$$\begin{array}{l} dbs(\emptyset, \emptyset) = 0 \\ dbs(r(i)..s, \emptyset) = distBosque(r(i)..s, \emptyset) \\ dbs(\emptyset, r(j)..t) = \begin{cases} 0 & \textbf{sii } r(t) = r(j) \textbf{ ó } r(padre(t)) = r(j) \\ dbs(\emptyset, r(j)..t-1) + \gamma(\varepsilon \rightarrow d[t]) & \text{en otro caso} \end{cases} \end{array}$$

$$dbs(r(i)..s, r(j)..t) = \begin{cases} \min \begin{cases} distBosque(r(i)..s, \emptyset) \\ distBosque(r(i)..s, r(j)..t) \end{cases} \\ \textbf{sii } r(t) = r(j) \\ \\ \min \begin{cases} dbs(r(i)..s-1, r(j)..t) & + & \gamma(p[s] \rightarrow \varepsilon) \\ dbs(r(i)..s, r(j)..t-1) & + & \gamma(\varepsilon \rightarrow d[t]) \\ dbs(r(i)..s-1, r(j)..t-1) & + & distArbol(s,t) \end{cases} \\ \textbf{en otro caso} \end{cases}$$

Tomando como punto de partida las fórmulas anteriores, revisaremos, en primer lugar, los aspectos básicos del reconocimiento de patrones con VLDC que es necesario redefinir para permitir el manejo de distancias respecto a bosques compartidos. Mostraremos también como las modificaciones propuestas en las secciones 8.2.3 y 8.3 mantienen su validez una vez redefinidos dichos conceptos. Una vez presentadas estas modificaciones, veremos como extender el razonamiento empleado en la adaptación y optimización del cálculo del distancias *distBosque* al caso de las distancias auxiliares entre sufijos, *dbs*.

8.5.1. Adaptación del reconocimiento aproximado con VLDC

Comenzaremos recordando el concepto original de sustitución-VLDC aplicado al reconocimiento aproximado en árboles. Una sustitución-VLDC asocia a cada símbolo VLDC presente en el árbol patrón

una porción del árbol dato. Resulta así, un nuevo árbol patrón sin símbolos de este tipo, a partir del cual se obtendrá la distancia final deseada. Dado que en nuestro caso manejamos un bosque compartido, debemos extender la noción de sustitución-VLDC para que refleje la presencia de múltiples árboles en la estructura dato.

Definición 8.7 (sustitución-VLDC en bosques compartidos)**.**
Dados un árbol patrón P que incluya símbolos VLDC *y un bosque compartido dato D, en el cual $B_D = \langle \hat{D}_1, \hat{D}_2, \ldots, \hat{D}_n \rangle$ es el conjunto de árboles que contiene.*
Diremos que s *es una* sustitución-VLDC *sobre P respecto al bosque compartido D si* s *es, a su vez, una sustitución-*VLDC *sobre P respecto a un árbol $\hat{D}_j \in B_D$*

En este caso, consideramos que una sustitución-VLDC respecto a un bosque compartido no es más que una sustitución-VLDC para el árbol patrón P que haya sido construida a partir de los nodos pertenecientes a uno de los árboles incluidos en el bosque dato D. De este modo, para definir la distancia con símbolos VLDC respecto a bosques compartidos extenderemos la definición original de Zhang y Shasha [66], mostrada en la sección 6.1.2 del capítulo 6, de modo análogo a la definición 8.5 para distancias sin VLDC en bosques compartidos.

Definición 8.8 (distancia con VLDC respecto a bosques compartidos)**.**
Dados un árbol patrón P que incluya uno o más símbolos VLDC*, y un bosque de análisis compartido, D, representado en forma de grafo* Y-O*. Si $B_D = \langle \hat{D}_1, \hat{D}_2, \ldots, \hat{D}_n \rangle$ es el conjunto de árboles contenidos en el bosque D, definiremos la* distancia con VLDC de P respecto al bosque compartido D*, $d(P, D)$, como*

$$d_{\text{VLDC}}(P, D) = min\{d_{\text{VLDC}}(P, \hat{D}_i) |\ \hat{D}_i \in B_D\}$$

Como sucedía en el caso de los árboles, no es necesario construir explícitamente las sustituciones-VLDC. Cada una de las distancias calculadas tendrá asociada implícitamente la mejor sustitución-VLDC que la origina. Por lo tanto, se puede plantear una propagación de distancias análoga a la realizada en el caso de patrones sin VLDC. Así, sigue siendo válido el cálculo de distancias en presencia de nodos-O mostrado en el lema 8.1. Siendo posibles, además, la compartición de las distancias asociadas a los contextos anterior y posterior a las distintas alternativas de este tipo de nodos.

Un razonamiento similar puede aplicarse en el caso de la propagación de distancias en bosques binarizados, mostrado en la sección 8.2.3. Como sucedía en ese caso, es necesario tener en cuenta el significado de los símbolos auxiliares $\nabla_{r,i}$, introducidos por ICE durante la construcción de los bosques de análisis, para poder recuperar los valores de distancia que obtendríamos trabajando directamente sobre los árboles originales no binarizados.

8.5.2. Comportamiento de los símbolos ∧-VLDC y |-VLDC

Como mostrábamos en el capítulo 6, los cálculos que implican directamente a símbolos VLDC se realizaban siempre entre subárboles. Nunca se manejaban símbolos VLDC en distancias entre bosques. Nos limitaremos en este apartado, por lo tanto, a mostrar cómo debemos adaptar los cálculos originales respecto a subárboles con raíces VLDC, para el caso de los bosques compartidos generados por ICE. Nuestro objetivo será seguir manteniendo el significado que asociábamos a los símbolos ∧-VLDC y |-VLDC en caso del reconocimiento aproximado sobre árboles. De nuevo nos vemos en la necesidad de manejar los problemas derivados de la binarización ímplicta realizada por ICE.

Nuestro estudio se centra en la problemática asociada al ∧-VLDC, puesto que para el caso del |-VLDC las consideraciones serían análogas. Si intentásemos aplicar directamente la definición original de ∧-VLDC y las fórmulas del lema 6.8 sobre un árbol binarizado, nos encontraríamos con que las sustituciones de dicho símbolo VLDC son muy diferentes a las que obtendríamos sobre el árbol equivalente no binarizado.

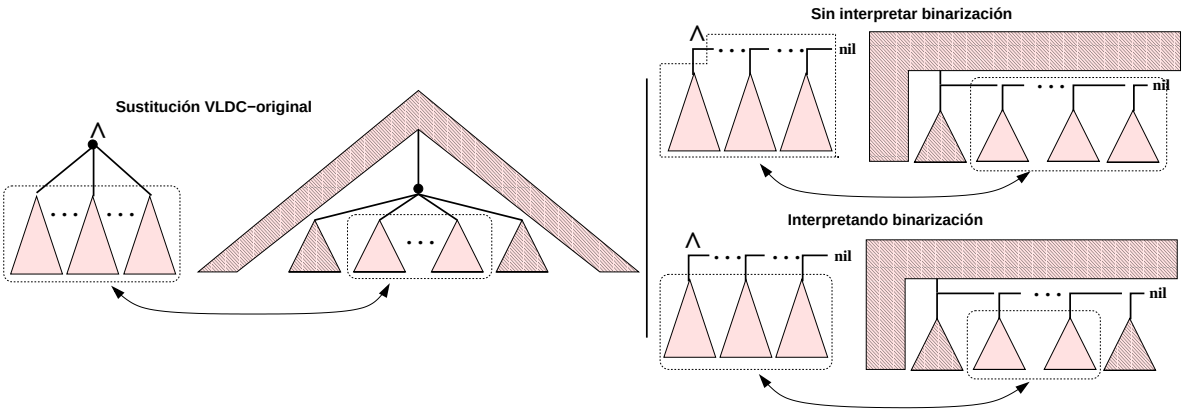


Figura 8.15: Interpretación de los ∧-VLDC en bosques binarizados.

Como muestra en el esquema superior de la parte derecha de la figura 8.15, no sería posible incluir en la sustitución-VLDC subárboles situados a la derecha de la región que no es cubierta por dicha sustitución.

Este comportamiento es consecuencia de las fórmulas del lema 6.8 y de la propia topología del árbol binarizado. Dicho lema nos dice que la distancia asociada a la mejor sustitución se obtiene a partir de las distancias correspondientes a los hijos del nodo $d[t]$. En el caso de árboles binarizados, dicho nodo $d[t]$ sólo tendrá dos hijos posibles. Para el hijo derecho no existe contexto derecho susceptible de ser eliminado al incluirlo en la sustitución-VLDC. Mientras que, por otra parte, la posible sustitución-VLDC para el hijo izquierdo eliminaría a la totalidad de sus hermanos derechos. En ambos casos, el comportamiento que se consigue no se corresponde con el que obtendríamos sobre el árbol no binarizado, aplicando la definición original de ∧-VLDC, mostrado en la parte izquierda de la figura 8.15.

Para mantener la semántica original del ∧-VLDC debemos de recuperar las distancias que se obtendrían sobre los árboles no binarizados equivalentes. Para ello es necesario tener en cuenta a los símbolos auxiliares $\nabla_{r,i}$, que servirán como guía para recuperar los valores de las distancias deseadas. Esta idea se muestra en el esquema inferior de la parte derecha de la figura 8.15. En este caso, lo que se hace es omitir los nodos etiquetados con símbolos $\nabla_{r,i}$ y tomar las distancias asociadas a sus hijos izquierdos. Dado que estos símbolos representan el reconocimiento de porciones de reglas, sus hijos se corresponden con las raíces de los subárboles de los cuales se deberían tomar las distancias para los cálculos del lema 6.8.

Para poder aplicar esta idea debemos poder identificar a esos subárboles hijo dentro del bosque binarizado. Utilizaremos la notación $hijo(i,t)$ para referirnos al índice en postorden inverso que corresponde, en nuestro bosque compartido, al que sería el i-ésimo hijo del nodo $d[t]$ en el árbol no binarizado equivalente. Si suponemos que el nodo $d[t]$ tiene n_t hijos en el árbol no binarizado, dicha función se define formalmente del siguiente modo.

Definición 8.9 (función $hijo(i,t)$)**.**
Dado un bosque compartido binarizado, D, que contiene al conjunto de árboles $B_D = \langle \hat{D}_1, \hat{D}_2, \ldots, \hat{D}_n \rangle$ Sea $d[t]$ un nodo de un árbol $\hat{D}_k \in B_D$ y sea n_t el número de hijos que tiene el nodo equivalente a $d[t]$ en el árbol no binarizado, T_k, asociado a $\hat{D}_k$. La notación $hijo(i,t)$ se refiere al índice en postorden inverso asociado en el árbol D_k al nodo equivalente al i-ésimo hijo de $d[t]$ en el árbol no binarizado T_k, y se define como:

$$hijo(i,t) = \begin{cases} t & \textbf{si } i = 0 \\ t-1 & \textbf{si } i = 1 \\ hijo(1, r(hijo(i-1,t)) - 1) & \textbf{en otro caso} \end{cases}$$

Intuitivamente, el caso general de la función $hijo(i,t)$ se basa en que en los árboles binarizados considerados, el índice en postorden inverso del hermano derecho de un nodo k se corresponde con

el valor $r(k) - 1$. Así, lo que se hace en el tercer caso de la expresión anterior para identificar al hijo i-ésimo de $d[t]$, es tomar el primer descendiente del hermano derecho del hijo de $d[t]$ situado en la posición $i-1$. Verificándose, además, que el mencionado hermano derecho será un nodo etiquetado con un símbolo auxiliar $\nabla_{r,i}$.

Aplicando esta función podremos recuperar las distancias asociadas a los hijos de un nodo determinado del bosque dato. Mostramos a continuación el conjunto de fórmulas resultantes para el caso de distancias entre subárboles donde la raíz del patrón es un símbolo VLDC. Como se puede apreciar, se trata de una adaptación directa de las mostradas al comienzo de la sección 8.5, en la que se ha incluido el uso de la notación $hijo(i,t)$. Siendo n_t el número de descendientes del nodo $d[t]$ en el árbol no binarizado equivalente, tenemos que:

Si $r(s) = r(i)$ **y** $r(t) = r(j)$**:**

$$distBosque(r(i)..s, r(j)..t) = \begin{cases} \min \begin{cases} distBosque(r(i)..s-1, r(j)..t) + \gamma(p[s] \rightarrow \varepsilon), \\ distBosque(r(i)..s, r(j)..t-1) + \gamma(\varepsilon \rightarrow d[t]), \\ distBosque(r(i)..s-1, r(j)..t-1) + \gamma(p[s] \rightarrow d[t]), \\ distBosque(\emptyset, r(j)..t-1) + \\ \qquad \min_k \{distArbol(s, hijo(k,t)) - \\ distArbol(\emptyset, hijo(k,t)) \mid 1 \leq k \leq n_t\} \end{cases} \\ \textbf{si } p(s) = \text{“ } \mid \text{ “} \\ \min \begin{cases} distBosque(r(i)..s-1, r(j)..t) + \gamma(p[s] \rightarrow \varepsilon), \\ distBosque(r(i)..s, r(j)..t-1) + \gamma(\varepsilon \rightarrow d[t]), \\ distBosque(r(i)..s-1, r(j)..t-1) + \gamma(p[s] \rightarrow d[t]), \\ min_k\{dbs(r(i)..s-1, r(j)..hijo(k,t)) | 1 \leq k \leq n_t\}, \\ min_k\{distArbol(s, hijo(k,t)) | 1 \leq k \leq n_t\} \end{cases} \\ \textbf{si } p(s) = \text{“ } \wedge \text{ “} \end{cases}$$

Sin embargo, en el caso del $\wedge$-VLDC no basta con estas modificaciones. Como veremos en el próximo apartado, es necesario adaptar el cálculo de las distancias auxiliares entre sufijos, *dbs*, para permitir que los cálculos sobre bosques compartidos sean equivalentes a los obtenidos en árboles clásicos.

8.5.3. Distancia auxiliar entre sufijos en bosques compartidos

Comenzamos, entonces, con el estudio del cálculo de la distancia auxiliar entre sufijos, *dbs*, en bosques compartidos. En primer lugar, es necesario modificar ligeramente su definición original para adaptarla a la nueva ordenación de los nodos. Así, en la propuesta original de Zhang y Shasha [66], teníamos que, siendo P un árbol patrón yD un árbol dato, $dbs(l(i)..s, l(j)..t)$ se refería a la menor distancia posible entre el bosque $P[l(i)..s]$ y un subbosque de $D[l(j)..t]$ en el cual habían sido eliminados un conjunto de subárboles consecutivos situados en su extremo izquierdo. Dado que ahora empleamos un postorden inverso, debemos cambiar la orientación del algoritmo y emplear una definición simétrica.

Definición 8.10 (distancia *dbs* con postorden inverso)**.**
Dados un árbol patrón P y un árbol dato D. Siendo B_P un bosque de P y B_D un bosque de D, se define la distancia entre sufijos siguiendo en postorden inverso*, entre los bosques B_P y B_D, en forma abreviada $dbs(B_P, B_D)$, como la distancia entre los bosques B_P y B'_D, en donde B'_D es un subbosque de B_D en cual ha sido eliminado un conjunto de subárboles verificando las siguientes restricciones:*

- *Todos ellos son subárboles completos.*
- *Son consecutivos*
- *Están situados en el extremo más a la derecha del bosque B_D*

- *Todos tienen el mismo nodo padre.*

Resultando que $dbs(B_P, B_D) = min_{B'_D}\{distBosque(B_P, B'_D)\}$.

En este caso, la única diferencia con respecto a la definición original, introducida en la sección 6.2.3, es que se considera la eliminación de conjuntos de subárboles por la derecha y no por la izquierda del bosque.

Nos centraremos ahora en estudiar como influyen las características específicas del formalismo de representación utilizado por ICE en el cálculo de este tipo de distancias. En lo que respecta al manejo de nodos-O durante el cálculo de las distancias *dbs*, nos encontramos con una situación análoga a la presentada en la sección 8.2.2 para el caso de las distancias *distBosque*. Consideraremos las extensiones en la numeración en postorden, introducidas en dicha sección, para manejar la identificación de nodos y subbosques incluidos en bosques compartidos. De este modo, tenemos la siguiente definición de distancia entre bosques sufijos en bosques compartidos:

Definición 8.11 (*dbs* respecto a bosques compartidos)**.**
Siendo P un árbol patrón etiquetado y ordenado y D un grafo Y-O*. Dados los nodos $s \in P$ y $t \in D$ y los raíces_D $i \in anc(s)$ en T y $j \in anc(t)$ en D, la distancia $dbs(P[r(i)..s], D[r(j)..t])$ se define como:*

$$dbs(P[r(i)..s], D[r(j)..t]) = min\{dbs(P[r(i)..s], \hat{D}[r(j)..t])|\ \hat{D}[r(j)..t] \in D[r(j)..t]\}$$

Al igual que en la definición 8.5, lo que deseamos calcular es el valor de la distancia *dbs* entre el bosque patrón $P[r(i)..s]$ y el bosque contenido en $D[r(j)..t]$ que sea más próximo a dicho patrón, manteniendo la suposición de que se elimina el conjunto óptimo de subárboles en su extremo derecho. Por lo tanto, para las distancias *dbs* que afecten a un nodo-O, podemos enunciar un resultado análogo al del lema 8.1.

Lema 8.2.
Sean P un árbol patrón y D un grafo Y-O*. Dados un nodo $i \in raíces\_D(P)$, un nodo $s \in desc(i)$ y un nodo $j \in raíces\_D(D)$. Si tenemos un nodo-O $k \in desc(j)$ con dos alternativas numeradas k' y k'', se verifica que:*

$$dbs(P[r(i)..s], D[r(j)..k]) = min\{\ dbs(P[r(i)..s], D[r(j)..k']),\ dbs(P[r(i)..s], D[r(j)..k'')\}$$

En el caso de que k tuviera n alternativas, $k'_1, k'_2, \ldots, k'_n$; la expresión resultante sería $dbs(P[r(i)..s], D[r(j)..k]) = min_{1 \leq i \leq n}\{dbs(P[r(i)..s], D[r(j)..k'_i])\}$.

Demostración.
La demostración es análoga a la del lema 8.1, simplemente basta tomar como base la definición 8.11. □

La consecuencia de este resultado es que en el caso de que existan nodos-O implicados en el cálculo de distancias *dbs*, podemos limitarnos a propagar las distancias mínimas, igual que hacíamos con las distancias *distBosque* en la sección 8.2.2. Por lo tanto, será posible compartir las tablas de distancias *dbs* asociadas a los contextos anterior y posterior a las alternativas de un nodo-O de la misma forma que mostramos en la figura 8.5.

En lo que respecta a las implicaciones de la binarización implícita del bosque dato sobre el cálculo de las distancias *dbs*, nuestro objetivo es asegurar la recuperación de las distancias que se obtendrían en el caso de operar directamente sobre los árboles no binarizados. En efecto, si no incluyéramos esa restricción, el cálculo de las distancias *dbs* en bosques binarizados sería trivial, puesto que sólo podría existir, como máximo, un subárbol a la derecha susceptible de ser eliminado. Sin embargo, realizando el cálculo de ese modo, se perdería la semántica de este tipo de distancia, que a su vez es consecuencia del significado asociado a los símbolos ∧-VLDC y al tipo de sustitución que realizan.

En nuestro caso, podemos emplear la misma estrategia expuesta en la sección 8.2.3. Lo que haremos será aprovechar el hecho de que los símbolos auxiliares $\nabla_{r,i}$ representan el reconocimiento parcial de un

subbosque para modificar las fórmulas del cálculo de distancias *dbs* afectas. De ese modo, empleando la siguiente definición de *dbs* podemos propagar y recuperar los valores de las distancias *dbs* que se obtendrían en los correspondientes árboles no binarizados.

Definición 8.12 (distancias *dbs* con símbolos $\nabla_{r,i}$)**.**
Siendo $i \in P$ y $j \in D$ raíces_D y dados los nodos $s \in desc(i)$ y $t \in desc(j)$, las distancias que impliquen a nodos etiquetados con símbolos $\nabla_{r,i}$ se definirán del siguiente modo:

$$dbs(P[r(i)..s], D[r(j)..t]) := \begin{cases} dbs(P[r(i)..s-1], D[r(j)..t]) & \textbf{si } p[s] = \nabla_{r,i} \\ dbs(P[r(i)..s], D[r(j)..t-1]) & \textbf{si } d[t] = \nabla_{r,i} \end{cases}$$
$$\forall \textit{ símbolo auxiliar } \nabla_{r,i}$$

Como en el caso de *distBosque*, lo que hacemos es recuperar las distancias asociadas al hijo derecho de los símbolos auxiliares $\nabla_{r,i}$, que se corresponden con las del último nodo incluido en el subbosque representado por dicho símbolo. Puesto que este cálculo se realiza en todos los nodos de este tipo, se logra asegurar que los valores *dbs* manejados son los mismos que obtendrían si se trabajara directamente sobre árboles no binarizados.

El último aspecto que debemos de tratar en cuanto a la adaptación del cálculo de distancias *dbs* en bosques compartidos es la posibilidad de reutilización de cálculos. A este respecto, recordemos que la aproximación que seguimos en nuestro trabajo es la de utilizar la compartición de estructuras sintácticas, para evitar la repetición de cálculos de distancias innecesarios.

Retomaremos el estudio realizado en la sección 8.3 respecto a las posibilidades de compartición de cálculos en función de las posiciones relativas de los nodos-o y de las *raíces_D* del bosque dato. Como hemos visto, distinguimos dos situaciones en las cuales la compartición de estructuras se puede transformar en compartición de distancias: la compartición dentro de una misma *raíz_D* y la compartición entre distintas *raíces_D*. Si revisamos el razonamiento seguido en las secciones 8.3.1 y 8.3.2, comprobamos que las fórmulas que se ven afectadas por estos dos tipos de compartición se refieren exclusivamente a cálculos de distancias entre subbosques.

A la vista de las fórmulas para el cálculo de distancias auxiliares *dbs* entre subbosques mostradas en secciones anteriores, y comparándolas con las utilizadas en el caso de distancias *distBosque*, comprobamos que en ambos casos existe un paralelismo en lo referente a las dependencias entre las distancias parciales empleadas. Tenemos que el cálculo de la distancia $distBosque(r(i)..s, r(j)..t)$ depende de los valores previos de $distBosque(r(i)..s-1, r(j)..t)$, $distBosque(r(i)..s, r(j)..t-1)$, $distBosque(r(i)..r(s)-1, r(j)..r(t)-1)$ y $distArbol(s,t)$. Mientras que, en el caso de las distancias auxiliares, $dbs(r(i)..s, r(j)..t)$ depende de $dbs(r(i)..s-1, r(j)..t)$, $dbs(r(i)..s, r(j)..t-1)$, $dbs(r(i)..r(s)-1, r(j)..r(t)-1)$ y $distArbol(s,t)$.

De esta comparación concluimos que las dependencias existentes entre distancias parciales *distBosque* son las mismas que existen entre distancias *dbs*. Por lo tanto, el razonamiento empleado en las secciones 8.3.1 y 8.3.2 para la compartición de distancias *distBosque* es perfectamente válido para el caso de las distancias *dbs*. Bastará simplemente, con transformar las fórmulas presentadas en el caso de la compartición en una misma *raíz_D* y de la compartición entre distintas *raíces_D*, en sus fórmulas equivalentes para distancias *dbs*. Del mismo, modo la compartición de las tablas de distancias se llevará a cabo de modo idéntico al que se muestra en las figuras 8.9 y 8.12, respectivamente.

Parte IV

Evaluación y Conclusiones

Capítulo 9
Resultados experimentales

Una vez descrita la adaptación del algoritmo de Zhang y Shasha [65] para manejar bosques de análisis compartidos, junto con las optimizaciones propuestas para sacar provecho de la compartición de estructuras sintácticas, ofrecemos en este capítulo los resultados experimentales obtenidos en la valoración de nuestras propuestas. El objetivo de estos experimentos es evaluar la eficacia computacional de las técnicas de reconocimiento de patrones en bosques de análisis compartidos que proponemos, así como características relativas a su utilización práctica en posibles entornos reales.

Hemos estructurado nuestros experimentos en dos grupos. En primer lugar, mostraremos una evaluación de los aspectos relativos al coste computacional de nuestro mecanismo de cálculo de distancias en bosques compartidos. Evaluaremos como afecta la topología de los árboles y bosques manejados a la eficiencia del proceso de reconocimiento. También presentaremos una serie de resultados experimentales que ponen de manifiesto la mejora en la eficiencia que resulta de emplear las estructuras compartidas generadas por nuestro analizador, para evitar cálculos redundantes. Dichos resultados muestran las ventajas derivadas del empleo de los distintos tipos de compartición de cálculos estudiados en el capítulo 8 para optimizar el mecanismo de cálculo de distancias en bosques compartidos.

En segundo lugar, hemos realizado un estudio experimental sobre los distintos tipos de símbolos VLDC presentados en el capítulo 6. Hemos estudiado cómo afectan las diferencias expresivas entre ambos símbolos, |-VLDC y ∧-VLDC, al coste computacional del algoritmo de reconocimiento aproximado sobre bosques compartidos. El interés de este tipo de experimentos es que permiten evaluar cómo influye el esfuerzo realizado en la especificación de los árboles patrón, bien por parte de un usuario o por parte de un mecanismo de generación automática de patrones, en el esfuerzo computacional que se efectúa durante el proceso de reconocimiento.

9.1. Evaluación de la compartición de cálculos y del rendimiento

Dedicaremos esta primera sección a presentar una serie de resultados experimentales que ponen de manifiesto las ventajas en cuanto al rendimiento que se obtienen con la incorporación de las optimizaciones expuestas en capítulos anteriores. Hemos realizado dos baterías de tests distintas. En la primera de ellas se ha empleado una serie de gramáticas para las expresiones aritméticas. El objetivo ha sido evaluar, en primer lugar, como se ve afectada la eficiencia del mecanismo de reconocimiento de patrones por la topología de los grafos Y-O dato. Un segundo objetivo es poner de manifiesto la ganancia en eficiencia que se deriva del uso de la compartición de cálculos basada en la utilización de las estructuras compartidas generadas por ICE. En el segundo conjunto de tests se ha empleado una gramática para frases sencillas en español. El principal objetivo aquí ha sido el de evaluar nuestra aportación en un entorno más próximo al de sus posibles aplicaciones reales. Esto es, su aplicación en sistemas de recuperación y extracción basados en técnicas de PLN; en concreto, en el uso de estructuras sintácticas para mejorar las tareas de indexación y consulta en estos sistemas. Este conjunto de tests ofrece, pese a su simplicidad, un entorno relativamente próximo a uno real, en el cual evaluar la eficiencia de nuestras propuestas de cara a posibles aplicaciones prácticas.

$\mathcal{G}_\mathbf{N}$	$\mathcal{G}_\mathbf{I}$	$\mathcal{G}_\mathbf{D}$
$S \rightarrow S + S$	$E \rightarrow E + T$	$E \rightarrow T + E$
$S \rightarrow a$	$T \rightarrow a$	$T \rightarrow a$
$S \rightarrow b$	$T \rightarrow b$	$T \rightarrow b$

Figura 9.1: Gramáticas para las expresiones aritméticas empleadas en los tests.

Consideramos en primer lugar el lenguaje de las expresiones aritméticas, comparando nuestra propuesta con Tai [43], y Zhang y Shasha [65]. Hemos empleado las dos gramáticas deterministas, $\mathcal{G}_I$ y $\mathcal{G}_D$, mostradas en la figura 9.1, representando las versiones asociativas por la izquierda y por la derecha de los operadores aritméticos; y una no determinista, $\mathcal{G}_N$. Asumimos que los analizadores han sido generados mediante ICE [56], y que los tests han sido aplicados sobre los bosques de análisis correspondientes a cadenas de entrada de la forma $w = a_1 + a_2 + \cdots + a_i + a_{i+1}$, con i impar. En el caso no determinista estos bosques de análisis tienen un número, C_i, de análisis ambiguos que crece exponencialmente con i:

$$C_0 = C_1 = 1 \quad \text{y} \quad C_i = \left(\begin{array}{c} 2i \\ i \end{array} \right) \frac{1}{i+1}, \text{ si } i > 1$$

Como patrón hemos considerado los árboles deterministas generados para entradas de la forma $a_1 + b_1 + a_3 + b_3 + \cdots + b_{i-1} + a_{i-1} + b_{i+1} + a_{i+1}$, donde $b_j \neq a_{j-1}$; siguiendo el formalismo de representación de ICE.

La medida básica en la que nos hemos centrado en esta evaluación es el número de operaciones elementales que es necesario realizar para calcular las distancias respecto a cada árbol dato. En este punto, debemos señalar que entendemos por operación elemental cada uno de los cálculos de distancias mínimas entre subbosques inducidos que se realizan durante el proceso de reconocimiento de patrones. Es decir, cada valor mínimo calculado para dos subbosques o dos subárboles representa una operación elemental, que se corresponderá con la aplicación de uno de los dos conjuntos de fórmulas descritos en el capítulo 8.

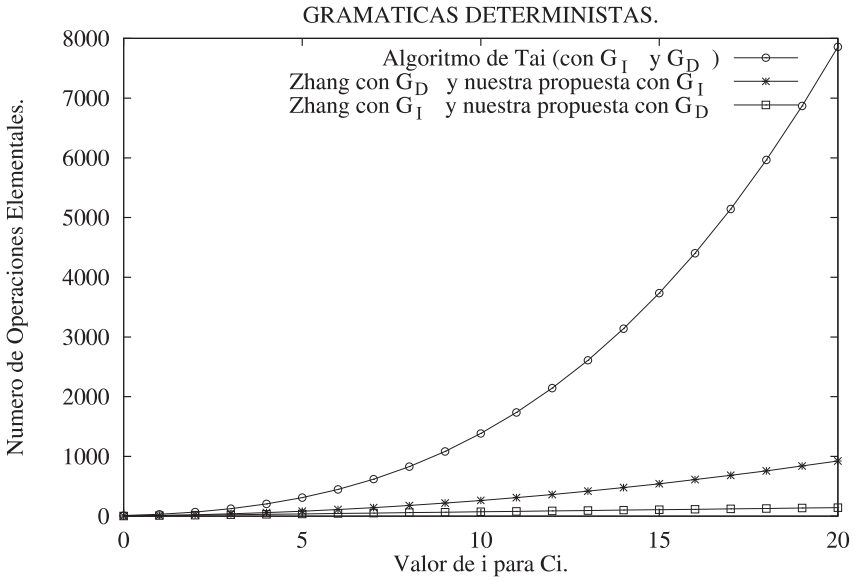


Figura 9.2: Resultados con las gramáticas deterministas.

En el caso determinista, los patrones han sido construidos a partir de la interpretación asociativa por la izquierda (resp. asociativa por la derecha) de las gramáticas $\mathcal{G}_I$ (resp. $\mathcal{G}_D$) de la figura 9.1, lo que nos permite evaluar el impacto de la orientación de los árboles en el rendimiento. Así, la figura 9.2 prueba la adaptación de nuestra propuesta (resp. del algoritmo de Zhang y Shasha [65]) a las derivaciones recursivas por la izquierda (resp. por la derecha), lo cual corrobora nuestras conclusiones. Estos tests también muestran la independencia del algoritmo de Tai [43] de la topología exhibida por las reglas gramaticales. Ello se debe a que esta propuesta es un algoritmo descendente que no utiliza el concepto de *raíces_I*, y no propone cálculos diferenciados para distancias entre árboles sin contexto y distancias entre bosques. De este modo, su complejidad no depende de la disposición de los nodos, sino únicamente de la profundidad de los árboles considerados.

En el caso no determinista, los patrones son construidos sobre la interpretación asociativa por la izquierda de la consulta, lo cual no es relevante puesto que las reglas en $\mathcal{G}_N$ son simétricas. El hecho de que las reglas, y por lo tanto los grafos Y-O resultantes, sean simétricos nos ofrece un marco neutro en el cual evaluar la eficiencia de las modificaciones propuestas para aprovechar la compartición de estructuras. En los tests con este tipo de bosques dato la topología del grafo Y-O no es relevante y no influye en la eficiencia del mecanismo de cálculo, como sí sucedía con las gramáticas anteriores.

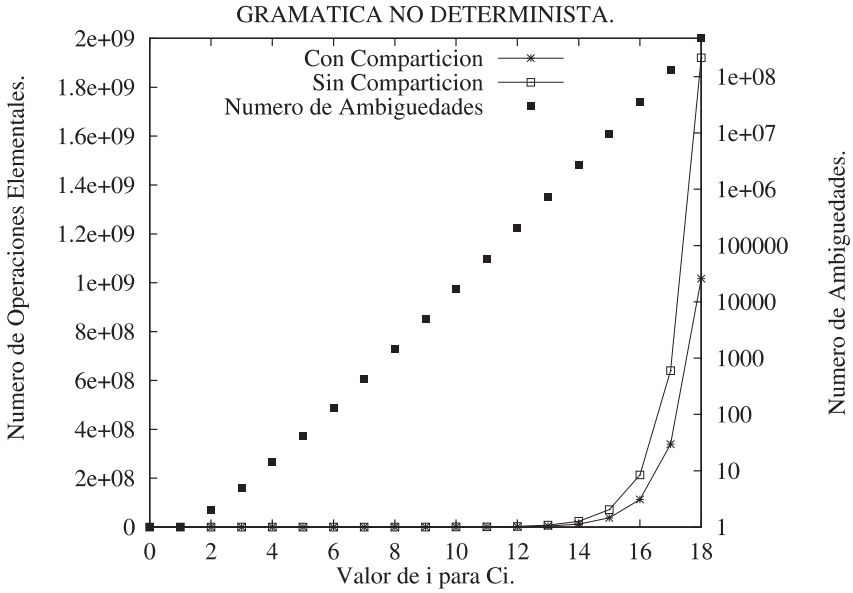


Figura 9.3: Resultados con la gramática no determinista.

Por lo tanto, emplearemos estos tests como medida para evaluar la ganancia en eficacia computacional debida a la compartición de cálculos en programación dinámica, como se muestra en la figura 9.3. En dicha figura se representa el número de operaciones elementales efectuadas en cada caso de prueba. Para cada cadena de entrada se realizaron dos ejecuciones, en una se utilizaron las modificaciones presentadas en capítulos anteriores y en la otra se contabilizó el número de operaciones elementales que fue necesario calcular sin emplear ninguna de dichas optimizaciones. Como se puede apreciar, el número de operaciones elementales calculadas con compartición es aproximadamente la mitad del número de operaciones que serían necesarias en el caso de que no se tuviera en cuenta la compartición estructural. Cabe señalar que en el caso de la gramática $\mathcal{G}_N$, al tratarse de una gramática muy simple, con pocas reglas y de pequeño tamaño, las posibilidades de compartición se limitan a subestructuras de talla muy reducida. Esto da lugar a que las opciones de compartición más complejas, esto es, la compartición dentro de una misma *raíz_D* o entre distintas *raíces_D*, tengan un peso relativamente reducido en el ahorro de cálculos global.

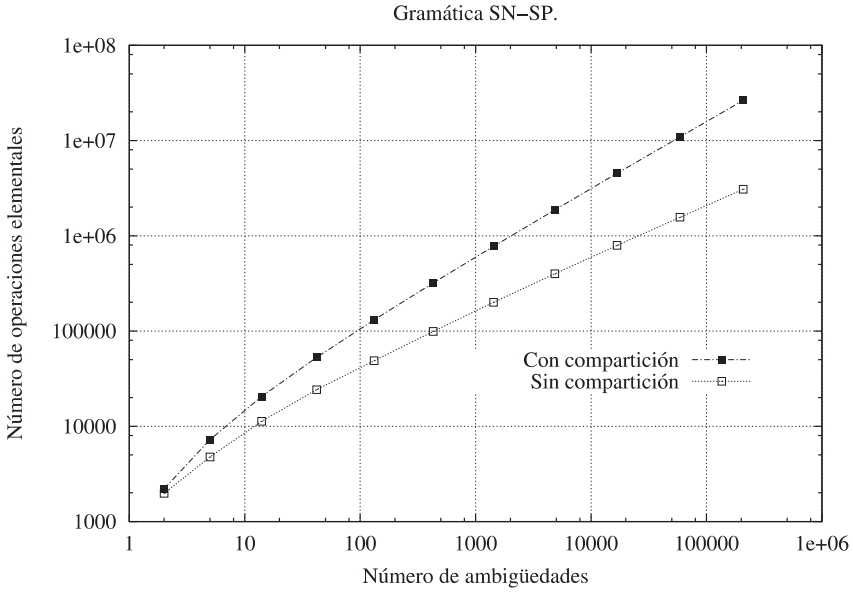


Figura 9.4: Número de operaciones con y sin compartición.

En el segundo tipo de tests de rendimiento que hemos realizado, se ha empleado una gramática simplificada que permite definir frases sencillas en español. Hemos utilizado la gramática $\mathcal{G}_{SN-SP}$, definida por la reglas de producción siguientes:

(1)	S	$\rightarrow$	$SN\ SV$	(6)	SV	$\rightarrow$	$verbo\ SN$
(2)	SN	$\rightarrow$	nom	(7)	SV	$\rightarrow$	$verbo\ SN\ SP$
(3)	SN	$\rightarrow$	$det\ nom$	(8)	SP	$\rightarrow$	$prep\ SN$
(4)	SN	$\rightarrow$	$det\ nom\ SP$				
(5)	SN	$\rightarrow$	$det\ nom\ SP\ SP$				

Como bosques dato hemos empleado los bosques de análisis que resultan de analizar sentencias de la forma *"Juan dió una tarta a (la amiga de)^i Maria"*, donde i representa el número de repeticiones de la subcadena *"la amiga de"*. En nuestro caso hemos utilizado un conjunto de valores consecutivos para i desde 1 hasta 11. Dado que la gramática $\mathcal{G}_{SN-SP}$ contiene la regla "$S \rightarrow det\ nom\ SP\ SP$", el número de árboles de análisis que genera, C_i, crece de forma exponencial con respecto al tamaño de la entrada. De hecho, dicho número de análisis ambiguos, C_i, viene dado por la misma fórmula que en el caso de $\mathcal{G}_N$. Esto nos permite evaluar nuestra propuesta en un entorno altamente ambiguo, a pesar de la relativa simplicidad de la gramática.

Para todas las cadenas de entrada hemos empleado el mismo árbol patrón, el que corresponde con el análisis determinista de la sentencia *"Juan dio una tarta a (la amiga de)^i Maria"*. Como en los experimentos anteriores hemos realizado dos ejecuciones para cada entrada, una empleando la reutilización de distancias basada en las estructuras compartidas y otra sin reutilizar cálculos. La figura 9.4 compara el número total de operaciones elementales que resultan para estos dos tipos de ejecuciones. En este caso, mostramos el número total de operaciones elementales con y sin compartición de cálculos en función del número de árboles ambiguos que tiene cada cadena de entrada.

Para facilitar la interpretación y mostrar más claramente el ahorro de cálculos producido por la compartición de estructuras empleamos escalas logarítmicas en ambos ejes de la gráfica 9.4. Como se puede apreciar, a medida que la longitud y el grado de ambigüedad de la cadena de entrada aumenta, el número de operaciones reutilizadas también es mayor. Este comportamiento se aprecia más claramente

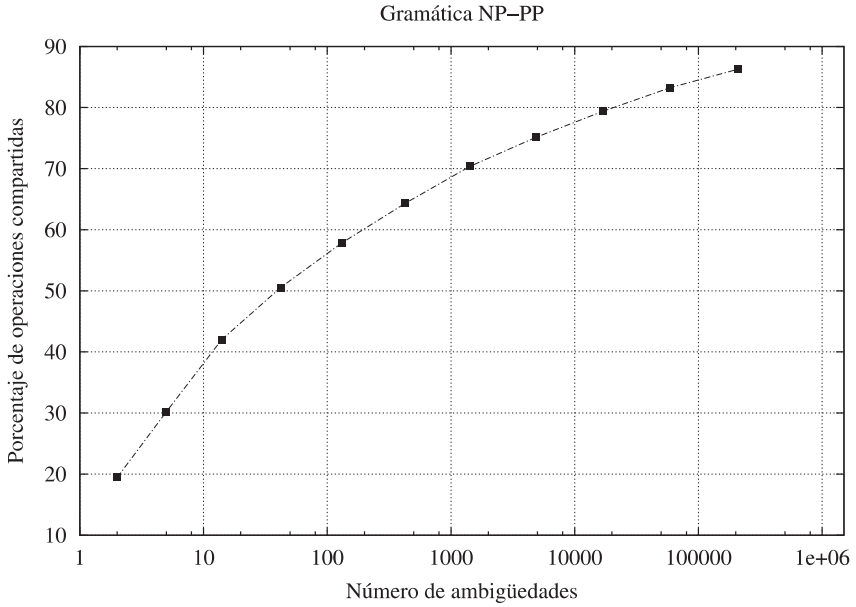


Figura 9.5: Porcentaje de operaciones elementales compartidas.

en la figura 9.5, en ella se representa el porcentaje de operaciones ahorradas como consecuencia de la compartición respecto al número de total de operaciones que sería necesario calcular de no emplear las optimizaciones. Vemos en esta gráfica que dicho porcentaje crece a medida que las cadenas de entradas se hacen más ambiguas. La razón de este comportamiento se debe a la propia naturaleza de la gramática y los ejemplos utilizados. Dado que el número de análisis posibles crece exponencialmente con el tamaño de la entrada, para las cadenas de entrada de mayor tamaño, el bosque compartido contiene un número de árboles extremadamente grande. Por lo tanto, dependiendo de la topología del bosque compartido, las posibilidades de compartir una determinada estructura pueden mayores, con lo que aumenta también el grado de reutilización de las distancias calculadas. Es más, tenemos que en bosques muy ambiguos, como en este caso, ocurre que cualquier subestructura estará, en general, compartida una o más veces por distintos árboles contenidos en el bosque. En lo que toca a nuestras propuestas respecto a la reutilización de cálculos, este comportamiento pone de manifiesto su correcto funcionamiento en casos de elevada ambigüedad.

9.2. Evaluación de la aplicación de los símbolos VLDC

El segundo tipo de evaluación que hemos realizado está destinado a ilustrar la relación entre el esfuerzo computacional del proceso de reconocimiento de patrones y el esfuerzo aplicado a la hora de especificar el árbol patrón. En concreto, nos hemos centrado en estudiar como afectan las diferencias entre los dos símbolos VLDC introducidos en el capítulo 6, el |-VLDC y el ∧-VLDC, al mecanismo de reconocimiento. También estamos interesados en mostrar el impacto de la compartición de estructuras en el número de cálculos de distancias efectuado, especialmente para el caso de los ∧-VLDC.

Para interpretar los resultados experimentales necesitamos un entorno formal. Para ello, siguiendo la aproximación expuesta en [52], hemos definidos una serie de propiedades que estimamos que son de interés en un algoritmo de reconocimiento de patrones como el que aquí tratamos.

La primera propiedad a considerar es el grado de *abstracción* obtenido durante el proceso de reconocimiento. Definimos la *abstracción* como el ratio entre el número de nodos que han participado en sustituciones-VLDC en relación al número total de nodos del árbol dato considerado. Intuitivamente,

$$\textbf{abstracción} = \frac{\text{número de nodos en sustituciones-VLDC}}{\text{número de nodos totales}}$$

$$\textbf{esfuerzo} = \text{número de operaciones elementales}$$

$$\textbf{rendimiento} = \frac{\text{número de operaciones VLDC utilizadas}}{\text{número total de operaciones VLDC calculadas}}$$

$$\textbf{calidad} = \frac{\text{distancia final}}{\text{número de nodos totales}}$$

Figura 9.6: Propiedades del proceso de reconocimiento de patrones con VLDC.

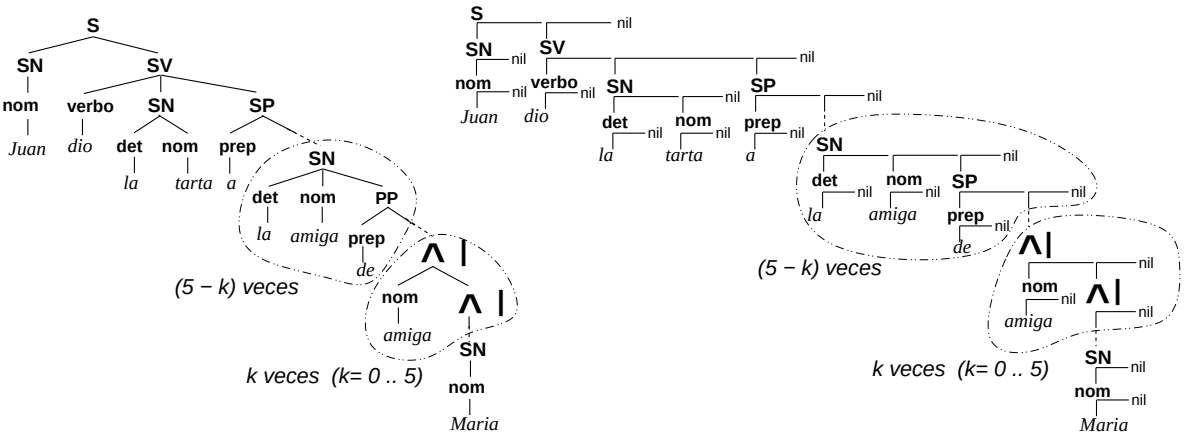


Figura 9.7: Arboles patrón empleados en los tests con VLDC.

la abstracción mide la entidad de las regiones cubiertas por los símbolos VLDC. Si consideramos que un mayor nivel de abstracción supone, *a priori*, un menor esfuerzo en la especificación del patrón, tenemos un criterio simple para cuantificar el esfuerzo empleado en la construcción del patrón. En efecto, la posibilidad de incluir símbolos VLDC hace que sea necesario un menor conocimiento de la estructura sintáctica empleada y que se pueda evitar el tener que especificar completamente determinados detalles estructurales que pudieran no ser relevantes

La segunda propiedad que hemos tenido en cuenta es el *esfuerzo*. En este caso, para medir el esfuerzo computacional del algoritmo de reconocimiento simplemente hemos contabilizado el número operaciones elementales necesarias para obtener la distancia final, como hicimos en la sección anterior. La tercera propiedad considerada es el *rendimiento*. Definimos el rendimiento del proceso de reconocimiento con símbolos VLDC como la relación entre el número de operaciones elementales donde estén implicados símbolos VLDC que han contribuido a la obtención de la distancia final y el número total de dichas operaciones con VLDC calculadas por el algoritmo. En este caso entendemos como operaciones con símbolos VLDC todos los cálculos de distancias mínimas adicionales que son necesarios para tratar los subárboles patrón etiquetados con |-VLDC ó ∧-VLDC. Asimismo, incluimos también los cálculos de las distancias auxiliares *dbs* necesarias para tratar los símbolos ∧-VLDC. Esta medida de rendimiento nos permite evaluar el aprovechamiento que realmente se hace de los cálculos que implican a símbolos VLDC.

Finalmente, para tener una indicación de la *calidad* del proceso de reconocimiento, empleamos un ratio entre la distancia final obtenida por el algoritmo y el número total de nodos del árbol incluido en el bosque compartido que originó dicha distancia. La figura 9.6 resume la definición de estas cuatro medidas.

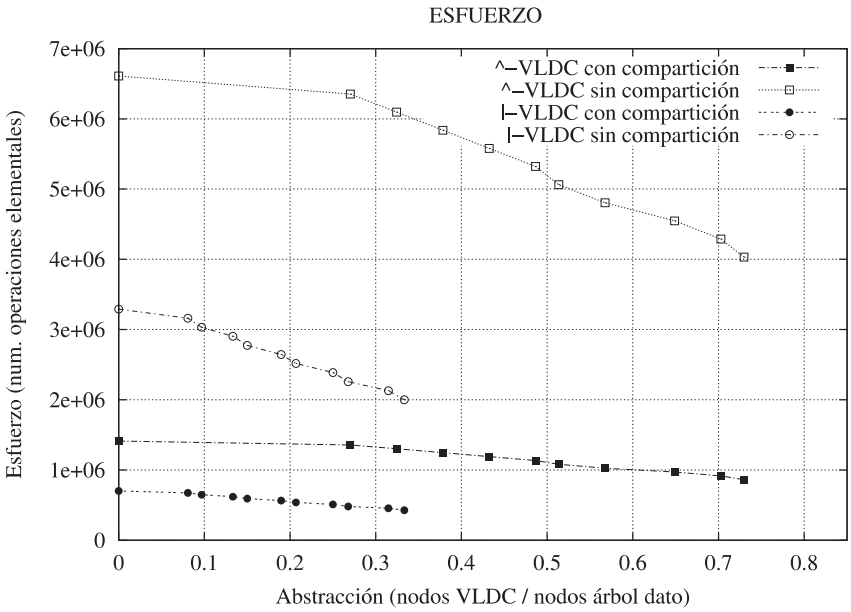


Figura 9.8: Esfuerzo con la gramática $\mathcal{G}_{SN-SP}$.

En nuestra evaluación hemos empleado de nuevo la gramática $\mathcal{G}_{SN-SP}$ para frases simples en español, mostrada anteriormente. En este caso consideramos un bosque dato fijo y hacemos variar a los árboles patrón utilizados. Como bosque dato empleamos el grafo Y-O generado por ICE para la cadena de entrada *"Juan dió una tarta a (la amiga de)^i Maria"*, con $i = 7$. El conjunto de patrones utilizados se ha construido a partir del árbol de análisis determinista de la sentencia *"Juan dió una tarta a (la amiga de)*[5] *Maria.* Utilizamos este árbol directamente para construir un primer árbol patrón sin VLDC. Los demás patrones se van generando a partir de él, haciéndolos cada vez más vagos y menos específicos mediante la adición de símbolos VLDC, tal y como se muestra en la figura 9.7. Para poner de manifiesto la diferencia entre los símbolos VLDC considerados, hemos generado dos conjuntos de árboles patrón, uno que contiene únicamente ∧-VLDC y otro que sólo tiene |-VLDC.

Los resultados obtenidos se muestran en las figuras 9.8, 9.9 y 9.10. En las tres figuras representamos en el eje de las abscisas el grado de abstracción de los patrones. Como es de esperar, los valores de abstracción crecen a medida que se emplean patrones menos específicos que incluyan mayor número de símbolos VLDC. También cabe señalar que los patrones que sólo poseen símbolos |-VLDC alcanzan valores de abstracción mucho menores que en el caso de patrones con sólo ∧-VLDC. Este comportamiento es consecuencia del tipo de sustituciones-VLDC que se realizan con los |-VLDC. Dichas sustituciones incluyen únicamente a los nodos de un camino del árbol dato, pero no sus subárboles completos. En las figuras 9.8 y 9.9 también se puede apreciar la reducción del coste computacional del proceso de reconocimiento debida a la utilización de las estructuras compartidas generadas por el analizador.

En la figura 9.8 se muestran los resultados obtenidos en cuanto al esfuerzo para diferentes patrones con grados de abstracción crecientes. En nuestros experimentos se produce una reducción en el esfuerzo a medida que crece la abstracción. Esto es debido, principalmente, al hecho de que los patrones más abstractos contienen un número de nodos menor que los más específicos. En el caso de patrones que únicamente tienen símbolos |-VLDC, el esfuerzo es menor debido a que el algoritmo no tiene que calcular todas las distancias auxiliares *dbs* que necesitan los ∧-VLDC. Como consecuencia, en general, el uso de símbolos ∧-VLDC multiplica por dos el número total de operaciones elementales que es necesario calcular para obtener la distancia final.

Desde un punto de vista intuitivo, estos resultados indican que el coste computacional está en relación inversa con el esfuerzo que se debe aplicar en la construcción de los patrones. Como hemos indicado

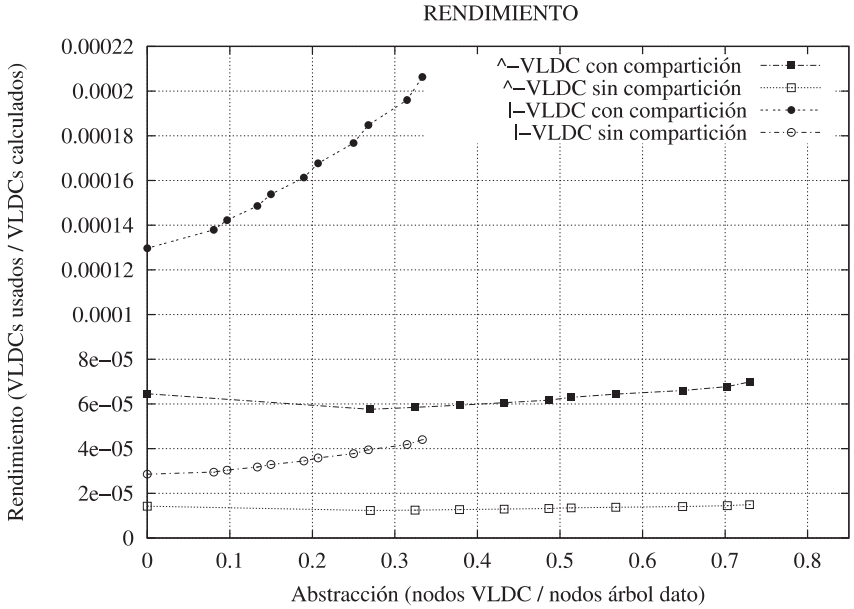


Figura 9.9: Rendimiento con la gramática $\mathcal{G}_{SN-SP}$.

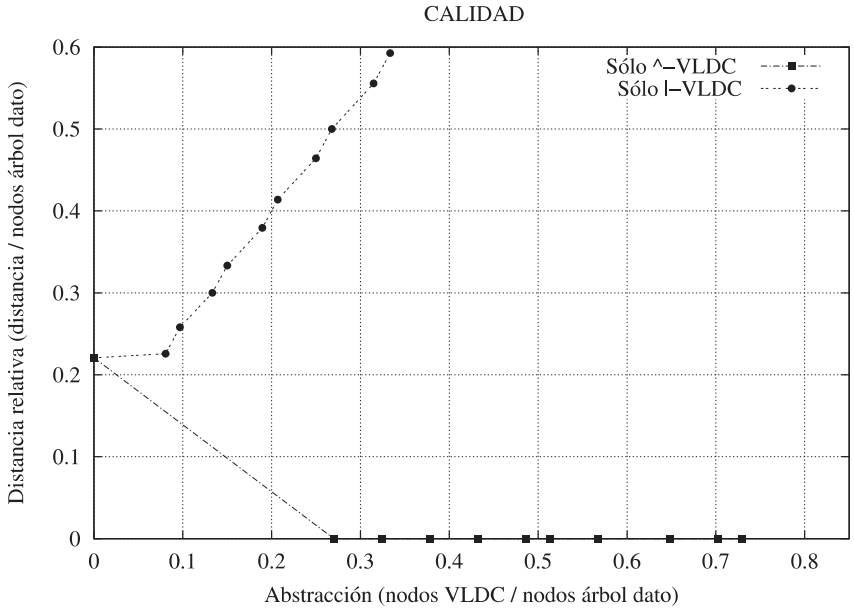


Figura 9.10: Calidad con la gramática $\mathcal{G}_{SN-SP}$.

anteriormente, en general, los patrones menos específicos requerían un menor esfuerzo a la de construirlos, bien por parte un usuario o bien por parte de un mecanismo que los genere automáticamente. Así, los patrones más abstractos no exigen por parte del usuario un conocimiento en profundidad de la gramática empleada ni de las estructuras sintácticas generadas. Además, permiten no tener que especificar los patrones completamente, eliminando detalles del árbol que no sean de interés para la aplicación.

En la figura 9.9 podemos ver como los valores más altos de rendimiento se obtiene para los patrones con símbolos |-VLDC. De nuevo, este comportamiento es consecuencia del hecho de que para estos patrones

no sea necesario calcular las distancias auxiliares *dbs*. En este caso tenemos que aunque el número de operaciones VLDC realmente utilizadas es menor, el número total de cálculos con VLDC es mucho menor que en el caso de los patrones con ∧-VLDC, al no tener que calcular todos los valores *dbs* adicionales.

Finalmente, en la figura 9.10 mostramos la influencia de los distintos tipos de patrones en el valor de la distancia final obtenida por el algoritmo en nuestros tests. Al tratarse de experimentos con una gramática y unos patrones artificiales, es difícil realizar una interpretación de los resultados. Máxime si tenemos en cuenta que dicha interpretación dependería, a su vez, del tipo de aplicación concreta en la cual se desease emplear estas técnicas de reconocimiento de patrones. En efecto, determinados tipos de aplicaciones pueden necesitar una correspondencia muy precisa con los patrones, mientras que otras aplicaciones requerirán justo el comportamiento contrario.

Aún con estas limitaciones, el aspecto más interesante de nuestros resultados es el comportamiento que obtenemos con los patrones que únicamente contienen |-VLDC. En este caso, a medida que el patrón se hace menos específico, la distancia aumenta. La razón de este comportamiento es que, a medida que los patrones con |-VLDC se hacen más generales, hay menos nodos especificados en el patrón. Por lo tanto, al identificar la mejor correspondencia, nos encontramos con que es necesario incluir operaciones de inserción adicionales para esos nodos no especificados. En patrones con símbolos ∧-VLDC, las sustituciones-VLDC ya incluyen esos nodos que sería necesario insertar, haciendo que la distancia final sea cero. Desde un punto de vista intuitivo, esto significa que los patrones con |-VLDC son más inflexibles que aquellos con ∧-VLDC y más sensibles a la topología de los bosques dato. Por lo tanto, el comportamiento de las sustituciones-VLDC para los |-VLDC debe ser tenido muy en cuenta cuando se construyen patrones que incluyen dicho símbolo.

Capítulo 10
Conclusiones y trabajo futuro

Terminamos con una breve recapitulación del trabajo realizado en esta tesis doctoral. Presentaremos, en primer lugar, un recorrido por los distintos capítulos de la memoria. Mostraremos los resultados y conclusiones de cada uno de los capítulos introducidos, junto con las aportaciones más importantes de nuestro trabajo. En una segunda parte del mostramos las posibles lineas futuras de investigación que surgen a partir del trabajo realizado en esta tesis. Nos centraremos en la descripción de las posibilidades que ofrece en el campo de la RI y de la EI la integración del análisis sintáctico de los lenguajes naturales con las técnicas de reconocimiento de patrones en bosques compartidos.

10.1. Resultados obtenidos y conclusiones

En esta memoria de tesis doctoral hemos realizado un estudio de los algoritmos de reconocimiento de patrones en árboles y hemos adaptado este tipo de técnicas al caso concreto de los bosques de análisis compartidos generados por el sistema ICE [56]. La motivación general de esta investigación deriva de la importancia que tienen las estructuras arbóreas en múltiples áreas de la informática, así como de las ventajas que ofrecen las técnicas de reconocimiento de patrones como mecanismo descriptivo para el acceso a datos estructurados.

En concreto, nuestro trabajo se enmarca en el campo de la aplicación de técnicas de reconocimiento de patrones en sistemas de recuperación y extracción de información basados en la utilización de técnicas de PLN. La motivación de nuestra aproximación se orienta al estudio las posibilidades del uso de estructuras sintácticas [11, 41] como elementos clave para la descripción y el acceso a los documentos relevantes para las necesidades de información de un usuario. En un sistema de este tipo, el mecanismo más natural para la identificación de los elementos relevantes para una consulta de un usuario estará basado en algún tipo de técnica de reconocimiento de patrones en conjuntos de datos estructurados. En el caso que nos interesa, la recuperación de documentos en lenguaje natural, debemos enfrentarnos al problema de la ambigüedad de estos lenguajes, que da lugar a que una misma frase pueda tener más de una estructura sintáctica. En nuestro trabajo hemos empleado el generador de analizadores ICE [56] que proporciona una representación compacta y sin redundancias de los múltiples análisis de una cadena de entrada en forma de bosque compartido [6].

Hemos realizado un estudio detallado de los algoritmos clásicos de reconocimiento de patrones en árboles. En concreto, hemos presentado y discutido las propuestas de Selkow [37], Tai [43] y Zhang y Shasha [65, 66] en los capítulos 4 y 5. Nuestro estudio se ha centrado tanto en aspectos relativos a la eficiencia computacional de las soluciones propuestas por estos autores, como en sus capacidades en cuanto a potencia expresiva. De entre estas alternativas la más adecuada para nuestro propósito es la de Zhang y Shasha. No sólo porque sea la más eficiente computacionalmente, sino por ser la más flexible y la que mejor se adapta a la estrategia de análisis empleada en el analizador ICE. En efecto, la técnica de reconocimiento de patrones de Selkow utiliza un concepto de distancia demasiado restrictivo que no se ajusta a las necesidades del tipo de aplicación que pretendemos desarrollar. Por su parte, la propuesta de Tai es muy costosa computacionalmente y difícilmente ampliable con nuevas capacidades que

permitan construir un mecanismo de reconocimiento de patrones flexible y útil. En cambio, el algoritmo de Zhang y Shasha al ser ascendente facilita la integración con los analizadores generados por ICE, que realizan un análisis sintáctico también ascendente. Por otra parte, el algoritmo básico puede ampliarse con relativa facilidad para incorporar técnicas de reconocimiento de patrones avanzadas, como se muestra en el capítulo 6 con la inclusión de los símbolos VLDC [66] en el árbol patrón.

Para realizar eficazmente la integración del algoritmo de Zhang y Shasha en ICE [56] hemos llevado a cabo un estudio del tipo de representación sintáctica utilizada por ICE [6, 28], que se muestra en el capítulo 7. También se han identificado los factores que determinan la compartición de estructuras en el análisis de sentencias ambiguas y cómo afectan al mecanismo de reconocimiento de patrones. Esto nos ha permitido modificar la propuesta original de Zhang y Shasha para adaptarla a las peculiaridades de las estructuras sintácticas generadas por ICE, como se muestra en [47, 48, 51] y en el capítulo 8. Estas modificaciones han sido de dos tipos:

- Hemos realizado un cambio en el tipo de ordenación empleado por el algoritmo clásico, utilizando un postorden inverso. Para ello ha sido necesario, además, adaptar los conceptos y cálculos básicos de la propuesta original al nuevo tipo de ordenación. También ha sido necesario definir el comportamiento del algoritmo para los nuevos tipos de nodos que emplea ICE. En concreto, se ha definido el cálculo de distancias con nodos-O y en el caso de los nodos adicionales introducidos durante la binarización implícita de las reglas realizada por ICE.

- La integración se ha realizado tratando de aprovechar al máximo la compartición de estructuras ofrecida por ICE, para evitar realizar cálculos redundantes. Hemos realizado un estudio de las posibilidades de compartición en base a las posiciones relativas de los nodos-O y de las subestructuras comunes presentes en el bosque compartido. Como resultado de ello, distinguimos los dos tipos de compartición de cálculos mostrados en el capítulo 8: compartición dentro de una misma *raíz_D* y compartición entre distintas *raíces_D*.

En el caso del reconocimiento de patrones en bosques compartidos se sigue manteniendo la distinción entre dos tipos de cálculos, distancias entre bosques y distancias entre árboles, que existía en el algoritmo original de reconocimiento de patrones en árboles de Zhang y Shasha. Esta distinción tiene una serie de ventajas de cara a la aplicación práctica de este algoritmo, puesto que nos ofrece dos alternativas de mejora del algoritmo. Por un lado, se pueden añadir capacidades de reconocimiento de patrones más sofisticadas que sólo deberán ser tenidas en cuenta durante el cálculo de distancias entre árboles. Ejemplos de este tipo de mejoras son el empleo de símbolos VLDC o también el caso de la inclusión de variables de extracción. Por otro lado, las posibles optimizaciones derivadas de la existencia de estructuras compartidas, son aplicadas, como mostramos en el capítulo 8, en los cálculos de distancias entre bosques. Es en este tipo de cálculos de distancias donde se concentra la mayor parte del coste computacional del proceso de reconocimiento de patrones. Y, por lo tanto, es en ellos donde se han centrado nuestras optimizaciones, de modo que no afecten a otros aspectos del algoritmo. debemos señalar que

Por último, en el capitulo 9 hemos realizado una evaluación de las ventajas aportadas por nuestra aproximación en cuanto a la mejora del rendimiento. Para ello hemos realizado una serie de baterías de tests empleando un conjunto de gramáticas altamente no deterministas. Si bien estas gramáticas no se corresponden exactamente con las que podríamos encontrar en las aplicaciones prácticas, si son una buena herramienta para realizar la evaluación de las modificaciones realizadas sobre el algoritmo. Los resultados obtenidos ponen de manifiesto una mejora importante del rendimiento en el caso del reconocimiento de patrones en bosques compartidos. Esto hace pensar que el comportamiento del algoritmo en aplicaciones reales puede ser admisible en determinados entornos, puesto que el tipo de gramáticas y bosques manejados nunca serían tan complejos y adversos como los empleados en nuestros tests.

En lo que respecta a las técnicas y optimizaciones propuestas, debemos señalar que estas no se limitan al problema del reconocimiento de patrones sobre bosques compartidos. En [53] mostramos como se puede aplicar el mismo esquema de cálculo, y las optimizaciones basadas en la compartición estructural, al problema de la identificación de subestructuras comunes en bosques compartidos. Este es un problema relacionado con el cálculo de la distancia de edición entre árboles. En este caso, el objetivo es determinar el

tamaño de la mayor subestructura común a dos árboles dados, permitiendo un número finito de diferencias o errores entre ellos. Siguiendo la misma técnica descrita en esta memoria se puede extender el algoritmo mostrado en [61] para permitir el trabajo sobre bosques compartidos, con un aprovechamiento máximo de la compartición de estructuras.

En cuanto a nuestro trabajo sobre reconocimiento de patrones en bosques compartidos, el principal problema de la aplicación práctica de esta técnica en sistemas de RI y de EI es que su coste computacional sigue siendo muy superior al coste de las aproximaciones clásicas basadas en el reconocimiento de cadenas de caracteres. Debemos tener en cuenta que nuestra aproximación integra un proceso análisis sintáctico y un proceso de reconocimiento de patrones. Esto hace que, para entornos de carácter general y con volúmenes de datos muy grandes, no sea suficientemente eficiente y su utilidad práctica resulte escasa. Sin embargo, como veremos en el apartado de trabajos futuros, pensamos que esta técnica puede ser prometedora en entornos de RI o EI restringidos, o planteándola una fase adicional de filtrado y mejora de los resultados obtenidos por un sistema de RI o EI convencional.

10.2. Desarrollos futuros

Como hemos mencionado, nuestro trabajo se enmarca dentro de una línea de investigación centrada en el estudio de las posibilidades de aplicación de técnicas de PLN en entornos de RI. Aunque hemos realizado una optimización de los algoritmos clásicos [65, 66] con la intención de permitir un procesamiento eficiente de los árboles de análisis de sentencias ambiguas, el coste computacional de nuestra aproximación [51, 50] sigue siendo demasiado alto como para poder ser aplicado directamente en sistemas de RI sin restricciones.

Sin embargo, creemos que si es factible su utilización en aplicaciones con menor volumen de datos. Por ejemplo, podría emplearse en entornos limitados, como por ejemplo bases de datos de textos legales o médicos. En estos casos si puede ser relevante considerar relaciones sintácticas complejas entre las palabras que componen los documentos, en lugar de tratarlos como simples conjuntos de palabras claves. Otra posibilidad es emplear el reconocimiento de patrones como una fase adicional que mejore los resultados de un sistema de RI convencional. En este caso se utilizaría el reconocimiento de patrones para realizar un filtrado previo de los documentos que serían presentados al usuario.

Veremos con más detalle como podemos emplear estas técnicas de reconocimiento de patrones para mejorar los procesos de indexación y consulta en sistemas de RI [31, 11]. Revisaremos también otras posibles líneas de aplicación de este tipo de técnicas.

Indexación basada en reconocimiento de patrones

En este caso, la idea básica es realizar la indexación de términos multipalabra complejos [11, 16, 15], que incluyan a dos o más palabras del documento, en lugar de la indexación de palabras simples. Para ello se deberá disponer de un conjunto de árboles patrón, que posiblemente incluyan símbolos VLDC, con variables de extracción asociadas a algunas de sus hojas. Generalmente esta aproximación no será posible en entornos no limitados, puesto que las posibles estructuras de interés serían demasiado numerosas. En cambio, si sería factible si se aplicase sobre documentos extraídos de un entorno restringido, como por ejemplo textos médicos o legales, o incluso manuales técnicos. En esos casos si sería posible identificar un conjunto razonablemente pequeño de estructuras sintácticas que representen relaciones sintácticas y semánticas de interés para ese dominio en particular.

Cada nuevo documento que vaya a ser indexado, será procesado por una versión de ICE que realiza un análisis sintáctico robusto [57, 54, 55]. De ese modo, el procedimiento sería menos sensible a los errores gramaticales presentes en el documento o a la falta de cobertura sintáctica de la gramática empleada. Dicho analizador robusto incluirá el mecanismo de reconocimiento de patrones descrito en esta memoria. Como se muestra en el esquema de la figura 10.1, el conjunto de árboles patrón se compararía con los bosques compartidos asociados a las sentencias presentes en cada documento. De cada patrón que encaje con alguna parte de dichas sentencias se recuperarían los fragmentos que estuvieran asociados con sus

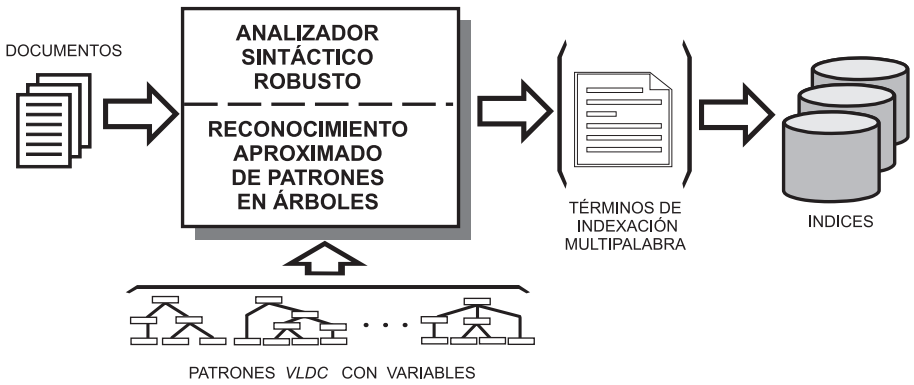


Figura 10.1: Selección de términos de indexación complejos basada en reconocimiento aproximado de patrones.

variables de extracción y se generarían con ellos los términos compuestos que se almacenarían en el índice.

Filtrado de los resultados de consultas utilizando reconocimiento de patrones

Como mencionamos anteriormente, en general, no es posible aplicar directamente al algoritmo de reconocimiento a una colección muy grande de documentos, como sucede en el caso de los sistemas de recuperación convencionales. Sin embargo, puede pensarse en reducir ese conjunto de documentos utilizando en primer lugar un sistema convencional que proporcione una lista preliminar de documentos supuestamente relevantes.

En este punto se podría pensar en utilizar el reconocimiento de patrones para identificar los documentos más prometedores de dicha lista y reordenarla de acuerdo a las distancias que resultaran del algoritmo de comparación. Esta idea se ilustra en el esquema de la figura 10.2. En primer lugar sería necesario transformar la consulta original del usuario en un árbol patrón o en un conjunto de árboles, probablemente incluyendo símbolos VLDC para que la recuperación fuese más flexible. Una vez que se dispone del patrón o patrones a buscar, se analizaría, como en el caso anterior, cada documento de la lista preliminar con una versión robusta de ICE que integrase nuestro algoritmo de comparación en bosques. En función de las distancias obtenidas se reordenaría la lista de documentos, generando la nueva lista de documentos relevantes que se presentaría al usuario.

La idea de utilizar el análisis sintáctico de los documentos en conjunción con un algoritmo de comparación de estructuras ya fue propuesta por Smeaton en [42]. Sin embargo, los resultados obtenidos por este autor no fueron todo lo buenos que cabría esperar. El principal problema de la aproximación de Smeaton es que no utilizaba mecanismos de reconocimiento aproximado, sino un método *ad hoc* definido para este problema en concreto [41, 40]. Además no realizaba ningún preprocesamiento específico sobre la consulta, simplemente se analizaba sintácticamente sin siquiera diferenciar si era una consulta en forma afirmativa o interrogativa. Por ello, resultaba un método de comparación poco flexible que, unido a un analizador poco robusto frente las agramaticalidades de la entrada, daba lugar a un sistema con una eficacia muy pobre.

Nuestra aproximación pretendería evitar estos problemas mediante el uso de un analizador sintáctico robusto y de técnicas de reconocimiento aproximado de patrones. Sin embargo, el mayor escollo a la hora de llevar a cabo una propuesta como la mostrada en la figura 10.2, sería el de definir el modo en que se deberían construir los árboles patrón. Desde nuestro punto de vista, no es sencillo diseñar un mecanismo suficientemente poderoso como para generar, a partir de la consulta en lenguaje natural unos árboles patrón que la representaran adecuadamente, pero que que a la vez no sean excesivamente inflexibles. Este es un tema abierto con interesantes implicaciones desde el punto de vista lingüístico y computacional.

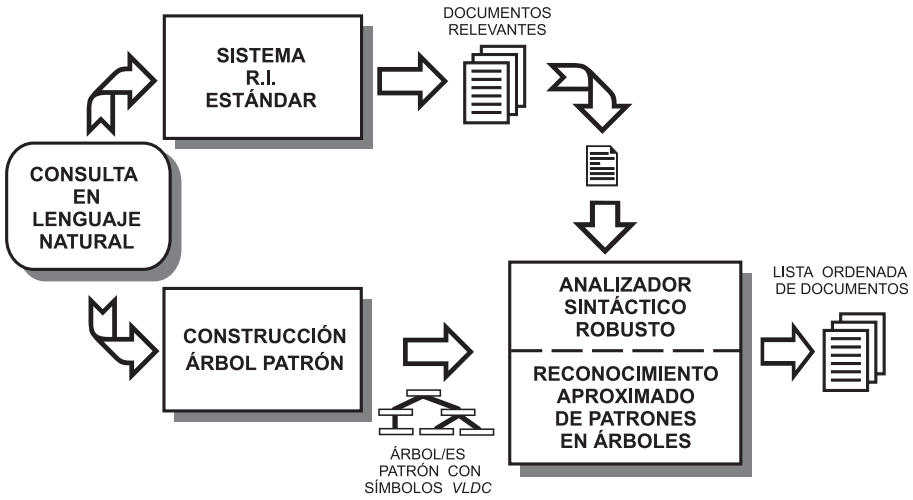


Figura 10.2: Reordenación de documentos relevantes basada en reconocimiento aproximado de patrones.

Otras líneas de aplicación

Por último citaremos dos aplicaciones de estas técnicas de reconocimiento de patrones en bosques de análisis compartido que se salen del campo de la RI, pero que representan futuras líneas de investigación interesantes.

En primer lugar, se puede pensar en emplear el reconocimiento de patrones en árboles como una técnica para evaluar la proximidad semántica entre dos frases en lenguaje natural. Obviamente, no basta sólo con comparar sus estructuras sintácticas y obtener un valor de distancias. Es necesario, además, evaluar y tener en cuenta la proximidad semántica entre las palabras que forman las sentencias. De esta forma será posible definir métricas que combinen de forma adecuada las dos medidas de similaridad; por una parte, el grado de sinonimia entre las palabras y, por otra, la proximidad entre las estructuras sintácticas de dichas frases en lenguaje natural.

El otro campo de aplicación de estas técnicas de reconocimiento de patrones es su utilización como guías durante el proceso de análisis sintáctico. La aproximación que hemos seguido en nuestro trabajo es la integrar el algoritmo de comparación de bosques en el seno del analizador ICE. Una extensión natural es intentar aprovechar los valores de las distancias parciales que se van calculando. Dichas distancias pueden utilizarse para definir heurísticas que permitan desechar determinadas alternativas de análisis o incluso abortar el análisis de una sentencia, una vez se concluya que no va a ofrecer un resultado próximo al árbol patrón que se desea. Esta clase de heurísticas pueden ser muy útiles en las aplicaciones de indexación y consulta que hemos descrito con anterioridad. En dichos tipos de aplicaciones tenemos un modelo de la estructura sintáctica que deseamos obtener, y sería posible usar las distancias parciales calculadas para desechar análisis que, *a priori*, no sean suficientemente prometedores. Además, en el caso de utilizar un analizador robusto, el número de alternativas de análisis a estudiar tenderá normalmente a aumentar con respecto a un analizador no robusto. Por lo tanto, el disponer de un mecanismo para desechar algunas de esas alternativas permitirá mejorar enormemente la eficiencia general del proceso.

Bibliografía

[1] A.V. Aho and J.D. Ullman. *The Theory of Parsing, Translation and Compiling*, volume 1-2. Prentice-Hall, Englewood Cliff, New Jersey, U.S.A., 1973.

[2] S.W.K. Tjiang A.V. Aho, M. Ganapathi. Code generation using tree matching and dynamic programming. *ACM Trans. Prog. Lang. and Systems*, 11(4):491–516, 1989.

[3] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[4] François Bancilhon, David Maier, Yehoshua Sagiv, and Jeffrey D. Ullman. Magic sets and other strange ways to implement logic programs (extended abstract). In ACM, editor, *PODS '86. Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 24–26, 1986, Cambridge, MA*, pages 1–15, New York, NY 10036, USA, 1986. ACM Press.

[5] R.E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, U.S.A., 1957.

[6] S. Billot and B. Lang. The structure of shared forest in ambiguous parsing. In *Proc. 27th Annual Meeting of the ACL*, 1989.

[7] Noam Chomsky. On certain formal properties of grammars. *Information and Control*, 1:91–112, 1958.

[8] Nicolas Duncan David T. Barnard, Gwen Clarke. Tree-to-tree correction for document trees. technical report 95-372, 1995.

[9] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, Febrero 1970.

[10] Jay Earley. *An Efficient Context-Free Parsing Algorithm*. PhD thesis, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, PA, 1968.

[11] J.L. Fagan. *A Comparison of Syntactic and Non-Syntactic Methods for Information Retrieval*. PhD thesis, Department of Computer Science, Cornell University, 1987.

[12] Michael A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, Reading, 1 edition, 1978.

[13] C. M. Hoffmann and M. J. O'Donnell. Pattern matching in trees. *Journal of the ACM*, 29:68–95, 1982.

[14] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, N. Reading, MA, 1980.

[15] Christian Jacquemin. Improving automatic indexing through concept combination and term enrichment. In *COLING-ACL*, pages 595–599, 1998.

[16] Christian Jacquemin, Judith L. Klavans, and Evelyne Tzoukermann. Expansion of multi-word terms for indexing and retrieval using morphology and syntax. In Philip R. Cohen and Wolfgang Wahlster, editors, *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 24–31, Somerset, New Jersey, 1997. Association for Computational Linguistics.

[17] T. Kasami. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, Massachusetts, 1965.

[18] Boris Katz. Using english for indexing and retrieving. Technical Report AIM-1096, Massachusetts Institute of Technology, 1988.

[19] Martin Kay. Algorithm schemata and data structures in syntactic processing. In Barbara J. Grosz, Karen Sparck Jones, and B. L. Webber, editors, *Readings in Natural Language Processing*, pages 35–70. M. Kaufmann, Los Altos, CA, 1986. Originally published as a Xerox PARC technical report CSL-80-12, 1980.

[20] Peka Kilpelainen. *Tree Matching Problems with Applications to Structured Text Databases*. PhD thesis, Dept. of Computer Science, University of Helsinki, 1992.

[21] Philip Klein, Thomas B. Sebastian, and Benjamin B. Kimia. Shape matching using edit-distance: an implementation. In *Symposium on Discrete Algorithms*, pages 781–790, 2001.

[22] Philip Klein, Srikanta Tirthapura, Daniel Sharvit, and Benjamin B. Kimia. A tree-edit-distance algorithm for comparing simple, closed shapes. In *Symposium on Discrete Algorithms*, pages 696–704, 2000.

[23] Knuth. *The Art of Computer Programming, Vol. 1.* Addison-Wesley, Reading, 1973.

[24] S.R. Kosaraju. Efficient tree pattern matching. In *Proc. 30th annual IEEE Symp. on Foundations of Computer Science, FOCS'89*, pages 178–183, 1989.

[25] C. H. A. Koster. A Technique for Parsing Ambiguous Languages. In D. Siefkes, editor, *GI – 4. Jahrestagung*, volume 26 of *Lecture Notes in Computer Science*, Berlin, Germany, 1975. Springer-Verlag.

[26] Gad M. Landau, Uzi Vishkin, and Ruth Nussinov. Fast alignment of dna and protein data sequences. Technical Report CS-TR-2199, UMIACS-TR-89-20, Univ. of Maryland, Inst. for Adv. Computer Studies, Dept. of Computer Science, 1989.

[27] B. Lang. Complete evaluation of Horn Clauses, an automata theoretic approach. Technical Report 913, INRIA, Rocquencourt, France, 1988.

[28] B. Lang. Towards a uniform formal framework for parsing. In M. Tomita, editor, *Current Issues in Parsing Technology*, pages 153–171. Kluwer Academic Publishers, 1991.

[29] Bernard Lang. Deterministic techniques for efficient non-deterministic parsers. In Jacques Loeckx, editor, *Automata, Languages and Programming*, volume 14 of *Lecture Notes in Computer Science*, pages 255–269. Springer Verlag, August 1974.

[30] V. Levenshtein. Binary codes capable of correcting spurious insertions and deletions. *Problems fo Information Transmition*, 1((1)):8–17, 1965.

[31] Douglas P. Metzler and Stephanie W. Haas. The constituent object parser: syntactic structure matching for information retrieval. *ACM Transactions on Information Systems (TOIS)*, 7(3):292–316, 1989.

[32] Gonzalo Navarro. *Approximate text searching*. PhD thesis, Dept. of Computer Science, University of Chile, 1992.

[33] S.Ñeedleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequences of proteins. *Journal of Molecular Biology*, 48((1)):444–453, 1970.

[34] M. J. Fisher R. A. Wagner. The string to string correction problem. *Journal of the ACM*, 21:168–173, 1974.

[35] F.J. Ribadas Pena, M. Vilares Ferro, and Darriba Bilbao V.M. Optimización en el reconocimiento de patrones. *Procesamiento del Lenguaje Natural*, 26:239–1246, September 2000.

[36] K. Sagonas, T. Swift, and D. S. Warren. XSB as an efficient deductive database engine. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 23(2):442–453, June 1994.

[37] S. M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186, 1977.

[38] B.A. Shapiro and K.Zhang. Comparing multiple rna secondary structures using tree comparisons. *Computational Applications on Bioscience*, 6(4):309–318, 1990.

[39] Dennis Shasha, Jason Tsong-Li Wang, and Rosalba Giugno. Algorithmics and applications of tree and graph searching. In *Symposium on Principles of Database Systems*, pages 39–52, 2002.

[40] Smeaton A.F. Sheridan, Paraic. The application of morpho-syntactic language processing to effective phrase matching. *Information Processing and Management*, 28(3):349–369, 1992.

[41] A. F. Smeaton. Incorporating syntactic information into a document retrieval strategy: An investigation. In *Proc. the 9 th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 103–113, 1992.

[42] Alan F. Smeaton, Ruairi O'Donnell, and Fergus Kelledy. Indexing structures derived from syntax in TREC-3: System description. In *Text REtrieval Conference*, pages 0–, 1994.

[43] K.-C. Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26(3):422–433, 1979.

[44] Hisao Tamaki and Taisuke Sato. OLD resolution with tabulation. In Ehud Shapiro, editor, *Proceedings of the Third International Conference on Logic Programming*, Lecture Notes in Computer Science, pages 84–98, London, 1986. Springer-Verlag.

[45] Matsuru Tomita. *Efficient Parsing for Natural Language. A Fast Algorithm for Practical Systems*. Kluwer Academic Publishers, Norwell, Massachusetts, U.S.A., 1986.

[46] N. Varile. Charts: a data structure for parsing. In Margaret King, editor, *Parsing Natural Language*, pages 73–87. Academic Press, London, 1983.

[47] M. Vilares Ferro, D. Cabrero Souto, and F. J. Ribadas Pena. Pattern matching in shared forest. In *Proc. of Sixth International Workshop on Natural Language Understanding and Logic Programming (NLULP'99)*, pages 79–87, Las Cruces, New Mexico, USA, 1999.

[48] M. Vilares Ferro, Cabrero Souto D., and Ribadas Pena F.J. On integration of parsing and tree matching schemes. In Alexander Gelbukh, editor, *International Conference CICLing-2000, Conference on Intelligent text processing and Computational Linguistics*, pages 257–272, Centro de Investigación en Computación, Instituto Politécnico Nacional, Mexico D.F., February 2000.

[49] M. Vilares Ferro, Ribadas Pena F.J., and Darriba Bilbao V.M. Pattern matching as dynamic facility to get aboutness. In David S. Warren, Manuel Vilares, Leandro Rodriguez Liñares, and Miguel A. Alonso, editors, *Proc. of Tabulation in Parsing and Deduction (TAPD 2000)*, pages 175–187, Vigo, Spain, September 2000.

[50] M. Vilares Ferro, Ribadas Pena F.J., and Darriba Bilbao V.M. Approximate vldc pattern matching in shared forest. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, volume 2004 of *Lecture Notes in Computer Science*, pages 483–494. Springer-Verlag, Berlin-Heidelberg-New York, 2001.

[51] M. Vilares Ferro, Ribadas Pena F.J., and Darriba Bilbao V.M. Approximate pattern matching in shared forest. In J. Küng M. Ibrahim and N. Revell, editors, *Database and Expert Systems Applications*, volume 1873 of *Lecture Notes in Computer Science*, pages 322–333. Springer-Verlag, Berlin-Heidelberg-New York, 2002.

[52] M. Vilares Ferro, F. J. Ribadas Pena, and Graña Gil J. On pattern-matching as query facility. In *Proc. of APPIA-GULP-PRODE 2002, 2002 Joint Conference on Declarative Programming*, pages 279–268, Madrid, Spain, September 2002.

[53] M. Vilares Ferro, F.J. Ribadas Pena, and Graña Gil J. Approximately common patterns in shared-forests. In Ling Liu Henrique Paques and David Grossman, editors, *Proc. of the 2001 ACM CIKM, Tenth International Conference on Information and Knowledge Management*, pages 73–80, Atlanta, Georgia, USA, February 2001. ACM Press.

[54] M. Vilares Ferro, Darriba Bilbao V.M., and Ribadas Pena F.J. A proposal on error repair. In David S. Warren, Manuel Vilares, Leandro Rodriguez Liñares, and Miguel A. Alonso, editors, *Proc. of Tabulation in Parsing and Deduction (TAPD 2000)*, pages 127–138, Vigo, Spain, September 2000.

[55] M. Vilares Ferro, Darriba Bilbao V.M., and Ribadas Pena F.J. Regional least-cost error repair. In Sheng Yu and Andrei Paun, editors, *Implementation and Application of Automata*, volume 2088 of *Lecture Notes in Computer Science*, pages 293–301. Springer-Verlag, Berlin-Heidelberg-New York, 2001.

[56] Manuel Vilares Ferro. *Efficient Incremental Parsing for Context-Free Languages*. PhD thesis, University of Nice, France, 1992.

[57] Manuel Vilares Ferro, Víctor Manuel Darriba Bilbao, and Miguel A. Alonso Pardo. Searching for asymptotic error repair. In Jean-Marc Champarnaud and Denis Maurel, editors, *Seventh International Conference on Implementation and Application of Automata (CIAA 2002)*, pages 275–280, Tours, France, July 2002.

[58] Eric Villemonte de la Clergerie. DyALog : une implantation des clauses de Horn en programmation dynamique. In *Proc. of the 9th Seminaire de Programmation en Logique*, pages 207–228. CNET, May 1990.

[59] Eric Villemonte de la Clergerie. *Automates à Piles et Programmation Dynamique. DyALog : Une Application à la Programmation en Logique*. PhD thesis, Université Paris 7, Paris, France, 1993.

[60] Eric Villemonte de la Clergerie and Bernard Lang. LPDA: Another look at tabulation in logic programming. In Pascal Van Hentenryck, editor, *Proc. of the 11th International Conference on Logic Programming (ICLP'94)*, pages 470–486, Massachusetts Institute of Technology, June 1994. The MIT Press.

[61] Jason Tsong-Li Wang, Bruce A. Shapiro, Dennis Shasha, Kaizhong Zhang, and Kathleen M. Currey. An algorithm for finding the largest approximately common substructures of two trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):889–895, 1998.

[62] Jason Tsong-Li Wang, Dennis Shasha, George J. S. Chang, Liam Relihan, Kaizhong Zhang, and Girish Patel. Structural matching and discovery in document databases. In *Proc. of ACM SIGMOD Conf. on Management of Data,*, pages 560–563, 1997.

[63] Jason Tsong-Li Wang, Kaizhong Zhang, Karpjoo Jeong, and Dennis Shasha. A system for approximate tree matching. *Knowledge and Data Engineering*, 6(4):559–571, 1994.

[64] D. H. Younger. Recognition and parsing of context-free languages in time $O(n^3)$. *Information and Control*, 10(2):189–208, 1967.

[65] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989.

[66] Kaizhong Zhang, Dennis Shasha, and Jason Tsong-Li Wang. Approximate tree matching in the presence of variable length don't cares. *J. Algorithms*, 16(1):33–66, 1994.

Índice alfabético