ECGI

Un algoritmo de Inferencia Gramatical mediante Corrección de Errores

Héctor Miguel Rulot Segovia

Trabajo presentado en la Universitat de València para optar al grado de Doctor en Ciencias Físicas. Realizado bajo la dirección de

D. Enrique Vidal Ruiz y D. Francisco Casacuberta Nolla.

Valencia, 1992.

**Este trabajo ha sido soportado en parte por el proyecto TIC-0448/89 de la CICYT.*

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

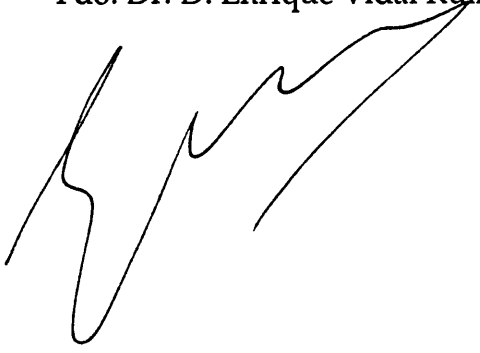
D. ENRIQUE VIDAL RUIZ, Catedrático Numerario del Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia.

D. FRANCISCO CASACUBERTA NOLLA, Catedrático Numerario del Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia.

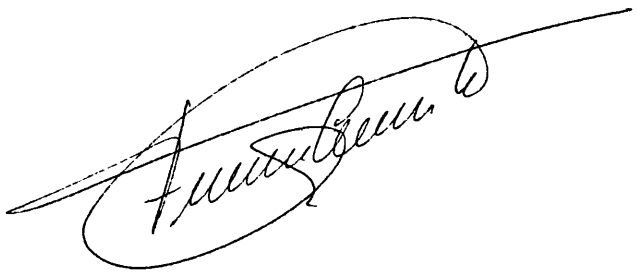
CERTIFICAMOS: Que bajo nuestra dirección, D. HECTOR MIGUEL RULOT SEGOVIA ha realizado el trabajo "ECGI: Un Algoritmo de Inferencia Gramatical Mediante Corrección de Errores", para optar al grado de Doctor en Ciencias Físicas.

Y, para que conste a los efectos legales, firmamos la presente en Valencia, abril de mil novecientos noventa y dos.

Fdo: Dr. D. Enrique Vidal Ruiz



Fdo: Dr. D. Francisco Casacuberta Nolla



-Magnífico –le dije–. No imagino nada más repulsivo que escribir una tesis, pero tu eres capaz de disfrutar con ello.

-Gracias. Representará un año de trabajo de todos modos.

William Sloane, en *"El tiempo de la noche"*.

Al Dr. Héctor Rulot†,
y también, al Dr. Héctor Rulot†.
Sin ellos, este trabajo nunca
habría sido posible.

Agradecimientos

Tengo que agradecer aquí la inestimable ayuda, consejo y aliento que me ha prestado a lo largo de todo este trabajo Enrique Vidal Ruiz, que ha excedido con creces sus obligaciones como codirector de esta tesis, demostrándome con ello una gran e inmerecida amistad.

Igualmente y cómo no, agradezco a Fracisco Casacuberta Nolla, también codirector de la misma tesis, su paciencia incansable y su colaboración siempre presente y nunca denegada.

Tengo aquí un extraordinario recuerdo para todos los componentes del Grupo de Investigación en Reconocimiento del Habla de Valencia, que primero en el Centro de Informática de la Universitat Literaria, y luego en el Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia, me han animado y ayudado a lo largo de muchos años para que diera fin a este trabajo.

Asimismo, debo mencionar a mis amigos y compañeros de trabajo del Centro de Informática por su paciencia conmigo durante todo este tiempo.

Lamento no disponer aquí de espacio suficente para mencionar la aportación de todos y cada uno de aquellos que han dedicados horas y más horas al desarrollo y mejora del ECGI (Natividad Prieto, Emilio Sanchis, Asunción Castaño, Francesc Torró, Isabel Alfonso) así como aquellos que me han permitido compararlo con sus propios métodos (Pedro García, Isabel Galiano, Encarna Segarra), todos los cuales han significado un continuo incentivo para que prosiguiera mi trabajo. Todos ellos y muchos más han colaborado, incluso poniendo su voz, en el duro trabajo de recolección y parametrización de las muestras (especialmente José Miguel Benedí).

Mención aparte merecen José Miguel Valiente y Gabriela Andreu del Departamento de Ingeniería de Sistemas, Computadores y Automática, pues sin ellos hubiera sido imposible la realización de los experimentos de reconocimiento de imágenes.

Universitat de València, 1992.

Héctor Rulot

INDICE

Prólogo I
En pocas palabras I
¿De qué se trata? I
¿Cómo se hace? II
Los Capítulos II
Capítulo 1: El Reconocimiento de Formas 1
1.1 Fundamentos 1
1.1.1 Las etapas de un reconocedor 2
1.1.2 Representación de objetos y formas 4
1.1.3 Composición de reconocedores 5
1.1.4 Reconocimiento Geométrico y Reconocimiento Estructural 5
1.1.4.1 Los modelos geométricos 6
1.1.4.2 Los modelos estructurales 7
1.1.4.3 Métodos geométricos versus métodos estructurales 7
1.1.5 Segmentación 8
1.1.6 El tamaño de las subestructuras 11
1.1.7 El Aprendizaje Automático 12
1.1.7.1 Taxonomía del aprendizaje 12
1.1.7.2 Factibilidad del aprendizaje Inductivo 12
1.2 El Reconocimiento Automático del Habla 13
1.2.1 Adquisición, parametrización y etiquetado en RAH 15
1.2.2 Estado del Arte 17
1.3 El Reconocimiento Automático de Caracteres 18
1.3.1 Adquisición, reducción de ruido y parametrización en RAC 19
1.3.2 Estado del Arte 20
Capítulo 2: Los Métodos Sintácticos 23
2.1 Introducción 23
2.2 Los lenguajes y sus gramáticas 23
2.3 Reconocedores de lenguajes 25
2.4 El problema de la imprecisión 27
2.4.1 Análisis sintáctico corrector de errores 28
2.4.2 Gramáticas estocásticas 31
2.4.2.1 Definiciones 31
2.4.2.2 Autómatas estocásticos 33
2.4.2.3 Aprendizaje de gramáticas estocásticas 34
2.4.3 Análisis sintáctico corrector de errores estocástico 35
2.5 Gramáticas SANSAT 37
2.5.1 Gramática SANSAT equivalente a una gramática. 37
2.5.2 Autómata de estados etiquetados (LAS) 39
2.5.3 LAS y Modelos Ocultos de Markov (HMM) 40
Capítulo 3: Inferencia Gramatical 43
3.1 Introducción 43
3.2 Especificación del problema de IG 44
3.2.1 Dominio de formas a inferir 44
3.2.2 Espacio de hipótesis o representaciones 44
3.2.3 Método de presentación de ejemplos 44
3.2.4 Método de inferencia 45
3.2.5 Criterio de éxito 46
3.3 Métodos enumerativos 46
3.4 Presentación positiva 47
3.5 Métodos Heurísticos y Métodos Caracterizables 48
3.6 Métodos constructivos de IG 49
Capítulo 4: Programación Dinámica 51
4.1 Introducción 51

4.2 Principio de optimalidad 52
4.3 Relación de recurencia, versión iterativa 53
4.4 Programación Dinámica y autómatas 56
4.4.1 Grafos multi-etapa 56
4.4.2 Camino mínimo en un grafo multietapa 57
4.4.3 Trellis 57
4.5 El algoritmo de Viterbi 59
4.6 Versión iterativa del algoritmo de Viterbi 60
Capítulo 5: Corrección de errores 63
5.1 Reglas de error y trellis 63
5.2 Autómatas libres de circuitos 65
5.2.1 El algoritmo ViterbiCorrector 67
5.3 Autómatas con circuitos 68
5.3.1 El trellis tiene circuitos 68
5.3.2 Camino óptimo en un grafo con circuitos 69
5.3.3 El algoritmo CICLICO 72
5.4 Camino y traza de edición 74
Capítulo 6: El método ECGI 77
6.1 Motivación 77
6.2 Método 79
6.2.1 Un algoritmo trivial 79
6.2.3 ECGI 80
6.3 Algoritmo 83
6.4 Propiedades de la gramática inferida 85
6.5 Heurísticos 87
6.5.1 Las asunciones de ECGI 87
6.5.2 No se añaden las reglas de error 90
6.5.3 Heurísticos no esenciales 91
6.5.3.1 Prioridad a la substitución 92
6.5.3.2 Inserción en paralelo 94
6.5.3.3 Otras consideraciones 95
6.6 (Di)similitudes 96
6.6.1 Menos errores. 97
6.6.2 . Relativamente menos errores 97
6.6.3 Más aciertos 98
6.6.4 Relación entre criterios 98
6.6.5 Nadie es perfecto 100
6.7 Reconocimiento 101
6.8 Aprendizaje continuo 102
6.9 Sólo substituciones 104
6.9.1 Consideraciones prácticas 105
6.10 Implementación 106
6.10.2 Representación Gráfica de un LAS de ECGI 107
6.10.3 Version LAS de la construcción 108
Capítulo 7: ECGI Estocástico 113
7.1 Ambiguedad y probabilidad 114
7.2 Construcción estocástica 114
7.3 Reconocimiento estocástico 115
7.4 La estocasticidad de la gramática expandida 118
7.4.1 Las probabilidades de deformación 118
7.4.2 Estocasticidad según ECGI 120
7.4.3 En la práctica 121
7.5 Sólo sustitución 124
Capítulo 8: Resultados experimentales 125
8.1 Reconocimiento del Habla 126
8.1.1 Representación simbólica de la señal vocal 126
8.1.1.1 Corpus piloto: Dígitos monolocutor 127
8.1.1.2 Corpus principal: Dígitos 129
8.1.1.3 Corpus difícil: letras 131
8.1.2 Experimentos de reconocimiento 131
8.1.2.1 Corpus piloto 131
8.1.2.2 Corpus principal 135
8.1.2.2.1 Experimentos sencillos 135

8.1.2.2.2 Leaving-k-out 136
8.1.2.3 Corpus de Letras 139
8.1.2.3.1 Experimentos sencillos 139
8.1.2.3.1 Leaving-k-out 140
8.2 Reconocimiento de imágenes planas 142
8.2.1 Representación simbólica de imágenes planas 143
8.2.3 Dígitos Manuscritos 144
8.2.3.1 Los experimentos 145
8.2.3.2 Los resultados 146
8.2.4 Dígitos Impresos 148
8.2.4.1 Los experimentos 149
8.2.4.2 Los resultados 150
8.2.4.3 El uno sin base 152
8.2.6 Invarianza a la rotación y otras posibles extensiones 154
8.3 Experimentos de síntesis 155
8.4 Resumen de resultados 157
Capítulo 9: ECGI: Estudios Empíricos 159
9.1 Introducción 159
9.2 ¿Minimizar errores o maximizar aciertos? 159
9.3 Convergencia del aprendizaje 162
9.4 Ambigüedad de las gramáticas 165
9.5 Sólo Substituciones 167
9.6 ECGI estocástico y no estocástico 169
9.6.1 Ignorar la frecuencia de las reglas 170
9.6.2 Ignorar las frecuencias de los errores 171
Capítulo 10: Caminos y Simplificación de Autómatas 173
10.1 Introducción 173
10.2 Caminos en un grafo sin circuitos 173
10.2.1 Numero de caminos por vértice 174
10.2.2 Número de caminos por arista 175
10.2.3 Camino más largo y más corto 176
10.2.4 Algoritmos no recursivos 176
10.2.5 Relación Tráfico/Probabilidad 177
10.3 Simplificación de autómatas 179
10.3.1 Método 180
10.3.1.1 Estado menos significativo 180
10.3.1.2 Criterios de detención 181
10.3.1.3 El algoritmo 181
10.3.2 Aplicación práctica y resultados 183
Capítulo 11: ECGI Determinista y Cadena Mediana 187
11.1 Introducción 187
11.2 Gramáticas Deterministas 187
11.2.1 Método 188
11.2.2 Implementación 190
11.2.3 Resultados Experimentales 192
11.3 Cadena mediana 194
11.3.1 Mediana generalizada y mediana de un conjunto 194
11.3.2 La cadena más probable 195
11.3.3 Consideraciones prácticas 195
11.3.4 La cadena más frecuente 196
11.3.5 Comprobación empírica 197
Capítulo 12: Comparaciones 201
12.1 Métodos similares de IG 201
12.1.1 Método de Thomason 201
12.1.2 Método de Falaschi 203
12.1.3 Método de Chirathamjaree 206
12.1.4 Método de Marco 206
12.2 Métodos en Reconocimiento del Habla 207
12.2.1 Modelos Ocultos de Markov 207
12.2.2 Alineamiento Temporal No lineal, K-Vecinos 209
12.3 Métodos en Reconocimiento de Imágenes 210
Epílogo 213
Conclusiones 213

Perspectivas 216
Apéndice A: Otros trabajos sobre ECGI ..... A-I
Apéndice B: Tablas complementarias ..... B-I
B.1 Número de estados de los autómatas inferidos ..... B-I
B.1.1 Dígitos Hablados: ..... B-I
B 1.2 Letras Habladas: ..... B-II
B.1.3 Dígitos Manuscritos: ..... B-III
B.1.4 Dígitos Impresos: ..... B-IV
B.2 Matrices de confusión ..... B-V
B.2.1 Dígitos Manuscritos (sólo substitución) ..... B-V
B.2.2 Dígitos impresos (sólo substitución) ..... B-VI

Prólogo

En pocas palabras

¿De qué se trata?

En el presente trabajo se introduce un nuevo método de inferencia gramatical, el algoritmo INGRACE (INferencia GRAmatical mediante Corrección de Errores).

Desafortunadamente para los hispano-parlantes, al INGRACE se le ha dado a conocer sobre todo según la versión inglesa de su nombre: «**Error-Correcting Grammatical Inference algorithm**», con lo cual a partir de ahora utilizaremos para referirnos a él el impronunciable acrónimo «**ECGI**».

Como todos los algoritmos de inferencia gramatical, la aplicación principal de ECGI se halla en inferir la estructura (gramática) que mejor represente una determinada forma (lenguaje) a partir de muestras de objetos pertenecientes a dicha forma (cadenas). Es pues un algoritmo cuya finalidad original se halla en el reconocimiento de formas.

En concreto, el (buen) funcionamiento de ECGI se ha comprobado en dos casos muy diferentes del reconocimiento de formas: el reconocimiento de la **palabra hablada**, campo al que se dedica nuestro grupo investigador, y el reconocimiento de **formas planas** (en imágenes bidimensionales), campo que nos interesó, pues es también poco usual en él el empleo de métodos de inferencia gramatical.

Ello no es limitativo, y en principio ECGI es aplicable a cualquier problema de reconocimiento de formas en el que se puedan representar las mismas mediante cadenas de símbolos.

Cabe notar sin embargo, el que la naturaleza inherentemente heurística del método restringe el campo de aplicación a aquellos casos en los que las características diferenciadoras de las formas se centran en la concatenación unidimensional de unos u otros (grupos de) símbolos y en la distinta longitud (duración) de los grupos de símbolos (las subformas o subcadenas).

¿Cómo se hace?

ECGI construye una gramática regular (un autómata de estados finitos) mediante un procedimiento incremental, que en principio permite efectuar simultáneamente inferencia y reconocimiento.

El procedimiento de construcción se apoya en un método de Programación Dinámica, el cual determina, para cada nueva muestra, la secuencia de reglas de la gramática que la generan con un mínimo número de reglas de error (de reglas **no** pertenecientes a la gramática). A partir de estas reglas de error, se determinan las reglas a añadir a la gramática actual, de forma que en lo sucesivo la muestra forma parte del lenguaje de la misma.

La minimización explícita permite controlar el crecimiento de la gramática, a la vez que induce una generalización que ha demostrado ser especialmente idónea para capturar la variabilidad presente en las (sub)estructuras de las cadenas consideradas, así como en la concatenación y la longitud de estas subestructuras.

Los experimentos realizados en reconocimiento del habla, destino inicial para el que se ideó ECGI, han demostrado una elevada capacidad de aprendizaje, que conduce a prestaciones en reconocimiento superiores a la conseguidas cuando los autómatas se construyen a mano, y muy similares (o superiores) a las obtenidas mediante otros métodos de aprendizaje, ya sean o no de inferencia gramatical.

En cuanto al reconocimiento de formas planas, los experimentos se centraron en formas muy típicas y sobre las cuales existe una abundante literatura: los dígitos impresos y/o manuscritos. Aquí también ECGI demostró ser comparable y en muchos casos superior a métodos clásicos en ese campo (momentos centrales normalizados con distancia de Mahalanobis).

Los Capítulos

El texto de la presente exposición se distribuye en 12 capítulos:

- El primer capítulo resume en unas páginas qué es y de qué trata la disciplina del "reconocimiento de formas", desarollando brevemente algunos conceptos de interés para situar este trabajo, tales como el reconocimiento geométrico (estadístico) y sintáctico de formas y el problema de la longitud de las subestructuras

  A continuación el capítulo se centra en el "reconocimiento automático del habla", del que se explican muy por encima los métodos básicos y el estado del arte, ambos necesarios para poder

entender y comparar los resultados experimentales que se presentarán al final de este volumen.

De la misma manera y por la misma razón, al final de capítulo se ofrece una introducción al reconocimiento de formas (imágenes) planas y más concretamente al reconocimiento de dígitos impresos/ manuscritos.

- El segundo capítulo se presenta como una introducción a la teoría de lenguajes formales, pero se centra casi de inmediato en los lenguajes y gramáticas regulares, así como en sus respectivas versiones estocásticas, todos ellos imprescindibles para explicar mínimamente el funcionamiento y aplicación de ECGI.

- El tercer capítulo intenta plantear formalmente el problema de la inferencia gramatical. En él, tras introducir los conceptos básicos sobre inferencia y gramáticas, se exponen toda una serie de resultados teóricos que van constriñendo el campo de lo que es posible inferir y con qué procedimientos puede hacerse. Ello permite delimitar claramente el dominio sobre el que trabajará nuestro algoritmo, así como tipificarlo y situarlo con relación a los otros métodos existentes en inferencia gramatical.

- El cuarto capítulo procede a una exposición rápida de los métodos y algoritmos de la Programación Dinámica, técnica de optimización en la que se basa ECGI a la hora de minimizar el crecimiento de la gramática que infiere. El capítulo se fija principalmente en el algoritmo de Viterbi, imprescindible cuando se manejan gramáticas estocásticas y correctoras de errores.

- En el quinto capítulo aparecen las primeras aportaciones originales de este trabajo. En este capítulo se lleva a cabo un análisis detallado de los algoritmos correctores de errores para el análisis sintáctico regular (estocástico o no), prestando particular atención a los problemas que se presentan con las gramáticas con circuitos. Se proponen, comprobándolos en la práctica, varios algoritmos para solventar estos problemas. Finalmente se introduce un concepto básico para comprender ECGI: el camino de derivación.

- El sexto capítulo presenta el método ECGI, el algoritmo y sus propiedades, así como la propiedades de las gramáticas inferidas. Se estudian detalladamente los heurísticos en que se basa el algoritmo, su utilización en reconocimiento de formas y su implementación real.

- En el séptimo capítulo se introduce la extensión estocástica de ECGI y se describen los detalles del procedimiento seguido para aprovechar la información estadística contenida en la frecuencia de utilización de las reglas. Se propone y estudia una nueva manera de definir la estocasticidad de la gramática expandida cuando se efectúa un análisis sintáctico corrector de errores.

- El octavo capítulo expone extensivamente los múltiples experimentos de reconocimiento de formas que han sido necesarios tanto para demostrar la viabilidad de ECGI y su eficacia como reconocedor, como para estudiar su comportamiento y propiedades en distintos casos. Se describen los corpus de datos utilizados (dígitos hablados e impresos), los métodos de parametrización aplicados y los resultados obtenidos.

- El noveno capítulo se dedica a profundizar en ECGI mediante una serie de estudios empíricos. Se muestran los resultados de distintos experimentos destinados a analizar la convergencia de ECGI, la ambigüedad de las gramáticas que genera y la cantidad de información que realmente aportan los distintos tipos de frecuencias que se utilizan en la extensión estocástica.

- En el décimo capítulo se introduce una serie de cantidades, todas ellas relacionadas con los caminos entre dos vértices de un grafo sin circuitos. Estas cantidades se emplean luego para reducir la complejidad de los modelos generados por ECGI (simplificación de autómatas).

- En el capítulo once se expone un heurístico (subóptimo) con el que se puede obligar a ECGI a generar gramáticas deterministas, lo que permite eliminar la ambigüedad de las mismas. Asimismo se presenta un método para obtener una aproximación a la cadena mediana de un conjunto de cadenas a partir del correspondiente modelo inferido por ECGI.

- El duodécimo capítulo y último compara ECGI con otros métodos similares de inferencia gramatical (basados también en la corrección de errores) y con diversas otras metodologías utilizadas generalmente en reconocimiento del habla y de formas planas.

Finalmente, unas páginas de epílogo, seguidas de algunos apéndices y de la bibliografía concluyen el presente trabajo.

1

El Reconocimiento de Formas

1.1 Fundamentos

Si bien es relativamente sencillo el conseguir que una máquina (p.e.: un ordenador) capte lo que le rodea (basta dotarle de una adecuada serie de sensores y/o transductores: células fotoeléctricas, micrófonos,...), no lo es tanto el conseguir que sea capaz de interpretarlo y/o reconocerlo.

Ciertamente, es casi trivial conseguir que la máquina "clasifique" de manera elemental una serie de medidas recogidas por un sensor (basta una tabla) e identifique de esta manera una situación, un objeto o una variación de su entorno (p.e.: bajada de presión) (figura 1.1). Y efectivamente, ello equivale a una tarea perceptiva simple.

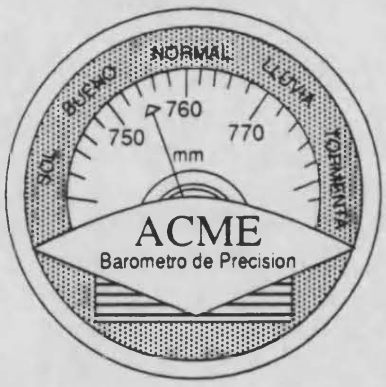


Figura 1.1 Una Máquina capaz de percibir (el estado del tiempo).

Pero el problema se presenta cuando la interpretación de lo que sucede en el entorno depende, no solo de un sensor, sino de varios; no solo de una medida de ese sensor, sino de muchas medidas recogidas a lo largo del tiempo. Y se complica mucho más cuando para poder comprender el entorno es necesario recurrir a una gran base de conocimientos

(experiencias) previamente almacenadas, y que debe ser consultada y comparada rápidamente con lo captado a través de los sensores. Con todo, la dificultad más grande aparece cuando se considera que las medidas de los sensores no son absolutamente fiables, y peor aún, que la descripción que se tiene del entorno tampoco lo es.

La complejidad involucrada es tal, que hoy en día, a pesar de largos estudios sobre la función perceptiva realizados por los psicólogos sobre personas y animales, y del ininterrumpido intento de duplicarla por parte de los técnicos (generalmente informáticos), la percepción, en su acepción más general, sigue siendo una tarea que, si bien todos los seres humanos llevamos a cabo rutinariamente, nadie sabe realmente cómo.

A pesar de todo ello, es enorme el interés que tiene el poder construir máquinas capaces de reconocer caracteres (leer libros), imágenes (p.e. identificar aviones), sonidos (p.e. entender cuando se les habla), etc..., por lo que nunca se ha cejado en el empeño. En la actualidad, principalmente gracias a la evolución de la informática, se han conseguido interesantes resultados prácticos, y lo que es más importante, grandes avances en la comprehensión de las dificultades y en la definición de los medios que deben ponerse en acción para superarlas. En el resto de este capítulo describimos (muy por encima) algunos de estos resultados, así como los medios que se utilizan para obtenerlos.

1.1.1 Las etapas de un reconocedor

La percepción de objetos por un sistema es el objetivo declarado de la disciplina del *Reconocimiento de Formas*, que se engloba en un conjunto de técnicas mucho más ambiciosas: la *Inteligencia Artificial* [Cohen,82]. El reconocimiento de formas, sin embargo, ha ido adquiriendo entidad de por sí mismo, hasta el punto de que en la actualidad constituye un campo de investigación que evoluciona con dinámica propia, a menudo independiente de la inteligencia artificial.

Reconocer un objeto consiste en asociarle (identificarlo con) un *mensaje semántico*, es decir un significado. En el caso más general, este mensaje semántico es simplemente un punto (también llamado *forma*) de un *universo semántico* (a veces llamado universo interno, en oposición al universo externo captado por los sensores). Normalmente, a una forma (p.e.: silla) le corresponden muchos posibles objetos (p.e.: todas las posibles sillas). Dicho de otro modo, una forma es un *conjunto de objetos* que se caracterizan por estar etiquetados por un mismo mensaje semántico.

Un sistema de reconocimiento de formas, en su versión más simple, estará constituído de la manera esquematizada en la figura 1.2, es decir, por dos módulos:

- Un módulo de *representación*, que obtiene, a partir de cada objeto, (usualmente) captado del mundo real mediante una serie de sensores, una representación conveniente para su utilización por el módulo de interpretación.

- Un módulo de *interpretación*, que proporciona, a partir del conjunto de formas, también representado convenientemente, y de la representación del objeto proporcionada por el módulo de representación, la forma a la que pertenece el objeto. Lleva a cabo pues un proceso de *comparación* ("pattern matching") entre el objeto y el conjunto de formas.

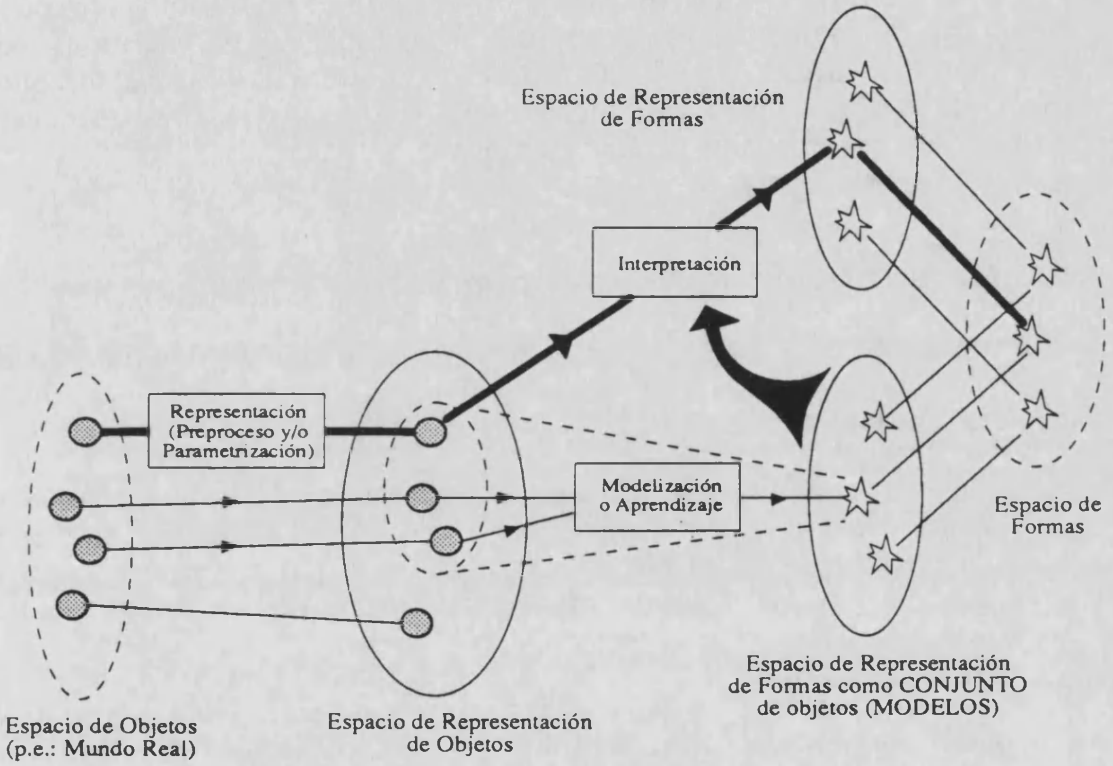


Figura 1.2 Composición de un reconocedor de formas. El trazo grueso muestra las sucesivas etapas de un proceso de reconocimiento.

El módulo de representación puede no ser necesario, si se da la circunstancia de que los objetos ya vienen dados de manera conveniente para el módulo de interpretación. En muchos otros casos (como los presentados en la parte experimental de este trabajo) el módulo de representación está compuesto por varias etapas, algunas de las cuales efectúan un *preproceso* y otras una *parametrización*:

- Las etapas de *preproceso* efectuan en general transformaciones que, o bien no cambian el dominio de representación, o bien llevan a subconjuntos de éste (p.e.: filtrado de la señal, supresión de grises por umbral...).

- Las etapas de *parametrización* cambian el dominio de representación y a menudo reducen drásticamente la cantidad de información, suprimiendo aquella que resulta redundante y/o inútil. En este último caso, se puede considerar la operación como equivalente a extraer las características más significativas del objeto, para representarlo por un conjunto de *parámetros* o *descriptores* (p.e.: transformada discreta de Fourier, cadena de símbolos,...).

El conjunto de formas utilizado por el módulo de interpretación representa el conocimiento que tiene el sistema de su entorno. Muy a menudo la representación del mismo es simplemente un conjunto de representaciones de formas (p.e.: un conjunto de gramáticas), pero puede estar representado por una estructura única y compleja (p.e.: una única red o gramática). El proceso de obtención de este conjunto de formas se conoce como *modelización* o *aprendizaje* del entorno (de sus objetos) a reconocer y es efectuado por el *módulo de aprendizaje.*

1.1.2 Representación de objetos y formas

La representación elegida para los objetos se deriva usualmente del método de interpretación escogido. Estas representaciones pueden clasificarse en dos grandes tipos:

- *No estructuradas*: El objeto se representa por un conjunto de (sub)objetos sin relación ellos (o con una relación arbitraria). Ejemplo típico es un vector o matriz en los que no importa cómo se distribuyen los componentes (p.e.: un punto en un espacio n-dimensional).

- *Estructuradas*: El objeto se representa mediante un conjunto de (sub)objetos más un conjunto de relaciones entre ellos (una *estructura*). Normalmente las relaciones son de contigüedad y/o sucesión: cadenas, árboles, grafos ...

A su vez, los subobjetos de que están formados los objetos pueden estar representados de la misma manera que los objetos (es decir, estar formados de otros subobjetos de nivel inferiror), o ser simplemente un conjunto de parámetros continuos (números reales) o cuantizados (numeros enteros). Cuando son parámetros cuantizados con pocos niveles de cuantización (del orden de 100) se puede asignar a cada valor un nombre o *símbolo.*

Las posibles representaciones de las formas son mucho más complejas: no deben representar un único objeto sino un *conjunto de ellos.* Las representaciones de un conjunto de objetos se conocen también como *modelos.* Dos de los casos más simples (y más corrientes) se engloban en lo que se conoce como el reconocimiento *geométrico* o *estadístico* de formas y el reconocimiento *sintáctico* de formas (ver apartado 1.1.4).

Por otra parte, con el fin de comunicar el resultado del reconocimiento al experimentador o a otro posible módulo, es necesaria una representación alternativa de la forma, pues la representación como conjunto de objetos es sólo útil para el módulo de interpretación. Esta segunda representación de cada forma será similar a las utilizadas para los objetos, puesto que la forma puede considerarse un objeto (punto) del universo semántico. Si el número de formas (la talla del universo semántico) es pequeño esta representación puede consistir simplemente en un único valor cuantizado o *etiqueta*. El reconocedor de formas se podrá entonces considerar como un *clasificador*. En casos más complejos, podrá recurrirse a una representación estructurada de las formas (p.e.: cadena). Se llamará *traductor* a un reconocedor en los que tanto los objetos como las formas en salida estén representados de manera estructurada.

1.1.3 Composición de reconocedores

La posibilidad que tiene un reconocedor de comunicar la forma reconocida a otro módulo lleva inmediatamente a pensar en la *composición de reconocedores*, es decir, a utilizar otro reconocedor para analizar (identificar o reconocer) con más detalle lo que han reconocido otros reconocedores. Este procedimiento es de hecho muy utilizado siempre que los subobjetos de un objeto son estructurados: uno o varios reconocedores de nivel inferior se dedican a extraer los subobjetos. Otro caso más complejo se da en los *reconocedores multinivel*, formados por varios reconocedores idénticos, cuyos objetos están representados en el mismo dominio que sus formas en salida, y que están interconectados de manera más o menos regular, a menudo organizados en capas (etapas o niveles). Típico ejemplo de ello son los perceptrones multicapa [Lippmann,87].

En un reconocedor compuesto, las formas intermedias, obtenidas por los reconocedores de las capas inferiores, constituyen los objetos de las capas superiores. Estos objetos son *objetos abstractos*, puesto que no tienen relación directa con lo captado por los sensores, siendo su definición arbitraria y particular a un reconocedor determinado. En determinadas situaciones, estos objetos abstractos corresponderán a abstracciones más o menos intuitivas (fonemas, sílabas, polígonos, etc...), pero éste no es el caso más usual, principalmente debido a lo difícil que resulta definir estas abstracciones.

1.1.4 Reconocimiento Geométrico y Reconocimiento Estructural

Dos son actualmente las maneras más usuales de representar las formas en su aspecto de conjunto de objetos: aquellas que recurren a una visión *geométrica* de lo que es un conjunto, considerándolo como una (sub)región

de un determinado espacio y aquellas que prefieren verlo como formado por elementos que cumplen ciertas reglas *estructurales*.

1.1.4.1 Los modelos geométricos

Estos modelos asumen que los objetos son "de una pieza" (están representados de forma NO estructurada). Cada objeto se representa por una matriz (generalmente de una o dos dimensiones) de números o símbolos (los *parámetros*) y la forma se representa mediante uno o varios *objetos patrón* y una *medida de disimilitud* (que puede ser una distancia, métrica o no) entre ellos y los objetos a reconocer [Duda,73]. Un ejemplo en reconocimiento del habla se puede encontrar en [Rulot,85].

Cuando el espacio n-dimensional en el que se hallan representados los objetos es continuo, y se escogen adecuadamente los parámetros, a menudo se puede conseguir que los puntos (objetos) pertenecientes a la misma forma se agrupen en regiones. Estas regiones definen entonces las formas, y sus fronteras con otras (*fronteras de decisión*) pueden entonces describirse mediante *funciones discriminantes* [Duda,73] (figura 1.3).

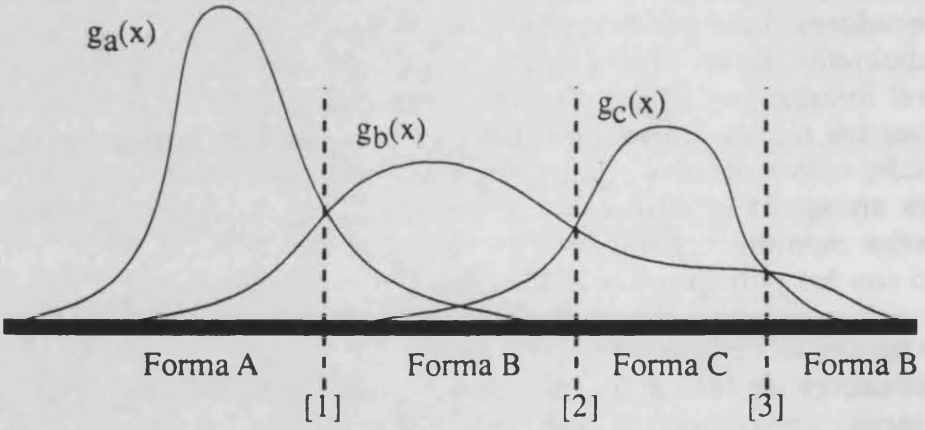


Figura 1.3 Formas (unidimensionales), sus funciones discriminantes (g(X)) y fronteras de decisión ([1][2][3])en un espacio unidimensional.

Ejemplo típico de función discriminante sería la inversa de la distancia a un centro (el patrón o *prototipo*). También lo sería la probabilida asociada a una distribución normal alrededor del mismo centro. En este último caso y en otros similares, se intenta aproximar una función discriminante mediante una función bien conocida, de la cual sólo resta determinar sus parámetros definitorios (aqui serían la media y desviación típica). Se habla entonces de reconocedor *paramétrico*. Muy a menudo estos modelos paramétricos son *estadísticos*, lo que justifica el nombre de *reconocimiento estadístico* de formas que también se le da al reconocimiento geométrico.

Otro ejemplo típico, en el que las funciones discriminantes son planos (o combinaciones de ellos), se encuentra en los clasificadores basados en redes neuronales tipo perceptrón [Duda,73] [Lippmann,87].

1.1.4.2 Los modelos estructurales

En estos modelos, se asume una representación estructurada de los objetos, estando usualmente los subobjetos representados de manera similar a la utilizada en los métodos globales. Las formas se representan mediante una serie de "reglas de composición" que deben cumplir los subobjetos para pertenecer al conjunto, existiendo un amplio abanico de posibilidades (árboles de decisión , expresiones lógicas, redes, modelos de Markov, reglas, gramáticas... [Hunt,75] [Gonzalez,78] [Fu,82] [Rabiner,83] [Cohen,82] [Valiant,84] [Quinlan,86] [Miclet,86]).

El reconocimiento debe basarse entonces, no sólo en identificar los subobjetos del objeto examinado aplicando para ello los métodos de reconocimiento geométrico de formas, sino también en analizar de alguna manera si la relación entre ellos es la buscada. Todo reconocedor estructural es por lo tanto, y por definición, un reconocedor compuesto, de por lo menos dos etapas: interpretación de subobjetos e interpretación de estructura.

Aquí el ejemplo serían los métodos gramaticales o sintácticos, en los que cada objeto es representado por un conjunto de símbolos cuya concatenación proporciona la estructura. Si esta estructura cumple una serie de reglas, especificadas mediante una gramática, es que pertenence a la forma representada por el lenguaje de esa gramática [Gonzalez,78] [Fu,82].

1.1.4.3 Métodos geométricos versus métodos estructurales

Los métodos geométricos de reconocimiento de formas son muy útiles para el reconocimiento de formas "simples", es decir, cuando se tiene un número pequeño de formas separables mediante funciones discriminantes no demasiado complejas. Estos métodos se utilizan por lo tanto muy frecuentemente para realizar clasificadores, han sido profusamente estudiados, y existe una gran variedad de algoritmos bien documentados y analizados que los utilizan y que funcionan perfectamente si los objetos tienen una distribución adecuada en el espacio de objetos [Duda,73]; lo cual desgraciadamente no es siempre cierto.

Por su parte, la aproximación estructural permite elaborar modelos más complejos, pues se dispone de una gran flexibilidad tanto para escoger el tipo de estructura y su configuración, como para elegir los subobjetos. Esta aproximación también permite introducir cómodamente conocimiento "a

priori", simplemente imponiendo restricciones a la estructura o difiniéndola "ad-hoc". Más importante aún, al contrario que en la aproximación geométrica, en la aproximación estructural es posible manejar espacios semánticos grandes e incluso infinitos (p.e.: para realizar traductores), así como obtener para cada objeto reconocido, no sólo la información de pertenencia a la forma, sino también una descripción en términos de subobjetos (la estructura) (p.e.: la secuencia de fonemas que forman la palabra reconocida).

En general pues, se emplea la aproximación estructural en los casos en que la aproximación geométrica es insuficiente, y en los que la información de estructura es de importancia relevante o fácil de utilizar (figura 1.4).

El que la aproximación estructural sea comparativamente más reciente que la geométrica, y el que su complejidad sea mayor, hacen que todavía quede mucho por hacer. En particular, una de las mayores dificultades, aún por resolver eficazmente, reside en la extracción de los subobjetos y su estructura a partir de un objeto complejo, operación que cae dentro del abierto problema de la *segmentación* (ver apartado siguiente). A pesar de ello, la ambición creciente de los diseñadores hace que la aproximación estructural se emplee cada vez más.

1.1.5 Segmentación

Uno de los problemas que más trabajos ha generado en el campo del reconocimiento de formas es el de la *segmentación*: la separación de los subobjetos de un objeto, o más específicamente, del fondo que lo rodea (en cuyo caso el problema es el de *detección de fronteras*).

La solución puede ser sencilla, cuando el objeto se halla sobre un fondo tiene una característica obvia que lo diferencia (mucha menor amplitud, color distinto, ...), o puede ser extremadamente difícil, cuando el objeto se mezcla con otros objetos, o forma parte integrante de un objeto más complejo (la pata de una silla, un fonema en una palabra) (figura 1.5). Ello explica el que generalmente, y en aras de la simplicidad, los sistemas no muy ambiciosos de reconocimiento de formas procuren evitar la segmentación. La mayoría de los experimentos que se encuentran en la literatura tratan de reconocer objetos *aislados* (en un fondo muy caracterizable), o como mucho objetos sumergidos en un ruido que aumenta su distorsión, pero no los mezcla con otros objetos. Incluso en el caso de reconocedores estructurales es posible obviar la segmentación utilizando como subobjetos particiones "naturales" de los parámetros que definen el objeto total (p.e., en reconocimiento de la palabra: un grupo de parámetros corresponde a un intervalo de tiempo, el siguiente grupo al siguiente intervalo, la estructura global es la secuencia de dichos grupos).

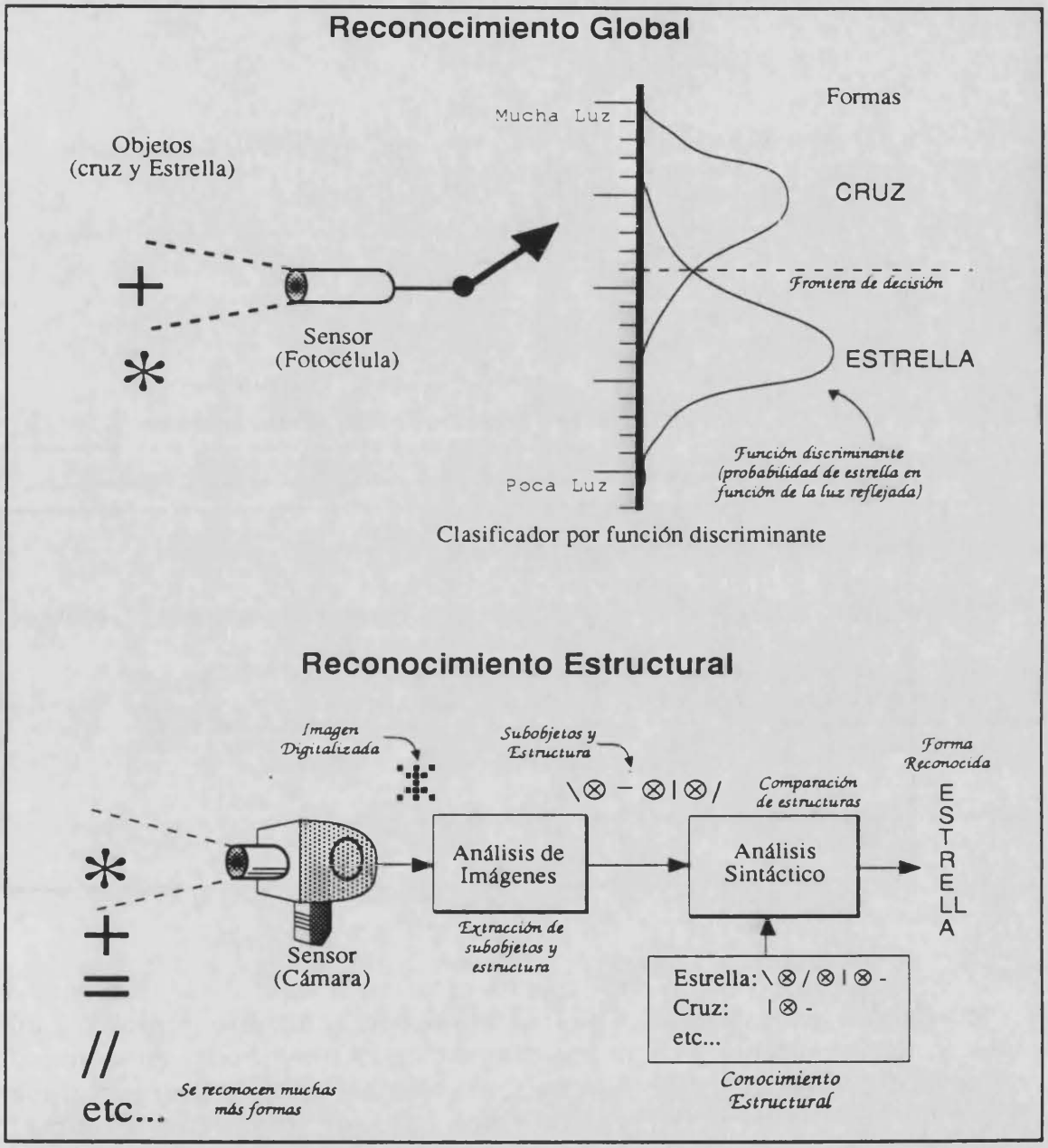


Figura 1.4 Comparación entre un método de reconocimiento global y otro sintáctico para la misma tarea. Obsérvese la diferencia de complejidad y potencia.

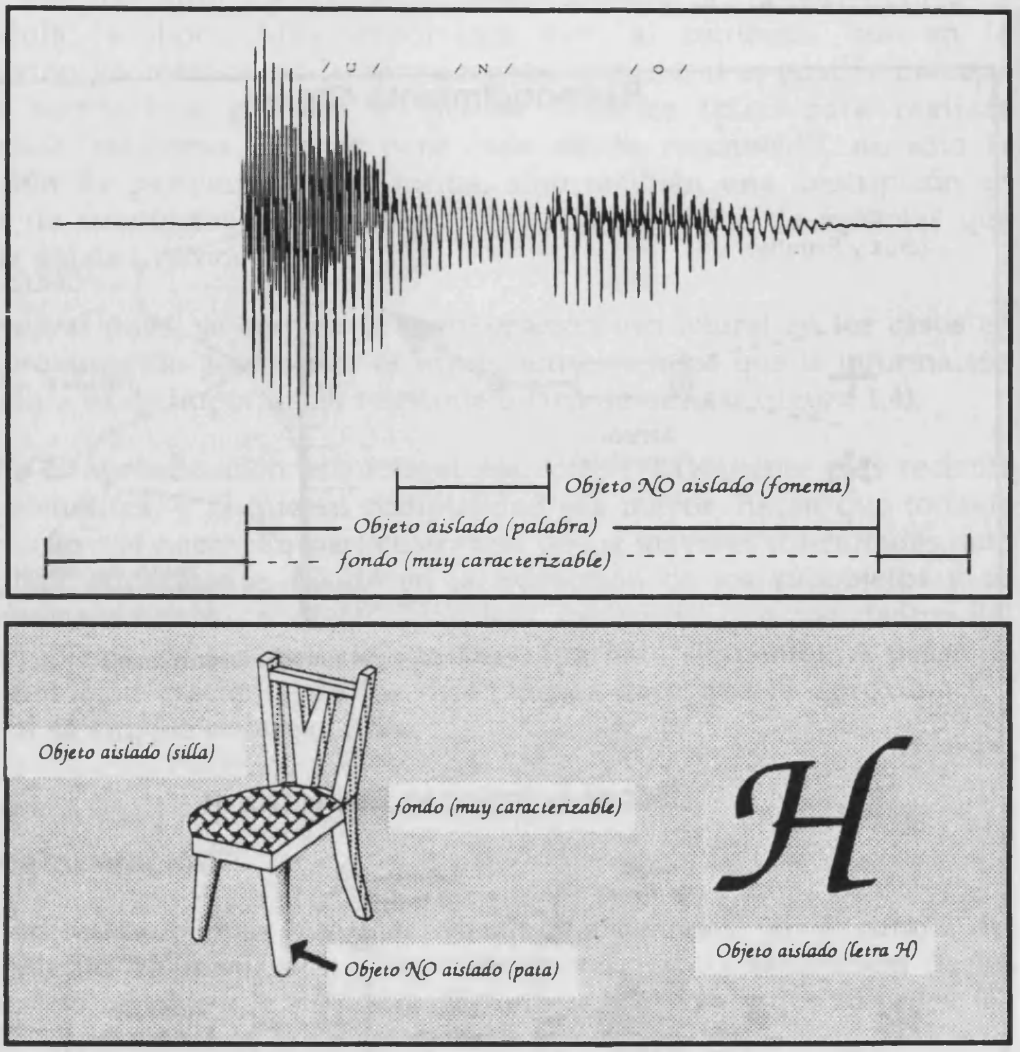


Figura 1.5 Formas aisladas y no aisladas. Segmentación.

Sólo una vez superado (en primera aproximación) el reconocimiento de objetos aislados, es cuando se aborda la segmentación. Esta representa aún un escollo importante para el logro de sistemas que operen en el mundo real (p.e.: el reconocimiento de la palabra continua, de la escritura enlazada), y aunque se puede demostrar que una solución automática (óptima o subóptima) es posible, los algoritmos conocidos hoy en día los siguen siendo muy costosos [Vidal,90].

En este trabajo sólo se tratará con objetos aislados, aunque no se descarta una extensión a objetos compuestos.

1.1.6 El tamaño de las subestructuras

Entre las muchas características relevantes de un objeto a la hora de reconocerlo, se encuentra una de especial interés para este trabajo: el tamaño (longitud) de las subestructuras.

El tamaño relativo (espacial o temporal) de los componentes de un objeto suele ser de carácter discriminativo, siendo por lo tanto a menudo parte integral de la descripción de una forma: no es posible distinguir un objeto de otro sin tener en cuenta lo grande o pequeño de las subformas que los componen.

De entre los muchos ejemplos posibles podemos mencionar: la acentuación de una sílaba (que a menudo sólo se percibe por su mayor duración), que distingue una palabra de otra; la duración de otra sílaba, que no cambia el significado de la palabra; la longitud de un brazo en la imagen de un cromosoma, que lo diferencia de otro; la longitud de un trazo en la imagen de una letra, que hace que sea esa y no otra... (figura 1.6).

Figura 1.6 La importancia del tamaño para el reconocimiento de formas.

A pesar de la importancia de esta característica, a menudo no se la ha tenido explícitamente en cuenta a la hora de estudiar y diseñar una determinada representación de objetos o formas, lo que conduce a que muchos modelos ampliamente utilizados hoy en día tengan una fuerte disfunción en este campo. Actualmente se tiende a introducir de manera más o menos explícita la representación de la longitud de las subestructuras, ya sea complicando modelos ya existentes o, como es el caso en el presente trabajo, desarrollando nuevos modelos que tengan implícita a priori dicha representación [Rulot,87].

1.1.7 El Aprendizaje Automático

Sea cual sea el modelo escogido para representar las formas, será necesario proporcionar el conjunto de éstas al sistema antes de que sea posible utilizar el módulo de interpretación. Desgraciadamente, en general **no se conoce** la descripción exacta de una forma en los términos del modelo elegido (y normalmente en ningún otro), lo que obliga a dotar al sistema de la capacidad de *aprender* (inferir) [Angluin,83] (el conjunto de) las formas.

1.1.7.1 Taxonomía del aprendizaje

El proceso de aprender una forma se puede llevar a cabo mediante instrucción directa por parte del "maestro", es decir mediante transferencia sin más del procedimiento para reconocerla (p.e: mediante programación, introducción de reglas). Ello se conoce como *aprendizaje deductivo*. Por el contrario, en el *aprendizaje inductivo*, el sistema debe llevar a cabo algún proceso de abstracción (generar formas) por sí mismo. Ello hace imprescindible la utilización de ejemplos, que el sistema analiza y clasifica sin ayuda (aprendizaje *no supervisado*) o con ayuda del maestro (aprendizaje *supervisado*) (figura 1.7).

En el caso del aprendizaje supervisado *activo o informante*, el sistema estudia los ejemplos y sugiere otros nuevos; el maestro le dice cómo clasificarlos. Si el aprendizaje es *pasivo o textual* el sistema no genera nuevos ejemplos y la labor del maestro se limita a clasificar los ejemplos iniciales. Los ejemplos en el aprendizaje supervisado pueden ser sólo *positivos* (son de esa forma) o positivos y *negativos* (no son de esa forma).

En general en el aprendizaje inductivo las únicas abstracciones que se le piden al sistema se refieren a la generación de nuevas formas. El resto de la información (representación más adecuada, tipo de estructura, método de identificación apropiado, etc...) se le proporciona de manera deductiva. Nada impide sin embargo que el sistema sea capaz de abstraer (generar hipótesis) en este campo también, aunque usualmente no es necesario (ni fácil).

1.1.7.2 Factibilidad del aprendizaje Inductivo

El método usual de aprendizaje inductivo es la *presentación de ejemplos*. Se debe disponer de una serie de objetos del mundo real, etiquetados o no con la forma a la que pertenecen. Estos objetos se muestran al sistema, y éste, mediante un proceso de *inducción*, debe inferir la descripción de la forma acorde con el modelo de representación e identificación que esté empleando.

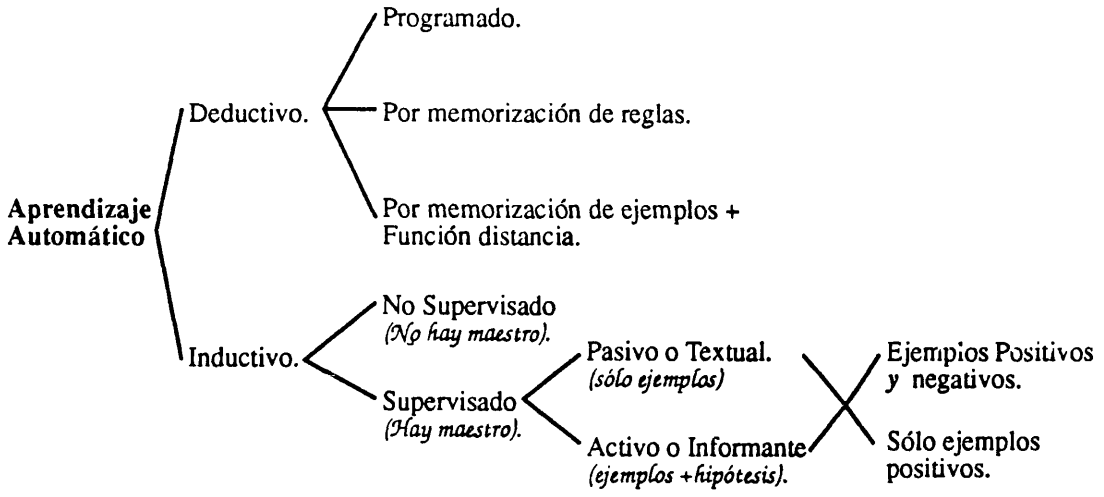


Figura 1.7 Taxonomía del aprendizaje.

Es evidente que la *convergencia* de este proceso (¿el sistema terminará por aprender al cabo de un número suficiente de ejemplos?) está lejos de ser obvia, y depende fuertemente del tipo de modelo de representación e identificación utilizado. Determinados resultados teóricos [Gold,67] afirman incluso que en el caso más general la convergencia es indemostrable.

En efecto, considerado formalmente, el aprendizaje o *inferencia* inductivos es un proceso de obtención de *reglas* a partir de *ejemplos*. De la manera más general posible, una regla es una *función recursiva parcial* **f** y un ejemplo es un par (**x**,**f**(**x**)), donde **x** pertenece al dominio de **f**. Un algoritmo de inferencia sintetiza el algoritmo de **f** a partir de sus ejemplos. Gold afirma que el inferir una función recursiva en general es un problema indecidible (!!). Afortunadamente el mismo Gold demuestra que si nos limitamos a las funciones recursivas *primitivas* sí que se las puede identificar en el límite a partir de ejemplos.

1.2 El Reconocimiento Automático del Habla

Como dice su nombre, el "Reconocimiento Automático del Habla" (RAH) pretende conseguir que las máquinas (usualmente ordenadores) sean capaces, tanto de obedecer órdenes expresadas oralmente, como de dialogar interactivamente con seres humanos mediante el uso de la palabra hablada.

El problema de la *síntesis del habla*, implícito en esta última y mayor pretensión, se puede dar hoy en día como resuelto, aunque en la práctica queden por pulir algunos "detalles" (calidad de la voz, entonación, ...) para conseguir una voz indistinguible de la de un ser humano [Casacuberta,87].

En RAH sin embargo, el panorama no es tan esperanzador. Las dificultades son debidas a una larga lista de factores, los cuales en su mayor parte dan cuenta de la mayor o menor variabilidad presente en los objetos a reconocer, y por lo tanto de la cantidad de información que necesitará el sistema para poder conseguir resultados aceptables. Entre estos factores cabe destacar:

- El *vocabulario*: número de palabras y/o frases a reconocer.
- La cantidad de locutores, utilizándose los términos: *Monolocutor* (el sistema reconoce con un único locutor), *Multilocutor* (sólo funciona con los locutores que conoce) e *Independiente del Locutor* (reconoce con cualquier locutor).
- El *sexo* de los locutores: como es de común conocimiento, la voz femenina es distinta de la masculina: no sólo cambia el tono fundamental de un sexo a otro, sino que la variación de formantes no es proporcional a la variación de dicho fundamental.
- La *calidad* de la señal vocal: la señal está afectada por *ruido*, procedente del ambiente (fábrica, cabina de avión,...) o de una mala transmisión (líneas telefónicas).

Los sistemas de RAH se pueden subdividir actualmente en dos grupos bien diferenciados por la complejidad del problema que intentan resolver: los que se dedican únicamente al reconocimiento de *palabras aisladas* (separadas unas de otras por silencios) y los que se dedican al reconocimiento de la *habla continua* (frases más o menos largas sin separación entre palabras).

En el habla continua, la cantidad de combinaciones de palabras (número de frases) posibles es extremadamente elevada, y es infactible el disponer de muestras de todas las combinaciones para realizar un aprendizaje fiable de cada una. Algo similar ocurre con el aprendizaje de modelos de palabras aisladas cuando el vocabulario crece. Todo ello obliga a recurrir a la descomposición en subunidades subléxicas (fonemas, palabras,...) y a utilizar algún método de representación estructural, que limite el número de posibilidades (p.e.: el número de palabras que pueden venir a continuación de una dada: la *perplejidad* del lenguaje). Como anteriormente se ha mencionado, esto lleva inevitablemente a enfrentarse con el problema de la segmentación, agravado en este caso por la *coarticulación* que existe entre las distintas unidades, que altera a éstas fuertemente en los puntos de unión.

Es sin embargo en estos sistemas en los que se centra actualmente la investigación, pues sólo mediante ellos se podrá resolver el problema del RAH.

1.2.1 Adquisición, parametrización y etiquetado en RAH

En reconocimiento del habla, los objetos son señales sonoras (ondas de presión en el aire) que representan palabras o frases. Es pues necesario utilizar una etapa de representación, que transforme estas señales en algo más conveniente para su utilización por el módulo de interpretación. En la actualidad, la etapa de representación de objetos en reconocimiento del habla está típicamente compuesta por tres subniveles: el preproceso, la parametrización y el etiquetado (que puede considerarse un nivel superior de parametrización) (figura 1.8).

- El primer subnivel, el del *preproceso* está formado por el conjunto mecánico - eléctrico - electrónico constituído por el micrófono (que transforma la onda sonora de presión en onda eléctrica), el filtro (que suprime componentes indeseables de la onda eléctrica) y el conversor analógico/digital (AD) (que transforma la onda eléctrica en una serie de medidas de amplitud).

- El segundo subnivel, el de *parametrización*, tiene como objetivo típico el de reducir la enorme cantidad de información proveniente del nivel anterior (≈120000 bits/sg.) en algo más manejable (≈5000 bits/sg.).

  La parametrización efectúa usualmente un cambio de espacio de representación, pasando de un espacio de una dimensión (tiempo) a otro de dos dimensiones (típicamente tiempo/frecuencia), siendo por lo tanto una transformación de tipo estrictamente matemático, que transforma una serie de medidas de amplitud en una serie de vectores de parámetros [Casacuberta,87].

  El tipo de parametrización varía de un sistema a otro, los más utilizados son: el banco de filtros, los coeficientes de predicción lineal, y los coeficientes cepstrales [Makhoul,75] [Rabiner,78] [Benedí,89]. El autor, en un trabajo anterior, propuso y estudió para este fin la los valores de la función de autocorrelación de la señal muestreada a un bit [Rulot,85].

- El tercer subnivel, el *etiquetado* o *cuantificación vectorial*, no siempre se halla presente, pero es muy utilizado en los sistemas que utilizan la aproximación estructural en RAH, puesto que permite una reducción aún mayor de la cantidad de información (≈300 bits/sg.) y proporciona una representación extremadamente adecuada para aplicar los métodos estructurales (gramáticas, Modelos de markov,...).

  El etiquetado se lleva a cabo normalmente mediante algún tipo de análisis estadístico que permite clasificar los vectores de parámetros del nivel anterior en una serie reducida de clases, cuyos nombres o

símbolos son los que se utilizan en lugar del vector original [Duda,73] [Gray,84].

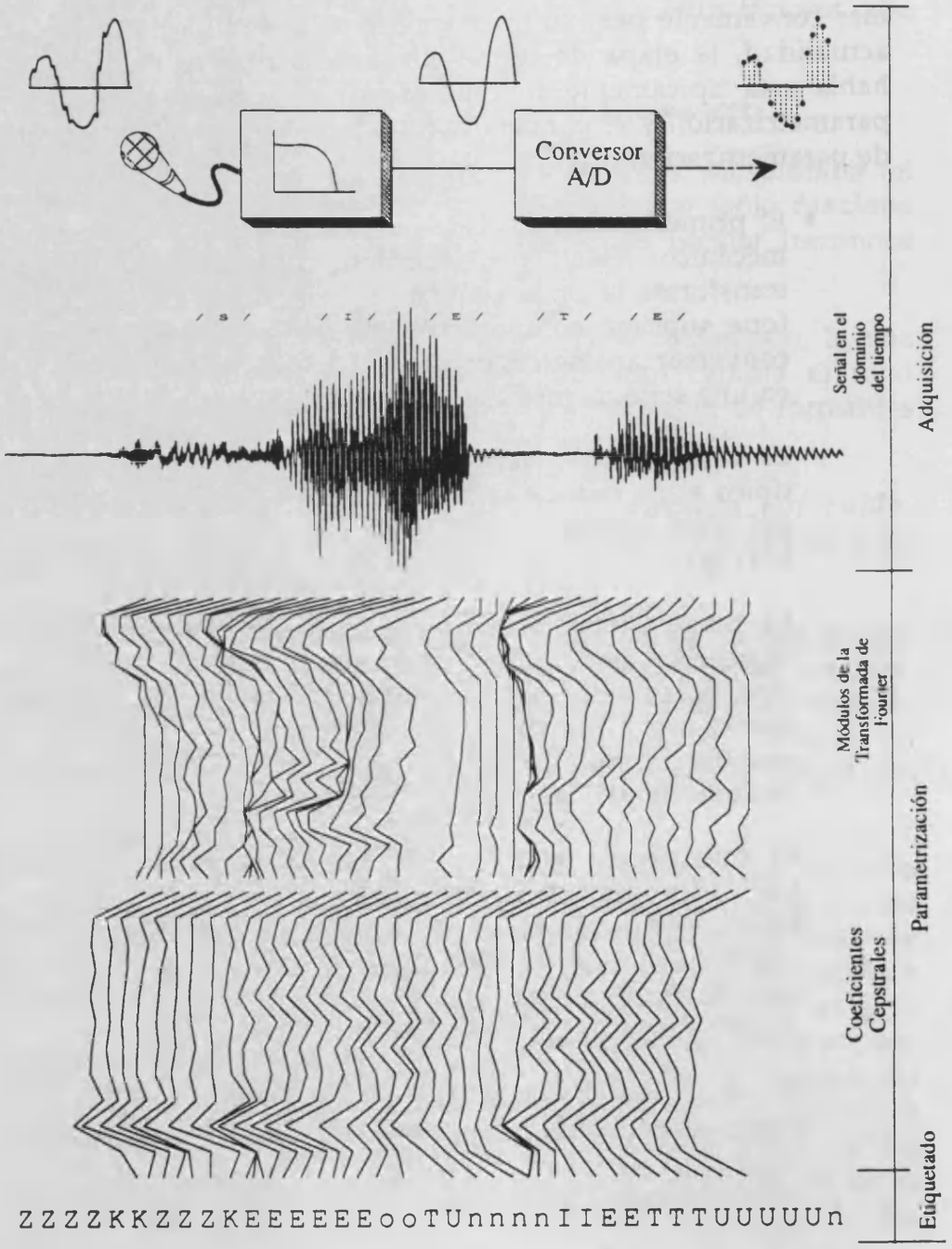


Figura 1.8 Adquisición, parametrización y etiquetado en RAH.

1.2.2 Estado del Arte

Actualmente existen sistemas capaces de reconocer con porcentajes de aciertos superiores al 98% vocabularios *sencillos* (palabras bien distintas fonéticamente) y *pequeños* (del orden del centenar de palabras) de palabras aisladas, todo ello en entornos independientes del locutor y con locutores de ambos sexos. Estos resultados se obtienen incluso en ambientes ruidosos [Rabiner,87] [Loeb,87] [Watanabe,88], pero aunque son suficientes para determinadas aplicaciones puntuales, no permiten una comunicación hombre-máquina realmente fiable. En el estado actual de la técnica, conseguir más del 99% de aciertos es posible únicamente imponiendo condiciones más restrictivas, por ejemplo suprimiendo el ruido (este trabajo) o restringiéndose a locutores de un único sexo [Lippmann,87].

La mayoría de estos sistemas se basan en métodos globales, aunque la tendencia actual es utilizar un modelo estructural proveniente de la teoría de la información: los *HMM* ("Hidden Màrkov Models": Modelos Ocultos de Markov) [Rabiner,83]. Empleando estos modelos, con fonemas o unidades subfonéticas como subformas, es posible manejar grandes vocabularios y obtener resultados esperanzadores. Por ejemplo, [Jouvet,86] con 1000 palabras obtiene hasta un 95% de reconocimiento mientras que [D'Orta,87] con 3000 palabras consigue hasta un 84% pero con un sistema dependiente del locutor.

Como se observa, la tasa de reconocimiento baja rápidamente en función de la complejidad de la tarea y se halla aún muy lejos de la máxima alcanzable, dada por la obtenida por seres humanos. Por ejemplo, considérese un vocabulario especialmente difícil: el *e-set*, formado por las letras que se pronuncian con /e/ (/be/, /ce/, /de/, /ge/, /pe/,...). Con él, los seres humanos consiguen un 98% de palabras reconocidas [Bahl,87]. Los sistemas actuales, a pesar de ser sólo 9 palabras, consiguen en el mejor de los casos, un 92% [Bahl,87] (multilocutor). Otro ejemplo, en el que la dificultad se debe a la enormidad del vocabulario, lo constituye el *Tangora* [Averbuch,87] [Cerf-Danon,91]. Este sistema, con un vocabulario de 20000 palabras, reconoce en promedio un 97% de las mismas, eso sí, es un sistema dependiente del locutor, considera solo palabras aisladas y restringe fuertemente la variabilidad del lenguaje mediante estadísticas lingüisticas. Los resultados de Tangora no varían sensiblemente cuando se le aplica a distintas lenguas europeas (ha sido probado en inglés, francés, alemán, italiano y español).

En el reconocimiento del habla continua, los resultados de reconocimiento a nivel de palabra se reducen drásticamente, haciéndose imposible obtener más de un 80% de reconocimiento si no se imponen restricciones lingüísticas. En este campo, los participantes en el proyecto *SPICOS* han desarrollado un sistema de reconocimiento que con un vocabulario de 1000 palabras reconoce un 75% de ellas a partir de frases, y ello independientemente del locutor [Ney,91]. El mismo sistema, cuando se

le impone un modelo de lenguaje que limita la perplejidad media a 100 obtiene incluso un 91% de tasa de reconocimiento de palabras. Por su parte, *Dragon Systems*, con un sistema que se adapta al locutor y con vocabularios de 5000 palabras de perplejidad media 140, obtiene un 94.2% de reconocimientos de palabras [Baker,91]; mientras que con otro vocabulario de 3400 palabras, pero de perplejidad mucho mayor (430), logra un más de un 85%. La misma compañía afirma que tras una adaptación de con alrededor de 2000 palabras, se reconocen textos formados a partir de un vocabulario de 25000 palabras con un porcentaje de aciertos de palabras del orden de 87%. Experimentos multilengua con este sistema (se han probado las mismas del *Tangora* más el holandés) han proporcionado resultados medios de 85%, con muy poca variación de una lengua a otra, en un vocabulario de 2000 a 4000 palabras [Bamberg,91]. También de gran interés es el sistema SPHINX [Lee,88], que con un vocabulario de 1000 palabras obtiene un 70,6% de aciertos en palabras a partir de frases (independientemente del locutor), consiguiendo incluso hasta un 96% si, con el mismo vocabulario, se restrige la perplejidad (20) mediante un modelo de lenguaje.

1.3 El Reconocimiento Automático de Caracteres

El reconocimiento de imágenes puede decirse que tuvo su comienzo en 1870, cuando Carey inventó el Scanner Retina, un sistema de transmisión de imágenes que usaba un mosaico de fotocélulas. Sin embargo, el verdadero impulso se dió con el invento del Scanner Secuencial por Nipkow, que más tarde daría lugar a la televisión, y por la aparición de los ordenadores al final de la década de los 40.

De entre los múltiples sistemas que desde entonces han tratado imágenes para su reconocimiento (fotos de satélites, imágenes de entorno para los sistemas de visión, piezas a clasificar, etc...), pronto destacaron los *reconocedores ópticos de caracteres* (OCR), por la cantidad de aplicaciones prácticas inmediatas que permitían vislumbar (ayuda a ciegos, comprobación de firmas, pero sobre todo ofimática y correo) y su relativa sencillez (imágenes planas, sin sombras, número limitado de formas).

No obstante los primeros logros en reconocimiento de caracteres los consiguiera Tyurin en 1900, y hubieran otros intentos memorables como el Optófono de Fourier d'Albe (1912) y el sistema táctil de Thomas (1926), las aplicaciones comerciales del reconocimiento de caracteres sólo tuvieron lugar a mediados de los '40, con el desarrollo de los primeros reconocedores de imágenes y la aparición de pioneros como David Shepard, el fundador de "Intelligent Machine Research Co." [Mantas,86].

Las posibles tareas a las que se puede asignar un OCR se pueden clasificar [Mantas,86] (figura 1.9), por orden de dificultad, según el reconocimiento sea de caracteres...

- **Impresos** (*Fixed Font* y *Multi Font* Character Recognition (CR)).
- **Trazados** (en tableta gráfica o similar), en los que además de la imagen del carácter se dispone de la información temporal de su trazado (*On-Line* CR).
- **Manuscritos**, caracteres también escritos a mano, pero separados y no caligráficos (*Handwritten* CR).

Por otra parte, si los caracteres no están aislados, sino que se encadenan unos con otros para formar palabras, se habla de reconocimiento de **Escritura** (*Script* CR). Aquí la dificultad es máxima, al presentarse en toda su magnitud el problema de la segmentación.

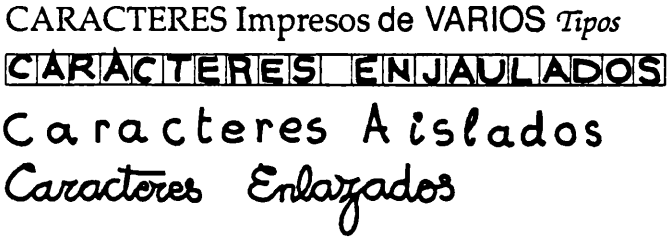


Figura 1.9 Distintos grados de dificultad en reconocimiento de caracteres

Aparte de la manera como están escritos, otras posibles fuentes de dificultad en el reconocimiento de caracteres, si nos restringimos a alfabetos occidentales (ni que decir tiene que el problema de los símbolos chinos, árabes, etc... presenta complicaciones adicionales), son debidas a lo parecido de ciertas letras y/o dígitos (U-V, C-L, a-d, n-h, I-1, Z-2, S-5, G-6), que incluso en algunos casos pueden escribirse igual (O-0, l-1) o casi indistinguiblemente de sus correspondientes en mayúsculas (O-o, K-k, C-c), lo que a menudo obliga a recurrir al contexto o incluso a su posición con respecto a la línea base (P-p, Y-y) para diferenciarlas.

1.3.1 Adquisición, reducción de ruido y parametrización en RAC

Ya que trata directamente con objetos del mundo real, en RAC también es necesario obtener una representación del objeto externo a reconocer, que sea adecuada para el módulo de interpretación. En el caso de RAC, el módulo de representación comporta usualmente 3 subniveles: *preproceso*, *reducción de ruido* (que en realidad es un preproceso de nivel superior) y *parametrización*.

- El nivel de *preproceso* consta normalmente (en reconocimiento óptico) de un dispositivo mecánico-electrónico (una cámara o un scanner óptico) que *barre* la imagen horizontal y verticalmente con

el fin de transformarla en una serie de medidas de intensidad luminosa; y de un digitalizador, que transforma esas medidas (expresadas normalmente en voltajes eléctricos) en números binarios que transfiere al ordenador.

Alternativamente (reconocimiento de caracteres trazados), lo que se obtiene y posteriormente se digitaliza son las coordenadas sucesivas de la posición del lápiz, junto con información de si éste toca la superficie o no (tableta gráfica o pantalla sensible).

- Durante la etapa de *reducción de ruido*, se procesa la matriz bidimensional obtenida en la etapa anterior mediante variados algoritmos de tratamiento de imágenes, con el fin de suprimir y/o realzar determinadas características: suavizado, realze de bordes, realze de contrastes, supresión de grises mediante umbral, submuestreo, supresión de puntos aislados, normalización de tamaño, alineamiento con línea base, normalización de orientación, esqueletización, etc... En todos los casos este proceso, fundamentalmente matemático, proporciona una matriz (imagen) similar a la inicial pero a menudo con mucha menos información.

  Muchos de estos tratamientos son innecesarios cuando se tiene la secuencia de posiciones obtenida de un carácter trazado [Tappert,90], aunque de igual manera se aplican suavizados, filtrados normalizaciones, etc...

- La *parametrización* permite a continuación obtener una descripción adecuada de la imagen en función de lo que se quiere reconocer y reducir aún más la cantidad de información. En el caso de los caracteres puede prescindirse de este paso (se reconoce simplemente la imagen mediante procedimientos globales), o bien pueden utilizarse transformaciones globales (de Karnhunen-Loeve, de Fourier, cálculo de momentos,...). También pueden extraerse propiedades locales (puntos extremos, ángulos, uniones en T y cruces,...) y/o geométricas (segmentos, curvaturas,...), o pueden buscarse representaciones estructurales (conversión a un grafo, a una secuencia de direcciones, descripción topológica, descripción en base a un conjunto de figuras elementales,...).

1.3.2 Estado del Arte

Existen en la actualidad gran cantidad de programas de OCR comercializados. Están principalmente destinados al reconocimiento de caracteres impresos (Multi-Font CR), y funcionan en su mayoría en ordenadores personales tipo PC-IBM Compatible o Macintosh. Su tasa de reconocimiento normalmente se halla entre **80%** y **95%**, obteniendo desde luego los mejores resultados cuando funcionan con tipos de letra para los

que han sido "afinados" o entrenados [Robinson,90]. Estos sistemas incluyen a menudo "reconocedores de composición", siendo capaces de separar columnas y bloques de texto y de distinguir a éstos de las figuras.

La literatura científica hoy en día se halla más centrada en el reconocimiento de caracteres manuscritos, que en cierta manera puede considerarse un "superconjunto" del de los caracteres impresos, y que desde luego involucra una mucho mayor dificultad dada la mucha mayor variabilidad (¿quién no se ha tropezado con una letra "de médico"?). En este campo, y restringiéndose al reconocimiento de caracteres aislados, se obtienen actualmente unas tasas de reconocimiento de **98.3** a **99%** en dígitos aislados [Kurosawa,86] [Shridar,86] [Baptista,88].

En condiciones poco favorables (enorme número de escritores, condiciones de escritura incontroladas) como es el caso cuando se quieren reconocer los códigos postales (ZIP codes), los resultados obviamente empeoran, lográndose un promedio de 92% de reconocimiento, variando entre **85%** y **97%** [Lam,88] [Nadal,90] [Kimura,91]. Lo mismo ocurre si se intenta reconocer un conjunto de formas mayor que el de los dígitos, como las 26 letras inglesas: 88% [Brown,88] o el kanji: **86.7%** (con símbolos difíciles) [Sekita,88].

Cuando se intenta reconocer escritura *enlazada*, se puede, a pesar de todo, conseguir resultados muy buenos aprovechando la información contextual que supone la palabra: **92,5%** de aciertos en palabras [Kundu,89], o limitando el reconocimiento a caracteres trazados: **89.9%** en caracteres (letras inglesas mayúsculas y minúsculas) [Fuyisaki,90].

2

Los Métodos Sintácticos

2.1 Introducción

Entre los métodos estructurales de reconocimiento de formas destacan los métodos englobados en lo que se conoce como *reconocimiento sintáctico de formas*. Estos métodos intentan aprovechar las técnicas desarrolladas por la *teoría de lenguajes formales* [Aho,73] [Eilemberg,74] [Harrison,78], las cuales proveen de una representación (las gramáticas) y de un mecanismo de interpretación ("parsing" o *análisis sintáctico*) para aquellas formas cuyos objetos se pueden describir como cadenas de subobjetos [Gonzalez,78] [Fu,82] [Miclet,86].

En su utilización en reconocimiento de formas, la teoría de lenguajes tropezó desde un principio con un problema inherente a este dominio de aplicación: la representación imprecisa de los objetos. Para solventar esta dificultad se hace necesario recurrir a métodos de análisis sintáctico tolerantes a cadenas "ruidosas" o plagadas de errores. Los más desarrollados de estos métodos son el *análisis sintáctico corrector de errores* y el *análisis sintáctico estocástico*, que utilizados simultáneamente permiten construir *analizadores sintácticos correctores de errores estocásticos.*

2.2 Los lenguajes y sus gramáticas

Dado un conjunto de símbolos o *alfabeto* V, el conjunto de todas cadenas posibles sobre ese alfabeto es V^* (el monoide libre sobre V mediante el operador concatenación) y V^+ será el mismo conjunto sin la cadena vacía. Un *lenguaje* sobre el alfabeto V es un subconjunto de V^*. Existen un

número no enumerable de lenguajes, todos ellos representables mediante *gramáticas*.

Una gramática se define como la cuádrupla G=(N,V,P,S), donde V es el *alfabeto* de G y N es un conjunto finito de símbolos ***no terminales***, $V \cap N = \emptyset$. Se conoce a $W=V \cup N$ como el *vocabulario* de G, una *frase* $\zeta \in W^*$ es una cadena de símbolos de W. $P \subset W^* N W^* \times W^*$ es un conjunto de *reglas de producción* o reescritura, que transforman una frase (en la que por lo menos hay un no terminal) en otra frase del mismo vocabulario. Estas reglas se denotan como $\zeta_1 A \zeta_2 \rightarrow \zeta_3$, donde $\zeta_1, \zeta_2, \zeta_3 \in W^*$ y $A \in N$. Finalmente, $S \in N$ es el símbolo no terminal *inicial* o *axioma* de G.

Mediante la aplicación sucesiva de reglas de G se puede transformar una frase $\zeta 1$ del vocabulario en otra ζ_n. Escribiremos esto como $\zeta_1 \stackrel{*}{\Rightarrow} \zeta_n$; $\zeta_1, \zeta_n \in W^*$; y si $D(\zeta_n)=(\zeta_1, \zeta_2, \zeta_3, \ldots, \zeta_n)$, $\zeta_i \in W^*$, es la secuencia de frases que han llevado de ζ_1 a ζ_n, diremos que D es una *derivación* de ζ_n desde ζ_1. Se define entonces el *lenguaje generado por G* como el conjunto de todas las cadenas **del alfabeto** que se pueden derivar del axioma de G: $L(G)=\{ \alpha : \alpha \in V^*, S \stackrel{*}{\Rightarrow} \alpha \}$.

Al proceso de obtener una derivación de una cadena a partir del axioma de una gramática y aplicando reglas de ésta, se le denomina *análisis sintáctico*. Se dice que una gramática es *ambigua* si para alguna cadena del lenguaje existe más de una posible derivación para generarla.

Chomsky dividió las gramáticas (y por lo tanto los lenguajes) en una jerarquía que va del tipo 0 a 3, en orden decreciente de complejidad, y en base a la forma de sus reglas:

Tipo 0: Gramáticas sin restricciones en las reglas.

Tipo 1: Gramáticas *sensibles al contexto*, con reglas de la forma $\zeta_1 A \zeta_2 \rightarrow \zeta_1 \beta \zeta_2$. Es decir, el no terminal A se sustituye por la frase β del vocabulario en el *contexto* $\zeta_1 \zeta_2$ ($\zeta_1, \zeta_2 \in W^*$, $\beta \in W^+$).

Tipo 2: Gramáticas de *independientes del contexto*, en las que $A \rightarrow \beta$ independientemente del contexto.

Tipo 3: Gramáticas *regulares*, cuyas reglas son de la forma $A \rightarrow aB$ o $A \rightarrow a$; $a \in V$; $A,B \in N$.

La gramáticas de tipo 0 y 1 son las que proporcionan el mayor poder descriptivo, aunque son las gramáticas del tipo 2 y 3 las más utilizadas en aplicaciones prácticas como el reconocimiento de formas, principalmente debido a su mucho menor complejidad. Todas ellas son las llamadas *gramáticas formales*.

2.3 Reconocedores de lenguajes

Según se ha definido en el apartado anterior, para cada lenguaje existe una gramática generadora del mismo. Similarmente, existe un *reconocedor* capaz de decidir si una cadena pertenece o no a dicho lenguaje.

Para los lenguajes regulares el reconocedor es un *autómata finito*. Se demuestra que para todo autómata finito existe una gramática regular que genera el lenguaje aceptado por el autómata y viceversa [Aho,73].

Un autómata finito se define como una quíntupla $A=(V,Q,\delta,q_0,F)$, donde V es un conjunto finito de símbolos y Q un conjunto finito de *estados*. q_0 es el *estado inicial* y $F \subset Q$ el conjunto de *estados finales*. δ una *función de transición* entre estados: $\delta:(Q\times V)\rightarrow \mathcal{P}(Q)$. El funcionamiento de un autómata se inicia en q_0 para ir pasando, mediante la función de transición, de un estado al estado siguiente según indiquen los sucesivos símbolos de la cadena a reconocer. Si después del último símbolo de la cadena se ha llegado a un estado final, la cadena *pertenece* al lenguaje del autómata (figura 2.1). Formalmente, el *lenguaje de cadenas de V^* aceptado por el autómata A* se define como $L(A)=\{\alpha \mid \alpha\in V^* \wedge \Delta(q_0,\alpha)\cap F\neq\emptyset\}$, donde $\Delta:(Q\times V^*)\rightarrow \mathcal{P}(Q)$ es la función de transición δ extendida a cadenas de símbolos (λ es la cadena vacía):

$$\Delta(q,\lambda)=\{q\};\quad \Delta(q,\alpha a)=\bigcup_{q'\in\Delta(q,\alpha)}\delta(q',a);\qquad \alpha\in V^*;\ a\in V;\ q\in Q;$$

Cada una de las posibles secuencias de estados recorridos por el autómata para reconocer una cadena, equivale a una secuencia de reglas de la gramática (regular) equivalente, y por lo tanto, representa una *derivación* de la cadena. Nótese que en general, el funcionamiento de un autómata implica el seguir simultáneamente varios posibles caminos a través del mismo (y posiblemente llegar a varios estados finales), puesto que, según la definición de δ, dado un símbolo y un estado son varios los estados siguientes (el autómata es *no determinista*). Para recorrer en paralelo varios caminos se suelen utilizar algoritmos basados en técnicas de *programación dinámica* (algoritmo de Viterbi, ver capítulo 4) [Bellman,57] [Forney,73]. Si la definición de δ se restringe a $\delta:(Q\times V)\rightarrow Q$ entonces el autómata es *determinista*. Se demuestra que dado un autómata no determinista siempre existe uno determinista que reconoce el mismo lenguaje (pero puede tener $2^{|Q|}$ estados) y viceversa (figura 2.2) [Aho,73].

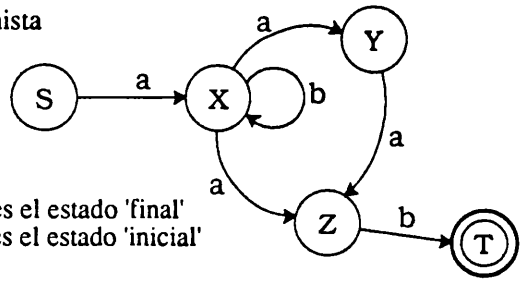
Sea el Lenguaje:

Todas las cadenas que empiezan con 'a', seguida de un número indeterminado de 'b', y que terminan con 'ab' o 'aab'

Su Gramática:

G=(V,N,P,S); V={'a','b'}; N={S,X,Y}
P={
S→aX (axioma)
X→bX (un bucle)
X→aZ, X→aY > G es NO determinista
Y→aZ
Z→b
}

Su atómata asociado:



T es el estado 'final'
S es el estado 'inicial'

Reconocimiento de cadenas: 'abba' y 'ab'

1) Secuencia de estados:
abbab: SXXXZT, reconocida.
ab: SXX, NO reconocida.

2) Secuencia correspondiente de reglas:
abbab: S→aX, X→bX, X→bX, X→bX, X→aZ, Z→bT
ab: S→aX, X→bX, X→???

Figura 2.1 Un ejemplo de Lenguaje regular, con su Gramática y su Autómata asociado. Análisis sintáctico de 2 cadenas.

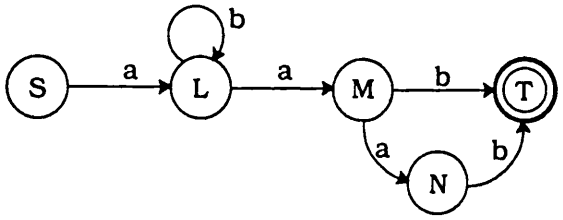


Figura 2.2 Autómata determinista equivalente al no determinista de la figura anterior. Nótese que reconoce el mismo lenguaje, pero NO representa la misma gramática (aunque sí a una equivalente).

Para los lenguajes de contexto libre los reconocedores correspondientes son los *autómatas a pila* [Fu,82], para los lenguajes dependientes del contexto se emplean los *autómatas lineales acotados*, y finalmente, para los lenguajes de tipo 0 las *máquinas de Turing* [Fu,82].

La definición de los reconocedores de lenguajes conduce inmediatamente a una posible manera de utilizar los métodos sintácticos en reconocimiento de formas. Para ello, y siempre que los objetos sean representables mediante cadenas de símbolos, basta construir un clasificador en el que se define una gramática por clase o forma. El correspondiente

reconocedor (p.e., un autómata si la gramática es regular) determinará si la cadena pertenece o no al lenguaje de la gramática (al conjunto de cadenas que éste representa) y por lo tanto, a la forma. La construcción o inferencia de los reconocedores asociados a cada clase, o equivalentemente de sus gramáticas, se puede llevar a cabo manualmente o mediante métodos de *inferencia gramatical* (ver capítulo siguiente).

2.4 El problema de la imprecisión

Es muy usual en reconocimiento de formas el que los objetos a reconocer, e incluso los ejemplos de aprendizaje, vengan distorsionados y embrollados por ruidos de diversa procedencia (medio ambiente, líneas de transmisión en mal estado, mancha en la foto,...) (figura 2.3).

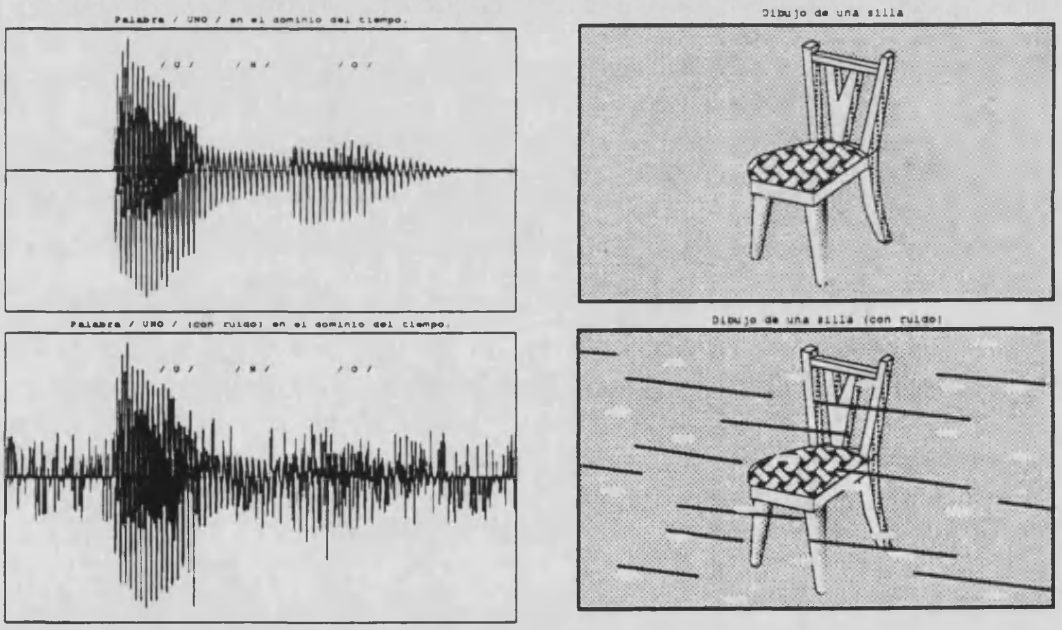


Figura 2.3 Formas "limpias" y distorsionadas por ruido.

En el caso del reconocimiento sintáctico de formas, ello quiere decir que las cadenas presentadas al reconocedor pueden no ser correctas, es decir, pueden presentar símbolos equivocados y símbolos de más o de menos. Esto a su vez conlleva el que las gramáticas, que a menudo son construídas (automática o manualmente) a partir de un subconjunto representativo de cadenas de la forma, puedan ser inexactas. Esta imprecisión, que afecta tanto en la descripción estructural de los objetos como en la de las formas, no puede deshacerse mediante los procedimientos usuales de análisis

sintáctico, por lo que se han desarrollado soluciones alternativas, las cuales se enmarcan en dos aproximaciones bien diferenciadas:

- Aquellas en las que la interpretación se lleva a cabo mediante el uso de métodos derivados de la teoría de la decisión, definiendo una distancia entre cadenas, a partir de información estadística relativa a los distintos errores debidos al ruido u otras causas [Aho,72] [Aho,73] [Thomason,74] [Bahl,75] [Tanaka,86] [Tanala,87]. Las técnicas correspondientes se engloban en el llamado *análisis sintáctico corrector de errores.*

- Aquellas que asocian a la información estructural una información probabilística, sobre la frecuencia de uso/aparición de las distintas partes de la estructura. Este suplemento de información flexibiliza notablemente la capacidad de representación del modelo, y ha conducido a la extensión de los lenguajes formales con el fin de incluir en ellos los llamados *lenguajes estocásticos*. Estos lenguajes se representan mediante *gramáticas estocásticas*, con las que se lleva a cabo un *análisis sintáctico estocástico* [Gonzalez,78] [Fu,82].

2.4.1 Análisis sintáctico corrector de errores

La idea básica del análisis sintáctico corrector de errores reside en la estimación de una *distancia* o *medida de similitud* entre una cadena distorsionada y su original perteneciente a una gramática dada.

La distancia entre dos cadenas se define a partir del "número mínimo" de *transformaciones de error* necesarias para pasar de una cadena a la otra. Se consideran tres tipos posibles de error: *borrado, inserción* o *sustitución* de un símbolo, pudiéndose pasar de una cadena a otra cualquiera simplemente mediante combinación de estos errores [Thomason,74] [Fu,82]. Formalmente, se definen tres transformaciones:

borrado: $\alpha a \beta \overset{T_b}{\vdash} \alpha\beta$;

inserción: $\alpha\beta \overset{T_i}{\vdash} \alpha a \beta$;

sustitución: $\alpha a \beta \overset{T_s}{\vdash} \alpha b \beta$

$T_b, T_i, T_s : V^* \rightarrow V^*$; $\alpha, \beta \in V^*$; $a, b \in V$; $a \neq b$;

a partir de las cuales se define la *distancia de Levenshtein* $d_L(\alpha,\beta)$ entre dos cadenas $\alpha,\beta \in V^*$ como el mínimo número de transformaciones necesarias para convertir α en β (figura 2.4). Es decir, si C es cualquier secuencia de transformaciones que convierta α en β y S_C, B_C, I_C son respectivamente el número de sustituciones, borrados e inserciones en C:

$$d_L(\alpha,\beta)=\min_{\forall C}\{S_C+B_C+I_C\}$$

4 Errores: Distancia=4

Figura 2.4 Distancia de Lenvenshtein entre dos cadenas.

Si es necesario, es posible extender esta definición dando más o menos importancia a un tipo de error frente a otro, obteniéndose entonces la *distancia de Levenshtein ponderada*:

$$d_{LP}(\alpha,\beta)=\min_{\forall C}\{w_s\cdot S_C+w_b\cdot B_C+w_i\cdot I_C\}$$

donde w_s, w_i, w_b son respectivamente los pesos (costes) de sustitución, borrado e inserción. Incluso cabe introducir una dependencia según el símbolo que se borra (se inserta o sustituye) (*distancia de Fu ponderada*) [Fu,82].

Una vez definida la distancia, un algoritmo *analizador sintáctico corrector de errores* es un algoritmo que busca una cadena β perteneciente al lenguaje de la gramática G tal que la distancia (d) entre β y la cadena a analizar α sea mínima:

$$\beta=\underset{\forall \omega\in L(G)}{\operatorname{argmin}}\{d(\alpha,\omega)\}$$

a la distancia que proporciona el mínimo se la puede considerar como la *distancia entre la cadena α y el lenguaje L(G)*:

$$D(\alpha,L(G)) = d(\alpha,\beta) = \min_{\forall \omega\in L(G)}\{d(\alpha,\omega)\}$$

Generalmente, los algoritmos de análisis sintáctico con corrección de errores se subdividen en dos etapas:

- Construcción de la *gramática expandida* G^e de la gramática G original, añadiendo a cada regla de G sus correspondientes *reglas de error*. Existirá una regla de error asociada a cada posible transformación de error (figura 2.5).

Concretamente, las reglas de error a añadir a una gramática **regular** G=(N,V,P,S) para construir su gramática expandida serán, $\forall x \in V$, (ε es el símbolo nulo *nil*):

Inserción (de x): $A \rightarrow xA \quad \forall (A \rightarrow bB) \in P$

Sustitución (de b por x): $A \rightarrow xB \quad \forall (A \rightarrow bB) \in P$; $A \rightarrow x \quad \forall (A \rightarrow b) \in P$

Borrado (de b)[1]: $A \rightarrow B \quad \forall (A \rightarrow bB) \in P$; $A \rightarrow \varepsilon \quad \forall (A \rightarrow b) \in P$

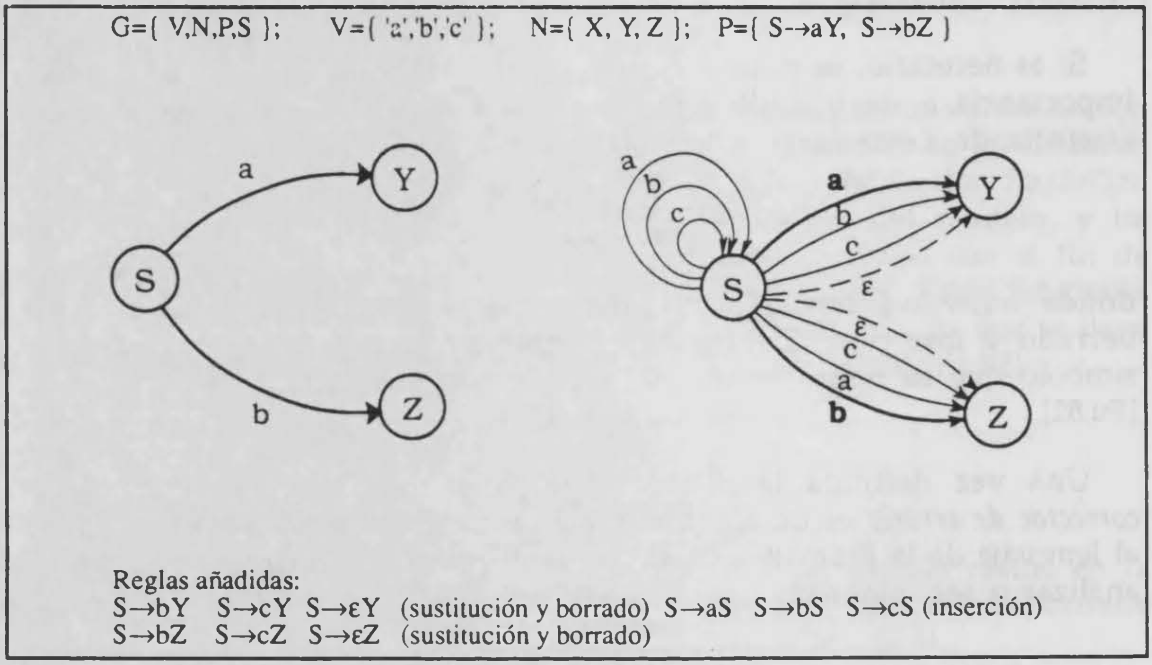


Figura 2.5 Autómata expandido: transiciones que se añaden al hacer la expansión de dos transiciones procedentes del mismo estado. Obsérvese la redundancia de transiciones de inserción.

- Analizar sintácticamente la cadena desconocida con G^e y luego, de entre todas las derivaciones obtenidas (G^e es siempre ambigua), buscar aquella cuyas reglas (transformaciones) de error produzcan la distancia mínima. Esta derivación[2] proporciona no sólo la cadena que más se asemeja a la muestra, sino también la distancia a la misma.

  Típicamente, los algoritmos que se emplean para esta búsqueda y minimización son los mismos que los utilizados para gramáticas no deterministas, por lo que se basan también en las técnicas de *programación dinámica*. Sin embargo, la particular forma de las

---

[1] Según la definición más corriente, estas reglas NO SON REGULARES. Esto se discute con detalle en el Capítulo 5.

[2] La derivación óptima puede no ser única.

reglas de borrado obliga a utilizar algoritmos especiales, algunos de los cuales se proponen en el capítulo 5.

2.4.2 Gramáticas estocásticas

Una gramática estocástica se forma añadiendo probabilidades a las reglas de una gramática no estocástica (la *gramática característica* de la gramática estocástica). El *lenguaje estocástico* generado por tal gramática está formado no sólo por las cadenas del lenguaje, sino también por su probabilidad de producción, que se obtiene mediante el producto de las probabilidades de las reglas utilizadas.

La teoría de las gramáticas estocásticas está perfectamente asentada, con una teoría matemática que complementa a la de los lenguajes formales. Por ejemplo, una gramática estocástica regular es equivalente (o se puede modelizar como) un *generador de Markov* [Casacuberta,90b], el cual destaca por ser uno de los procesos estocásticos más tratables y mejor comprendidos [Rabiner,89a].

2.4.2.1 Definiciones

Una gramática *ponderada* es una quintupla $G_s=(N,V,P,S,D)$. Donde $G=(N,V,P,S)$ es la gramática característica de la gramática ponderada y D es el conjunto de pesos asociado a las reglas de P. Para cada regla $\zeta_{1i}A_i\zeta_{2i}\rightarrow\zeta_{ij}$ el peso se escribe p_{ij} ($i=1,\ldots,K$; $j=1,\ldots,n_i$; K es el número de partes izquierdas distintas y n_i el número de reglas con la parte izquierda i). Se dice que G_s es *estocástica* si los pesos se comportan como probabilidades, es decir, si $p_{ij}\subseteq]0,1]$ y la suma de las probabilidades de todas las reglas con la misma parte izquierda es igual a la unidad:

$$\forall\ \zeta_{1i}A_i\zeta_{2i}\in W^*NW^*:\ \zeta_{1i}A_i\zeta_{2i}\rightarrow\zeta_{ij},\ j=1,\ldots,n_i,\ \sum_{i=1}^{ni} p_{ij}=1$$

La tipología 0,1,2,3 así como otros conceptos que dependen de las formas de las producciones (forma normal de Greinbach, etc...) se extienden por similitud con las no estocásticas.

Una gramática estocástica es *no-restringida* si la probabilidad de una regla no depende de la secuencia de reglas anteriormente aplicada. En una gramática no-restringida (y no ambigua), la probabilidad de generación de una cadena $\alpha=a_1a_2\ldots a_m$; $a_1,a_2,\ldots,a_m\in V$, a partir del axioma S, aplicando la secuencia de reglas $r_1,r_2,r_3,\ldots,r_m$, viene dada por $p(\alpha)=p(r_1)\cdot p(r_2)\cdot\ldots\cdot p(r_m)$, donde $p(r_i)$ es la probabilidad de la regla r_i. Escribiremos esto $S \overset{*}{\underset{p(\alpha)}{\Rightarrow}} \alpha$.

En una gramática ambigua, existirán n_α secuencias de reglas (derivaciones) para generar la misma cadena α, cada una con su respectiva probabilidad $p_i(\alpha)$, $i=1,...,n_\alpha$. En este caso la *probabilidad de generación de* α se puede definir de dos posibles maneras:

- suma de la probabilidad de un número finito de eventos excluyentes (*aproximación aditiva*):

$$p(\alpha) = \sum_{i=1}^{n_\alpha} p_i(\alpha)$$

- máxima verosimilitud (*aproximación maximal*):

$$p(\alpha) = \max_{i=1,..,n_\alpha} p_i(\alpha)$$

Una gramática estocástica G_s genera un *lenguaje ponderado* , es decir, un lenguaje en el que cada cadena lleva asociada un peso $p(\alpha)$ y que se define como (utilizando la aproximación aditiva):

$$L(G_s) = \{ [\alpha,p(\alpha)] \mid \alpha \in V^*,\ S \overset{*}{\underset{p_i(\alpha)}{\Rightarrow}} \alpha,\ i=1,...,n_\alpha,\ p(\alpha) = \sum_{1=i}^{n_\alpha} p_i(\alpha) \}$$

si la gramática es ambigua. Si no lo es:

$$L(G_s)=\{ [\alpha,p(\alpha)] \mid \alpha \in V^*, S \overset{*}{\underset{p(\alpha)}{\Rightarrow}} \alpha \}$$

Para poder considerar a $L(G_s)$ como un espacio muestral, en el que un suceso es la ocurrencia de una cadena con probabilidad dada por la gramática G_s, es necesario que la suma de las probabilidades de todas las cadenas que pueda generar la gramática sea igual a la unidad [Gonzalez,78]:

$$\sum_{x \in L(G_s)} p(\alpha)=1$$

A un lenguaje que cumple esta propiedad se le denomina un *lenguaje estocástico*, y a una gramática que genera un lenguaje estocástico es una gramática *consistente* :

$$G_s \text{ es consistente} \Leftrightarrow L(G_s) \text{ es estocástico}$$

Las condiciones de consistencia para gramáticas de contexto libre se pueden consultar en [Fu,82] [Wetherell,90]. Para que una gramática regular (de tipo 3) sea consistente basta con que sea *propia* (es decir, que no existan símbolos inútiles, ver [Fu,74] y [Miclet,86]).

2.4.2.2 Autómatas estocásticos

Del mismo modo en que se extienden las gramáticas para obtener gramáticas estocásticas, se pueden extender los reconocedores para obtener reconocedores estocásticos. En concreto, para las gramáticas estocásticas de tipo 3 o regulares, los reconocedores designados serán los *autómatas finitos estocásticos*, que se derivan de los autómatas finitos añadiendo probabilidades a las transiciones.

Un autómata finito estocástico (AFE) es una quíntupla $A_s(Q,V,\delta,I,F)$, donde Q es un conjunto finito de estados y V el conjunto de símbolos terminales. $\delta:Q\times E\times Q\rightarrow 0,1]$ es una función parcial que asigna las probabilidades a todas las posibles transiciones. $I:Q\rightarrow[0,1]$, $F:Q\rightarrow\{0,1\}$ definen para cada estado su probabilidad de ser inicial y su pertenecia o no al conjunto de estados finales, respectivamente. Para asegurar la estocasticidad, la suma de probabilidades de las transiciones que salen de un mismo estado debe ser igual a la unidad:

$$\forall q\in Q;\ \sum_{q_i\in Q,a\in V}\delta(q,a,q_i)=1 \quad \wedge \quad \sum_{q\in Q} I(q)=1$$

Se pueden dar toda una serie de definiciones similares a las formuladas para gramáticas estocásticas:

Si el AFE es *no-restringido* (la probabilidad de una transición es independiente de las de las demás transiciones) la probabilidad de aceptar una cadena $\alpha=\alpha_1\alpha_2...\alpha_n$ siguiendo la secuencia de estados $q_o,q_1,...,q_n$ se escribe como:

$$I(q_o)\cdot\delta(q_o,\alpha_1,q_1)\cdot...\cdot\delta(q_{n-1},\alpha_n,\alpha_n)\cdot F(q_n)$$

Dado que para una determinada cadena α pueden haber varios caminos a través del autómata que lleguen al estado final, cada uno con su probabilidad $p_c(\alpha)$, la probabilidad de generación de una cadena se puede definir también mediante la aproximacíon aditiva o la maximal:

$$p(\alpha)=\sum_{\forall c} p_c(\alpha)\ \textit{(aditiva)} \quad \text{o bien} \quad p(\alpha)=\max_{\forall c}\{\,p_c(\alpha)\,\}\ \textit{(maximal)}$$

Al igual que en las gramáticas, el *lenguaje aceptado por un AFE* está formado por las cadenas que a partir del estado inicial llevan a un estado final, **junto** con su probabilidad de aceptación. Se dice que dos AFE son *equivalentes* si aceptan el mismo lenguaje ponderado (mismas cadenas y mismas probabilidades).

Dada una gramática estocástica regular, existe un AFE que acepta un lenguaje idéntico al generado por la gramática (pero lo contrario no es cierto) [González,78] (figura 2.6).

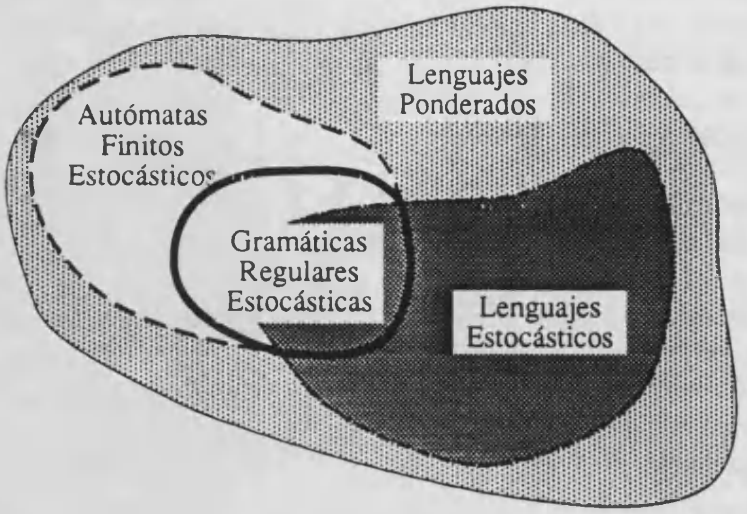


Figura 2.6 Relación de inclusión entre los lenguajes ponderados (P), los estocásticos (E), los generados por gramáticas regulares estocásticas (GRE) y los autómatas finitos estocásticos (AFE).

Como se recordará, en el caso general de los autómatas finitos no deterministas, al llevar a cabo el análisis sintáctico es necesario utilizar métodos que permitan evaluar en paralelo varios caminos en el autómata. Cuando se trata de autómatas estocásticos ello se complica debido a la presencia de las probabilidades, y si bien el *algoritmo de Viterbi* sigue siendo válido cuando se trabaja con la aproximación maximal, es necesario emplear el algoritmo *"Forward"* para el caso de la aproximación aditiva [Rabiner,89]. Ambos algoritmos son de complejidad similar (ver capítulo 4).

2.4.2.3 Aprendizaje de gramáticas estocásticas

En el aprendizaje de gramáticas estocásticas, no sólo se debe de tener en cuenta la estructura de los ejemplos, sino también la frecuencia de aparición de los mismos. Para ello, el método generalmente empleado se basa en la adquisición independiente del modelo estructural (de la gramática no estocástica), añadiéndose con posterioridad las probabilidades de las reglas mediante un análisis estadístico de los ejemplos y de sus derivaciones por la gramática [Maryanski,77] [Chandhuri,86]. Más raramente, es el propio proceso de inferencia de las reglas (estructura) el que se ve guiado por el comportamiento estadístico de los ejemplos [Thomason,86].

Consideraremos aquí únicamente el proceso de inferir las probabilidades de las reglas de una gramática característica G=(V,N,P,S) ya conocida. Dicho de otro modo, se trata de inferir el conjunto de probabilidades D de la gramática estocástica G_s correspondiente a G, a partir de un conjunto de m

ejemplos $X=\{\alpha_1,...,\alpha_m\}$ con frecuencias respectivas de aparición $f_1,...,f_m$. Entonces, y siempre que G no sea ambigüa, la probabilidad por máxima verosimilitud p_{ij} de la regla $\zeta_{1i}A_i\zeta_{2i}\rightarrow\zeta_{ij}$ se puede estimar en base a:

$$\hat{p}_{ij} = \frac{n_{ij}}{\sum_{\forall j} n_{ij}}; \qquad n_{ij} = \sum_{\alpha_k \in X} f_k \cdot N_{ij}(\alpha_k)$$

donde $N_{ij}(\alpha_k)$ representa el número de veces que la regla se utiliza para analizar α_k, y por lo tanto n_{ij} simboliza la frecuencia de utilización de la regla por parte de todas las cadenas de X. Para gramáticas independientes del contexto, se puede demostrar que esta estimación $\hat{p}_{ij}$ se aproxima a p_{ij} cuando el número m de ejemplos tiende a infinito, siempre que simultánemente X se aproxime a L(G) y f_k en X se aproxime a $p(\alpha_k)$ [Maryanski,77]. Esta estimación además garantiza que la gramática sea consistente [Fu,74].

Obsérvese que el procedimiento de estimación descrito no es válido en el caso de gramáticas ambiguas, pues no se ha definido ningún método para "distribuir" las probabilidades cuando hay más de una derivación. En el caso concreto de las gramáticas estocásticas ambigüas regulares existen métodos adecuados, entre los que cabe destacar, por su uso generalizado, el algoritmo de «Baum-Welch» y el de reestimación por Viterbi [Levinson,83] [Rabiner,89].

2.4.3 Análisis sintáctico corrector de errores estocástico

A la hora de efectuar el análisis sintáctico de cadenas muestras deformadas, es posible combinar en un solo proceso las dos estrategias presentadas en los apartados anteriores, obteniéndose entonces la metodología conocida como el*análisis sintáctico corrector de errores estocástico*. La idea de este método consiste simplemente en aplicar a las gramáticas estocásticas el mismo procedimiento de corrección de errores que se utilizó para las gramáticas no estocásticas; para lo cual, a una gramática estocástica se le añadirá el *modelo de corrección de errores* (las reglas de error) [Thomason,75]. Como la gramática expandida resultante deberá ser estocástica también, es necesario complementar cada transformación de error, no sólo con un peso como en la distancia de Levenshtein y la de Fu, sino con una *probabilidad*. En estas condiciones, la similitud entre cadenas no se mide mediante una distancia entre ellas, sino por **la probabilidad de que una se transforme en otra**. El análisis sintáctico corrector de errores no minimizará una distancia, sino que maximizará una probabilidad.

Utilizando un modelo de error similar[3] al utilizado como ejemplo en el apartado 2.4.1, las transformaciones de error posibles serán: inserción de un símbolo 'b' (**antes de otro** 'a', a,b∈ V), borrado del mismo (substitución de a por ε, el símbolo nil), y sustitución de 'a' por otro símbolo 'b'. A cada uno de estos errores se le asocia una probabilidad, respectivamente $p_i(a \mid b)$, $p_b(\varepsilon \mid a)$ y $p_s(b \mid a)$. Denotaremos $p_s(a \mid a)$ a la probabilidad de sustituir 'a' por sí mismo, es decir, la probabilidad de que NO se produzca error en 'a' (*probabilidad de no error*). Nos referiremos a una cualquiera de estas*probabilidades de deformación*, como **pd**. A partir de estas definiciones, y por extensión, pueden definirse la probabilidad de transformar un símbolo en una cadena y, aún con más generalidad, la de una cadena α en otra cadena β: $p(\beta \mid \alpha)$ [Fu,82].

Un analizador sintáctico corrector de errores estocástico buscará la cadena β del lenguaje de la gramática G que maximize a la vez la probabilidad de que la cadena α (que no pertenece a dicho lenguaje) se pueda transformar en β y la probabilidad de que β sea del lenguaje:

$$\beta = \underset{\forall \omega \in L(G^e)}{\operatorname{argmax}} \{p(\alpha \mid \omega) \cdot p(\omega)\}$$

donde el que ω pertenezca al lenguaje de la gramática expandida equivale a decir que se prueban todas las deformaciones permitidas de α. Los algoritmos utilizados para construir estos analizadores son similares a los utilizados en los autómatas analizadores sintácticos correctores de errores no estocásticos.

La probabilidad que maximiza la expresión anterior:

$$p^e(\alpha) = \max_{\forall \omega \in L(G^e)} \{p(\alpha \mid \omega) \cdot p(\omega)\}$$

se puede considerar como la *probabilidad de que α sea una cadena deformada del lenguaje de G*, o dicho de otro modo, la probabilidad de que α pertenezca a $L(G^e)$. También en este caso es necesario asegurar la consistencia de la gramática, es decir, asegurarse de que su lenguaje es estocástico:

$$\sum_{\forall \alpha \in L(G^e)} p^e(\alpha) = 1$$

Con el modelo de error que se está utilizando, es posible demostrar que ello será cierto siempre que las *probabilidades de deformación sean consistentes* [Fu,82], es decir:

---

[3] En este caso, el coste de inserción depende del símbolo antes del cual se realiza la inserción.

$$\sum_{\forall b \in V} p_s(b \mid a) + p_b(\varepsilon \mid a) + \sum_{\forall b \in V} p_i(b \mid a) = 1;\ \forall a \in V$$

En el caso de una gramática **regular**, en que se utilize un modelo de error en el que la probabilidad de inserción **no** depende del símbolo siguiente (ejemplo del apartado 2.4.1), la condición de consistencia es diferente y se estudia con detalle en el capítulo 7.

2.5 Gramáticas SANSAT

En los restantes capítulos de este trabajo se tratará exclusivamente con gramáticas regulares (y en su mayoría estocásticas). En particular, en las implementaciones prácticas se ha recurrido a una subclase de las gramáticas de tipo 3: las gramáticas SANSAT ("**SA**me **N**on-terminal, then **SA**me Terminal"), de las cuales el autor no ha encontrado en la literatura ninguna mención explícita. Una gramática G=(N,V,P,S) será una SANSAT, si cumple la condición de que todas las reglas que tienen el mismo **no** terminal a la derecha tienen también el mismo terminal:

$$\text{si } (B \rightarrow aC) \in P \wedge (A \rightarrow bC) \in P \Rightarrow a=c;\ \forall A,B,C \in N,\ \forall a,b \in V$$

Tal como se verifica a continuación, toda gramática regular tiene una gramática SANSAT equivalente, y un *autómata de estados etiquetados* (o autómata LAS: "**LA**belled **S**tate automata") igualmente equivalente.

2.5.1 Gramática SANSAT equivalente a una gramática.

Dada una gramática regular G=(N,V,P,S), una gramática SANSAT equivalente a ella, $G_t=(N_t,V,P_t,S)$, se construye de la siguiente manera (ver figura 2.7):

- Se genera un no terminal por cada parte derecha de regla diferente:

$$N_t=\{\ X_{aA} :\ \text{si } (B \rightarrow aA) \in P,\ a \in V,\ A,B \in N\ \}$$

- Para cada no terminal X_{aA}, y cada no terminal X_{bB} que corresponde a una regla r de G que tiene A como parte izquierda r=(A→bB), se genera una regla que reescribe X_{aA} como X_{bB} precedido del terminal de la regla r, es decir $X_{aA} \rightarrow bX_{bB}$:

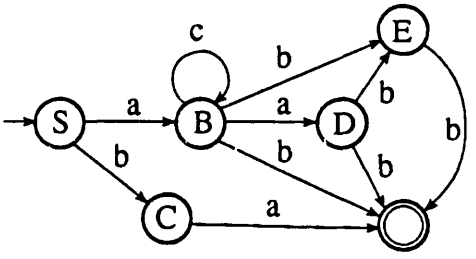
$$P_t=\{\ X_{aA} \rightarrow bX_{bB} : \text{si } (A \rightarrow bB) \in P\ \} \cup \{\ X_{aA} \rightarrow b : \text{si } (A \rightarrow b) \in P\ \} \cup \{\ S \rightarrow bX_{bB} : \text{si } (S \rightarrow bB) \in P\ \}$$

G=(N,V,P,S) N={B,C,D,E} V={ a,b,c}

L(G)={ ac*bb, abb, ac*abb,
aabb, ac*ab, aab,
ac*b, ab, ba
}

P = { S → aB, S → bC,
B → aD, B → bE,
B → b, B → cB,
D → bE, D → b,
E → b, C → a,
}

Autómata



Gramática SANSAT equivalente

Nt = { (aB),(bC),(cB),(bE),(aD) }

Pt = { S → a(aB), S → b(bC),
(aB) → c(cB), (aB) → b(bE),
(aB) → a(aD), (aB) → b,
(cB) → c(cB), (cB) → b(bE),
(cB) → a(aD), (cB) → b,
(aD) → b(bE), (aD) → b,
(bE) → b, (bC) → a
}

Autómata de Estados Etiquetados (LAS)

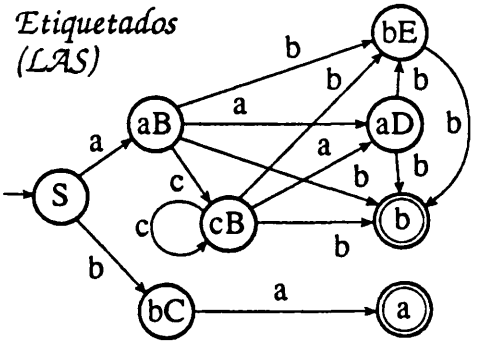


Figura 2.7 Una gramática regular G y su autómata equivalente. La gramática SANSAT equivalente a G, y su LAS (autómata de estados etiquetados) correspondiente (no se muestran las etiquetas de los estados).

Es inmediato comprobar que ambas gramáticas son equivalentes, o lo que es lo mismo que $L(G)=L(G_t)$. Para ello, basta tener en cuenta que toda derivación por G de una cadena $a_1a_2,...a_n \in L(G)$ se puede reescribir mediante reglas de G_t (y por lo tanto pertenece al lenguaje de ésta) y viceversa. En efecto:

$$D=r_1r_2,...r_n=$$
$$(S\rightarrow a_1A_1)(A_1\rightarrow a_2A_2)...(A_{n-1}\rightarrow a_n)=$$
$$(S\rightarrow a_1X_{a1A1})(X_{a1A1}\rightarrow a_2X_{a2A2})...(a_{n-1}X_{an-1An-1}\rightarrow a_n)$$

2.5.2 Autómata de estados etiquetados (LAS)

Un *autómata de estados etiquetados* LAS=(V_t,Q,δ,q_o,F,E), es un autómata equivalente a una gramática SANSAT G=(N_t,V,P_t,S), construído de manera similar a la seguida usualmente para obtener el autómata equivalente a una gramática, pero:

- Se añade una función de etiquetado E:Q→V que asigna a cada estado el símbolo terminal asociado a todas las reglas con ese no terminal a la derecha. Por completitud de E, se etiqueta arbitrariamente el estado inicial con el símbolo "~", (símbolo inicial) que se añade al conjunto de terminales.

- Se obliga a que haya tantos estados finales como símbolos terminales puedan terminar una cadena del lenguaje.

Todo lo cual queda expresado formalmente como sigue:

$V_t=V \cup \{\sim\}; \quad \sim \notin V$

$Q=\{ q_{aA}: \forall X_{aA} \in N_t \} \cup F; \; F=\{ q_a : \forall a \in V, (X_{bA} \rightarrow a) \in P_t \}; \quad q_o=S;$

$\delta: Q \times V \rightarrow Q;$

$\delta=\{ (q_{aA}, b, q_{bB}) : \text{si } (X_{aA} \rightarrow bX_{bB}) \in P_t \} \cup \{ (q_o, a, q_{aA}) : \text{si } (S \rightarrow aA) \in P \};$

$E: Q \rightarrow V; \; E(q_o)=\sim; \; E(q_{aB})=a; \; E(q_a)=a;$

En un LAS, los terminales asociados a las transiciones son **siempre iguales a la etiqueta el estado destino de la transición**. Esto permitiría definir las transiciones sin etiquetas (terminales asociados), al ser éstas redundantes, pero se las ha conservado para permitir que un LAS siga siendo un autómata en el sentido clásico. Por otra parte, gracias a esta propiedad, un LAS puede ser representado esquemáticamente de idéntica manera que un autómata clásico, pero etiquetando los estados en lugar de las transiciones (ver figura 2.8). También aprovechando esta propiedad es posible dar una definición alternativa del lenguaje del LAS: *una cadena es reconocida por un LAS* (y por lo tanto pertenece a su lenguaje) si existe un camino en el LAS que partiendo del estado inicial llegue a uno final, tal que la secuencia de **etiquetas de estados** se corresponda con la secuencia de símbolos de la cadena:

$$a_1a_2 \ldots a_n \text{ es reconocida por LAS} \Leftrightarrow$$
$$\exists \; q_o, q_1, \ldots q_n \mid E(q_i)=a_i; \quad i=1..n; \quad q_i \in Q; \quad q_n \in F$$

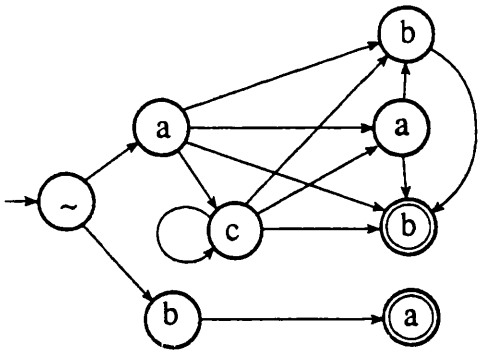


Figura 2.8 LAS de la figura anterior, representado con etiquetas en los estados en vez de en las transiciones.

Una gramática SANSAT $G_t=(N_t,V,P_t,S)$ (y por lo tanto su LAS) tiene generalmente más estados que su equivalente convencional $G=(N,V,P,S)$. Del procedimiento de construcción de la gramática SANSAT se observa que puede llegar a tener $N_t=|N|\cdot|V|$ no terminales y $P_t=|P|\cdot|V|$ reglas[4] (ver figura 2.9). De ello se deduce que la complejidad espacial de la representación de un lenguaje mediante una gramática SANSAT (o su LAS) puede llegar a ser $|V|$ veces mayor que su equivalente convencional, lo cual es especialmente significativo si el número de terminales es muy superior al de no terminales.

$P = \{ S \rightarrow aA,\ S \rightarrow bA,\ S \rightarrow cA,$
$A \rightarrow aA,\ A \rightarrow bA,\ A \rightarrow cA,$
$A \rightarrow a,\ A \rightarrow b,\ A \rightarrow c \}$

$N = \{ A \}$ $V = \{ a,b,c \}$

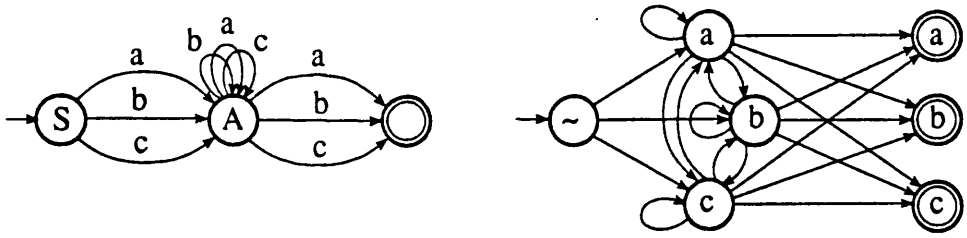


Figura 2.9 Caso extremo de incremento de transiciones y estados al pasar de un autómata a un autómata de estados etiquetados (LAS).

2.5.3 LAS y Modelos Ocultos de Markov (HMM)

Es extremadamente interesante notar cómo, a partir de la equivalencia entre una gramática regular y un autómata de símbolos etiquetados, es inmediato relacionar las gramáticas con los*Modelos Ocultos de Markov* (Hidden Markov Model: HMM [Rabiner,89a]) con estados finales. En efecto,

[4] Basta notar que, en realidad, lo que se está haciendo es desdoblar los no terminales (estados) en tantos como sea necesario para que a cada uno de ellos llegue siempre el mismo terminal.

si se define un *autómata estocástico de estados etiquetados*, como un LAS, con:

- Probabilidades de transición $p(q \mid q')$ (la probabilidad de una transición no depende del símbolo).
- Una función de etiquetado $E(q,a):Q \times V \rightarrow [0..1]$ probabilística (probabilidad de que la etiqueta de $q \in Q$ sea $a \in V$).

de forma que:

$$\sum_{\forall q} p(q \mid q')=1 \text{ y } \sum_{\forall a} E(q \mid a)=1$$

se tiene una definición equivalente a la de un Modelo Oculto de Markov con estados finales, en el que la distribución inicial de probabilidad de los estados $P_0(q_0)$, $P_0(q_1)..P_0(q_{|Q|})$ es [1,0,..,0] (sólo hay un estado inicial).

3

Inferencia Gramatical

3.1 Introducción

La utilización en la práctica de los métodos sintácticos de reconocimiento de formas viene condicionada, no sólo por la necesidad de tener resuelta la etapa de *representación*, que en la suposición de que los objetos se presten a una descripción en términos de subobjetos, se resume en disponer de un método satisfactorio de selección y/o extracción de los mismos, sino también por la obligatoriedad de conocer la descripcción estructural de todos los posibles objetos que toman parte de la forma. Es decir, es necesario conocer la *gramática*. Como generalmente ello no es posible a priori, se hace necesario construirla u obtenerla mediante algún método de *aprendizaje*; y dado que en la mayoría de los problemas de reconocimiento de formas la única información de que se dispone (o se es capaz de transmitir) sobre la forma se halla resumida en un conjunto de *ejemplos*, se recurre usualmente a métodos de *aprendizaje inductivo*, que en el caso de los métodos sintácticos se engloban en lo que se conoce como *inferencia gramatical* (IG) [Fu,75] [Miclet,90].

Al igual que cualquier otro problema de inferencia Inductiva, la inferencia gramatical se especifica mediante la definición de [García,88]:

a) Un *dominio* de formas a inferir.
b) Un *espacio de hipótesis* o representaciones.
c) Un *método de presentación* de ejemplos.
d) Un *método de inferencia*.
e) Un *criterio de éxito*.

Usualmente las aplicaciones prácticas descartan, por su intratabilidad, la utilización de los métodos *enumerativos* de inferencia gramatical, centrándose en métodos *constructivos*, los cuales utilizan normalmente sólo muestras *positivas*. La gran mayoría de los métodos de inferencia

gramatical existentes en la actualidad son métodos*heurísticos* (no *caracterizables*) e infieren gramáticas **regulares**.

3.2 Especificación del problema de IG

3.2.1 Dominio de formas a inferir

Para la inferencia gramatical el dominio de formas es cualquier subconjunto de los lenguajes formales. Más concretamente, se le restringe a cualquier subconjunto de los *lenguajes recursivos*. (los lenguajes recursivos son aquellos lenguajes formales $L \subset V^*$ sobre el alfabeto V, para los cuales es decidible si una cadena $\alpha \in V^*$ pertenece o no a dicho lenguaje).

3.2.2 Espacio de hipótesis o representaciones

El espacio de las hipótesis depende del dominio de formas que debe de inferir el sistema y de la representación utilizada. Como única condición, debe de estar compuesto de por lo menos una *representación* (descripción de una hipótesis) para cada forma (en nuestro caso, lenguaje) del dominio. En particular, en el caso de que se trate de inferir un lenguaje de la subclase de los lenguajes regulares sobre un alfabeto V, el espacio de las hipótesis lo forman las gramáticas regulares sobre V, aunque también lo podrían serlo los autómatas finitos sobre V.

3.2.3 Método de presentación de ejemplos

En general en inferencia inductiva se utilizan dos métodos de presentación de ejemplos [Gold,67]:

- Presentación *positiva* del lenguaje L: es una sucesión de elementos de L (muestras positivas).
- Presentación *completa* del lenguaje L: es una sucesión de elementos de L y de su complementario (muestras positivas y muestras negativas), marcados para indicar su pertenencia o no a L. Todas las cadenas de V^* aparecen en la secuencia.

Ambos métodos son utilizados en inferencia gramatical, aunque básicamente se emplee la presentación positiva por las razones que se expondrán más adelante.

3.2.4 Método de inferencia

Que consiste en un algoritmo que a cada nuevo ejemplo proporciona una forma (su representación, la hipótesis) válida (o no) para los ejemplos presentados hasta ese momento. Expresado formalmente, un *método de inferencia* $M_{\pounds,\Gamma}$ es una función $M_{\pounds,\Gamma}:2^{\Theta}\rightarrow\mathcal{H}$, donde 2^{Θ} es el conjunto de todos los subconjuntos finitos del espacio de objetos Θ (de cadenas de V^* para la IG), $\pounds$ el dominio y $\mathcal{H}$ el espacio de las hipótesis. Γ es el *criterio de preferencia* que utiliza el método para escoger la siguiente hipótesis $H_t\in\mathcal{H}$, válida para el conjunto de ejemplos $\{\alpha_1,...,\alpha_t\}$, cuando le llega el ejemplo α_t en el instante t (ver figura 3.1).

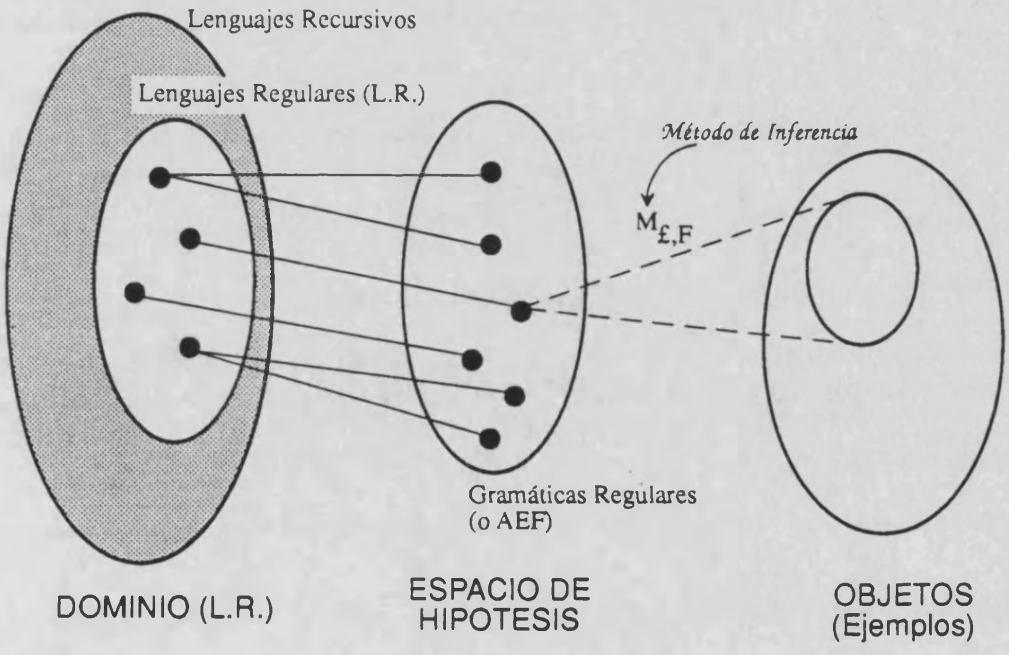


Figura 3.1 Dominio y Espacio de las Hipótesis en Inferencia Gramatical de Lenguajes Regulares.

Es deseable que un método de inferencia sea **Consistente**, es decir, que acepte todos los ejemplos positivos leídos hasta ese momento y rechace los negativos si los hay; y **Conservativo**, es decir, que sólo cambia de hipótesis si un nuevo ejemplo es incompatible con la hipótesis presente.

Existen dos tipos básicos de métodos de inferencia, los:

- *Constructivos*, que van construyendo una nueva hipótesis con cada nuevo ejemplo.
- *Enumerativos*, que asumen que es posible enumerar las hipótesis. A cada nuevo ejemplo, un método enumerativo buscará secuencial y exahustivamente en la lista de hipótesis la primera que sea compatible con todos los ejemplos presentados.

En inferencia gramatical, como en casi todos los casos prácticos de reconocimiento de formas, se emplean métodos constructivos que usualmente son conservativos y consistentes.

3.2.5 Criterio de éxito

Si un proceso de inferencia se considera infinito, se puede determinar su éxito estudiando su comportamiento en el límite. El criterio de éxito más fuerte que se puede exigir es que el método dé con la solución correcta, es decir, que a partir de cierto momento (de cierto ejemplo) no cambie de hipótesis y que la hipótesis que tiene en ese momento sea la buena (lo que no quiere decir que el método sepa detenerse en ese punto). Este es el criterio de *identificación en el límite.*:

$$\text{M identifica en el límite a L} \Leftrightarrow \exists t_0\ (t>t_0 \Rightarrow (H_t=H_{t_0})\ \&\ L(H_t)=L)$$

Desde luego, existen métodos menos exigentes (Concordancia en el límite, aproximación y aproximación fuerte, ε-identificación en el límite,...) [García,88].

Se dice que un conjunto de lenguajes recursivos es *identificable a partir de presentación completa* si para cualquier lenguaje L del conjunto y cualquier secuencia infinita de cadenas de L, existe un algoritmo que identifica a L en el límite.

3.3 Métodos enumerativos

Un método de inferencia es más **potente** que otro, si dado un criterio de éxito y un método de presentación, el conjunto de lenguajes que es capaz de inferir (su *alcance*) es más amplio. Puesto que los métodos enumerativos se basan en una búsqueda exahustiva del espacio de las hipótesis, es evidente que pueden inferir cualquier clase de lenguajes, y que por lo tanto son de una potencia insuperable. Toda limitación en la potencia de los métodos enumerativos será pues válida también a los otros métodos de inferencia, lo que permite limitar el estudio teórico a los enumerativos, más sencillos conceptualmente.

El **tiempo de convergencia** de un método de inferencia se define como el punto (número del ejemplo) a partir del cual se se ha conseguido la identificación en el límite. Un método es **uniformemente más rápido** que otro cuando, sea cual sea la presentación, su tiempo de convergencia es menor en por lo menos uno de los lenguajes a inferir (y no es mayor para ningún otro de estos lenguajes). Un teorema de Gold [Gold,67], establece

que, para un determinado lenguaje y una determinada presentación, *no existe un algoritmo uniformemente más rápido que el correspondiente método enumerativo.*

Desgraciadamente, es muy difícil implementar en la práctica un método enumerativo: la complejidad de la búsqueda exahustiva que implican crece exponencialmente con la talla del espacio de las hipótesis. A pesar de ello, algunas variantes de los métodos enumerativos han sido estudiadas. Estos algoritmos, que se basan en estructurar el espacio de las hipótesis con alguna relación más compleja que la simple enumeración, consiguen disminuir drásticamente la complejidad de la búsqueda en algunos casos concretos (Poda, Búsqueda en un retículo,...) [García,88].

3.4 Presentación positiva

Dos teoremas básicos, también debidos a Gold, delimitan drásticamente lo que posible aprender mediante cualquier método de inferencia [Gold,67]:

- Cualquier clase de lenguajes recursivos primitivos es identificable en el límite a partir de presentación completa.
- Si una clase de lenguajes recursivos contiene *todos* los lenguajes finitos y al menos uno infinito (se dice que es *superfinita*), entonces **no** es identificable mediante presentación positiva.

Estos teoremas no hacen más que asentar formalmente un razonamiento intuitivo elemental: si no se dispone de muestras negativas es imposible limitar la generalización efectuada por el método de inferencia: cualquier hipótesis cuyo lenguaje sea un superconjunto del buscado será compatible con los datos. Con todo, un corolario inmediato evidencia que *la clase de los lenguajes regulares no es identificable en el límite a partir de presentación positiva.*

A pesar de lo anterior, la mayoría de los sistemas de inferencia gramatical sólo utilizan muestras positivas, principalmente debido a que los métodos enumerativos no son lo suficientemente rápidos, y los métodos constructivos existentes no permiten la utilización de muestras negativas. Otra razón, de no menos peso en la práctica, reside en el hecho de que, prescindiendo de consideraciones teóricas, no es necesario identificar perfectamente un lenguaje para poder construir un sistema reconocedor que funcione. En un clasificador, por ejemplo, basta con que el lenguaje inferido contenga al de una clase y no se intersecte con los de las otras clases. Por otra parte, no es necesario empeñarse en inferir lenguajes superfinitos: existen clases de lenguajes identificables en el límite a partir de presentación positiva. Por ejemplo, cuando los lenguajes de una clase son todos finitos,

ésta es identificable con sólo ir construyendo el *autómata canónico* (si el lenguaje es finito, es regular) (figura 3,2).

L={ aabbab,
aabba,
abbbbaa,
abbaaaabb }

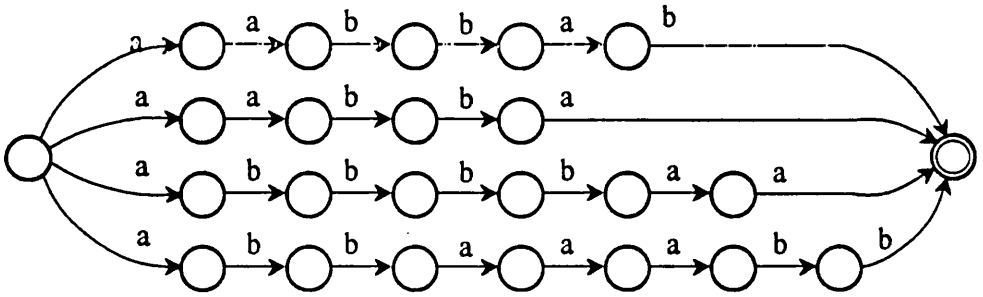


Figura 3.2 Autómata Canónico de un Lenguaje Finito L.

De hecho, es posible caracterizar los lenguajes inferibles a partir de presentación positiva. Para que una familia de lenguajes recursivos $\pounds=\{L_1,L_2,...\}$ sea identificable en el límite a partir de datos positivos, es *necesario y suficiente* el que exista un método que, para cualquier lenguaje $L_i \in \pounds$, permita enumerar un conjunto **finito** de frases $F_i \subset L_i$, tal que si $F_i \subseteq L_j$ para todo $j \geq 1$, L_j no es un subconjunto propio de L_i [Angluin,80].

3.5 Métodos Heurísticos y Métodos Caracterizables

Generalmente, en un proceso de inferencia a partir de ejemplos la solución no es única, debiéndose en la práctica escoger, de entre un conjunto de soluciones posibles, la solución que mejor se adapte a la aplicación en particular. Esta elección presupone la introducción en el proceso de inferencia de un conocimiento adicional, que se puede aportar de dos maneras, correspondientes a dos clases de métodos de inferencia [Angluin,83] [Garcia,90]:

- *Métodos heurísticos,* en los que el conocimiento está incluído en el heurístico que define el procedimiento de generación de la siguiente hipótesis. Lo ideal es que se emplee un heurístico lo más ceñido posible al campo de aplicación, para así evitar hipótesis inconsistentes con dicho campo. En este tipo de métodos no suele ser posible definir formalmente el dominio de lenguajes que infieren.

- *Métodos caracterizables*, en los que la hipótesis siguiente se escoge siempre dentro de una clase de lenguajes conocida y bien definida (que generalmente formará parte de los "inferibles mediante presentación positiva" según el teorema de Angluin). Aquí, el conocimiento del problema se introduce escogiendo la clase de lenguajes cuyas propiedades sean apropiadas para la aplicación concreta.

En la práctica, y dada la escasez existente de métodos caracterizables cuyos lenguajes sean utilizables en aplicaciones reales, los diseñadores acaban escogiendo algún método heurístico conocido, procediendo a adaptarlo a su problema concreto. A esto se añade la dificultad de que la mayoría de los métodos heurísticos disponibles actualmente se han diseñado de manera de aprovechar ciertas propiedades generales de los lenguajes regulares (lema de la estrella, equivalencia de los buenos finales,...), con lo que resultan poco aplicables en la práctica.

Por otro lado, recientemente se ha propuesto una nueva metodología que permite desarrollar nuevos métodos de inferencia gramatical, en los que el conocimiento a priori se incorpora con cierta facilidad en base a las propiedades requeridas para el lenguaje buscado [García,88].

3.6 Métodos constructivos de IG

Por razones como las expuestas en apartados anteriores, la gran mayoría, por no decir todos, los métodos prácticos de inferencia gramatical son constructivos y utilizan únicamente presentación positiva. A su vez, de entre todos ellos, la gran mayoría está orientada a la inferencia de lenguajes regulares, siendo escasos los que infieren gramáticas de contexto libre o superiores [García,88].

De entre los métodos de inferencia de gramáticas **regulares** (autómatas) caben destacar los que se basan en distintos métodos de agrupar los estados del autómata *árbol aceptor de prefijos* (ver figura 3.3) de las muestras positivas. Este autómata proporciona un espacio de búsqueda muy adecuado, siempre que la muestra sea *estructuralmente completa* (se ha utilizado para generarla todas las reglas de la gramática a inferir).

Ejemplos de este tipo serían (ver [García,88] para una breve explicación de cada uno): método de las k-colas, que se basa en la "equivalencia de los buenos finales" [Biermann,72]; algoritmo k-RI, que infiere lenguajes regulares de la subclase de los "k-reversibles" [Angluin,82]; inferencia de lenguajes k-contextuales [Muggleton,84], que produce resultados similares al anterior; y otros dos también basados en los "buenos finales": método de Levine [Levine,82] y método de comparación de finales [Miclet,80].

PREFIJOS de $L \subseteq V^*$: (· es la concatenación cadena-cadena/símbolo)
$Pr(L)=\{ u \in V^*: u \cdot v \in L, v \in V^* \}$

L={aaa,
aabb,
aba,
abb,
baaa,
bab,
aa }

Pr(L)={ a, aa, aaa,
aab, aabb,
ab, aba,
abb,
b, ba, baa, baaa,
bab }

ACEPTOR de PREFIJOS de L finito:

(V,Q,δ,q_0,F)
$Q=Pr(L)$;
$q_0=\lambda$ (cadena vacía)
$F=L$
$\delta(u,a)=u \cdot a$

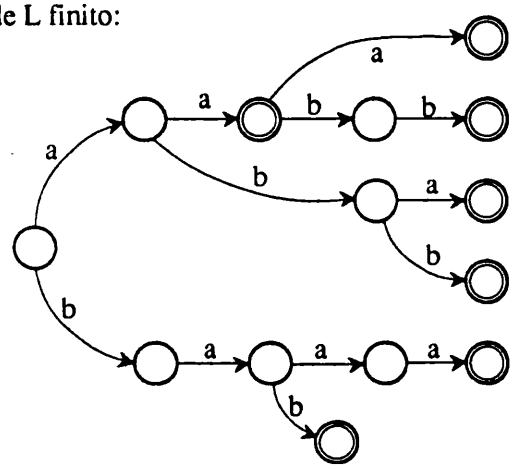


Figura 3.3 Arbol (autómata) aceptor de prefijos.

Otros métodos de inferencia de autómatas regulares, que no se basan en simplificar el árbol aceptor de prefijos, buscan estructuras repetitivas en los ejemplos (método uv^iw [Miclet,79]), o analizan la aparición de símbolos consecutivos (método del sucesor y método del antecesor-sucesor [Richetin,84]).

Finalmente, métodos como el presentado en este trabajo (ECGI) y otros expuestos en el capítulo 7 [Chirathamjaree,80] [Thomason,86] [Falaschi,90], utilizan la *corrección de errores* para construir incrementalmente gramáticas, añadiendo a cada nueva muestra únicamente las reglas necesarias para que sea aceptada.

4

Programación Dinámica

4.1 Introducción

Los problemas de análisis sintáctico, en gramáticas regulares no deterministas (en las cuales se centra este capítulo), implican generalmente una búsqueda no trivial en el espacio de todas las posibles derivaciones. Esto es aún más inevitable en tanto en cuanto, en la mayoría de los casos prácticos, es imposible transformar una gramática no determinista en su correspondiente determinista, dado el enorme coste espacial (número de no-terminales o estados) que usualmente ello implica. El problema es especialmente agudo en reconocimiento sintáctico de formas, pues normalmente es necesario emplear gramáticas correctoras de errores, las cuales son innatamente no deterministas, y/o gramáticas estocásticas, en las cuales el análisis sintáctico consiste de por sí en el análisis de todas las posibles derivaciones (puesto que, en general, todas las reglas tienen alguna probabilidad de ser usadas). Las técnicas utilizadas en la práctica para llevar a cabo esta búsqueda en el espacio de derivaciones, se agrupan dentro del conjunto de algoritmos y técnicas de *optimización* conocido como *Programación Dinámica* (PD).

Las técnicas de programación dinámica resultan aplicables dado que el problema de análisis sintáctico en gramáticas regulares cumple el *principio de optimalidad* de Bellman. Una vez comprobado esto, el problema es sencillamente resoluble, con complejidad polinómica si se utilizan las verisones *iterativas* de los algoritmos, sólo con plantear el análisis sintáctico como la búsqueda de un camino *óptimo* en un grafo *multi-etapa* (conocido como «celosía» o «retículo», o más usualmente por su nombre inglés: *trellis*).

4.2 Principio de optimalidad

En este contexto, "optimizar" equivale a seleccionar (buscar) la «mejor» solución de entre muchas posibles alternativas. Este proceso de optimización puede ser visto como una *secuencia de decisiones* que nos proporcionan la solución correcta. Si, dada una subsecuencia de decisiones, siempre se conoce cual es la decisión que debe tomarse a continuación para obtener la secuencia óptima, el problema es elemental y se resuelve trivialmente tomando una decisión detrás de otra, lo que se conoce como estrategia *voraz*.

A menudo, aunque no sea posible aplicar la estrategia voraz, se cumple el *principio de optimalidad de Bellman* [Bellman,57]: «dada una secuencia óptima de decisiones, toda subsecuencia de ella es, a su vez, óptima». En este caso sigue siendo posible el ir tomando decisiones elementales, en la confianza de que la combinación de ellas seguirá siendo óptima, pero será entonces necesario explorar muchas secuencias de decisiones para dar con la correcta, siendo aquí donde interviene la programación dinámica.

Contemplar un problema como una secuencia de decisiones equivale a dividirlo en subproblemas de talla inferior, en principio más fácilmente resolubles. Ello se enmarca en el método de *Divide y Vencerás* [Horowitz,78], técnica similar a la de Programación Dinámica. La programación dinámica se aplica cuando la subdivisión de un problema conduce a:

- Una enorme cantidad de subproblemas.
- Subproblemas cuyas soluciones parciales se solapan.
- Grupos de subproblemas de muy distinta complejidad.

circunstancias que en conjunto o por separado, llevarían a una complejidad *exponencial* de la estrategia "Divide y Vencerás".

4.3 Relación de recurencia, versión iterativa

La versión *recursiva* del método de programación dinámica se resume en el siguiene esquema:

```
Esquema PDR(D:δ):ρ
Auxiliar  contorno:δ→B      inicializa:δ→ρ   ⊗:ρ×ρ→ρ
          elemental:δ×δ→ρ   decide:Cρ→ρ
          descompone:δ→Cδ×δ
Método
si contorno(D) entonces devuelve inicializa(D)
sino
  devuelve      decide      {PDR(Z)⊗elemental(Z,z)}
                ∀(Z,z)∈descompone(D)
finsi
fin PDR
```

La idea del método consiste en subdividir el problema D en un conjunto de pares de subproblemas, uno trivial (resuelto mediante "elemental") y otro de complejidad «menor» que D, que a su vez se subdivide en pares de subproblemas y así sucesivamente. La recursión prosigue hasta alcanzar el problema de talla mínima (detectado por "contorno") cuya solución devuelve "inicializa". En cada paso, "decide" escoge, de entre todas las subdivisiones del problema proporcionadas por "descompone", aquella que nos lleva a la solución «óptima». ⊗ va combinando las soluciones elementales, proporcionando el resultado cuando se deshace la recursión.

Dos puntos deben resaltarse en este esquema:

- Puede ocurrir que se tenga que resolver varias veces el mismo subproblema, al presentarse éste en varias ramas distintas de la recursión, lo que en el caso general nos llevará a una complejidad exponencial.

- La solución obtenida es «óptima» sólo en el sentido indicado por "decide", cambiando esta función se pueden obtener resultados completamente distintos, cada uno óptimo a su manera.

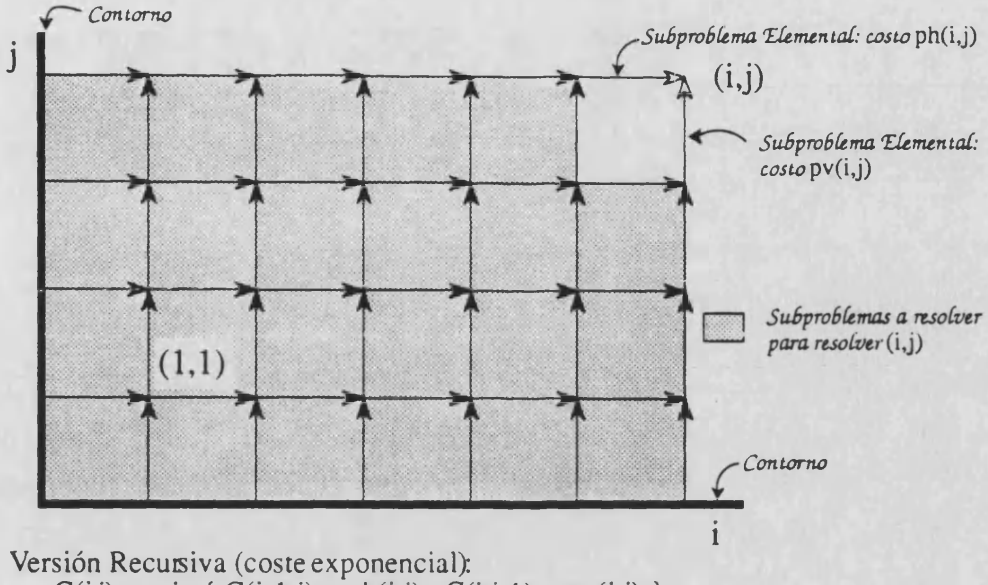
Con el fin de reducir la complejidad de exponencial a *polinómica*, evitando recalcular subproblemas ya calculados, se transforma este esquema recursivo en uno *iterativo*:

```
Esquema PDI(D:δ):ρ
Auxiliar  contorno:δ→B      inicializa:δ→ρ   ⊗:ρ×ρ→ρ
          elemental:δ×δ→ρ   decide:C_ρ→ρ
          descompone:δ→C_δ×δ  siguiente:δ→δ
          d_o:δ  /*Primer subproblema (según siguiente) no en contorno*/
Variables d:δ
          G:C_δ×ρ /*Almacén de resultados intermedios, indexado por d*/
Método
  ∀d∈δ si contorno(d) entonces G[d]:=inicializa(d)
  finV
  si ¬contorno(D) entonces
     d:=d_o
     bucle
        G[d]:=      decide      {G[Z]⊗elemental(Z,z)}
                ∀(Z,z)∈descompone(d)
        si d=D entonces salirbucle finsi
        d:=siguiente(d)
     finbucle
  finsi
  devuelve G[D]
fin PDI
```

La función "siguiente" ordena el espacio de subproblemas de tal manera que todo problema «menor» que uno dado se resuelve antes. Los problemas que no tienen subproblemas «menores» se resuelven al principio mediante "inicializa"; a partir de ellos los demás problemas se calculan siguiendo el orden definido y almacenando los resultados intermedios en G. De esta manera, cuando en cada punto se deba resolver un conjunto de subproblemas para que "decide" pueda escoger entre ellos, todos los subproblemas menores ya habrán sido resueltos y sus resultados se hallarán disponibles en G. La solución a cada subproblema se obtiene entonces evaluando un subproblema elemental y combinándolo mediante ⊗ con un resultado contenido en G.

En la figura 4.1 se muestra la resolución por PD (en su versión iterativa y recursiva) del problema de encontrar el camino de coste mínimo entre dos vértices de un grafo dirigido y ponderado en forma de cuadrícula.

Asumiendo costes unitarios para las operaciones elementales, la *complejidad temporal* de la versión iterativa de programación dinámica es O(k·m), donde k es el máximo número de subproblemas en los que se puede subdividir un subproblema y m es el número de subproblemas "diferentes" del problema a resolver:



Versión Recursiva (coste exponencial):

$C(i,j) = \min \{ C(i-1,j) + ph(i,j),\ C(i,j-1) + pv(i,j) \}$

Contorno: $C(0,j)=0;\ C(j,0)=0;\ \forall j$

Versión Iterativa (coste polinómico):

Misma operación, pero recorriendo la cuadrícula en orden creciente de (i,j)

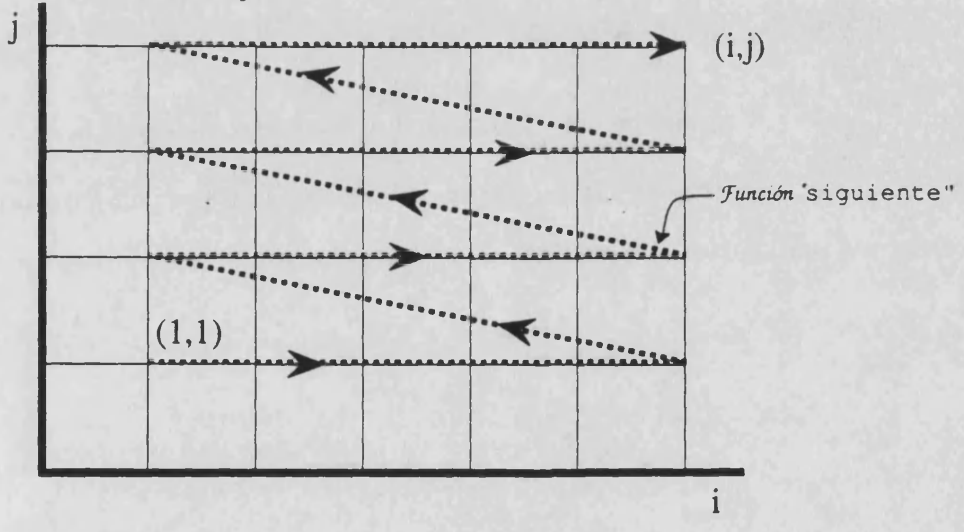


Figura 4.1 Versión Recursiva e Iterativa de un problema simple de PD (camino de coste mínimo en la Cuadrícula entre (1,1) e (i,j) en un grafo dirigido y ponderado) [Torró,89].

$$k=\max_{d\in\delta} |descompon(d)|; \qquad m=|\{d\in\delta: d<D \vee contorno(d)\}|$$

La *complejidad espacial*, debida en su totalidad a G, es obviamente O(m).

4.4 Programación Dinámica y autómatas

El análisis sintáctico en gramáticas regulares no deterministas se basa en un problema típicamente resoluble mediante programación dinámica: la búsqueda del camino de coste mínimo (o máximo) en un *grafo multietapa.*

4.4.1 Grafos multi-etapa

Se define un *grafo dirigido* G=($\mathcal{V}$,A) como un conjunto $\mathcal{V}$ de*vértices* (también llamados *nodos*) y un conjunto A={(u,v): u∈ $\mathcal{V}$, v∈ $\mathcal{V}$} de *arcos*. Si el grafo es *ponderado,* habrá además un *peso* (definido en el conjunto ρ) asociado a cada arco: p:A→ρ. Un *grafo multietapa* es un grafo en el que (figura 4.2):

- El conjunto de vértices está particionado en K *etapas*:

$$\mathcal{V}=\bigcup_{i=1..K}\mathcal{V}_i \qquad \mathcal{V}_i\cap\mathcal{V}_j=\emptyset;\quad \forall i,j=1,\ldots,K;\quad i\neq j;\quad k\geq 2$$

- Todos los arcos se producen entre etapas adyacentes y en el mismo sentido:

$$\forall (u,v)\in A \quad \exists i,\ 1\leq i\leq K-1 :\ u\in\mathcal{V}_i \wedge v\in\mathcal{V}_{i+1}$$

- Existe un conjunto de vértices iniciales $\mathcal{I}\subseteq\mathcal{V}_1$ y un conjunto de vértices finales $\mathcal{F}\subseteq\mathcal{V}_K$.

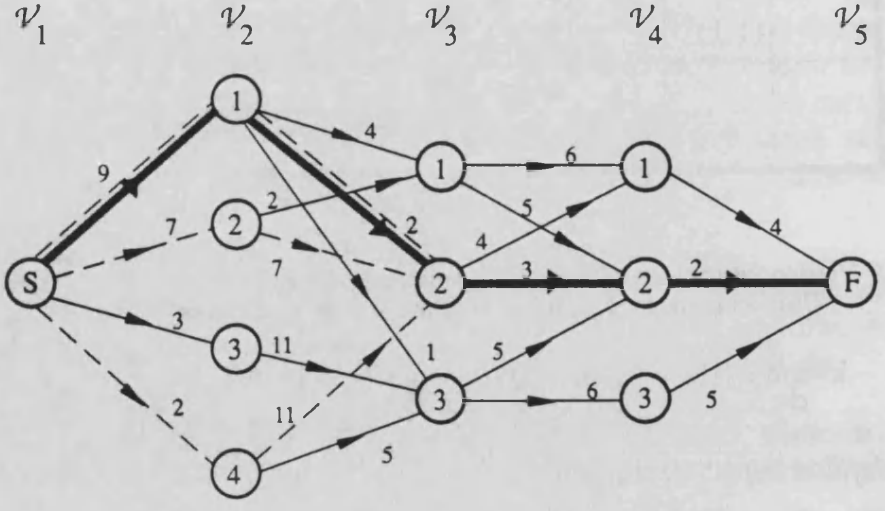


Figura 4.2 Un grafo multietapa ponderado de 5 etapas, un estado inicial y uno final. El trazo grueso indica el camino de mínimo coste [Horowitz,78]. Los arcos en trazo punteado representan el subgrafo del nodo (3,2).

Un *subgrafo* g(j,w) de un grafo multietapa dirigido G es un grafo que contiene todos los caminos en G que van desde cualquier vértice inicial

hasta el vértice w de la etapa $\mathcal{V}_j$. Se define $U_{j,w}$ como el conjunto de todos los vértices de una etapa $\mathcal{V}_{j-1}$ conectados al vértice w de la etapa $\mathcal{V}_j$:

$$U_{j,w} = \{ u \in \mathcal{V}_{j-1} : (u,w) \in A \} \quad 1<j\leq K; \quad w \in \mathcal{V}_j$$

4.4.2 Camino mínimo en un grafo multietapa

Se considera el problema de encontrar un camino que, yendo desde un vértice inicial a uno final de un grafo multietapa ponderado, extremice el coste acumulado C según cierto operador $\otimes$ (suma de pesos, producto de pesos, operación booleana...). Para este problema es posible obtener inmediatamente la relación de recurrencia que lo resuelve por programación dinámica:

$$C(j,w) = \underset{\forall u \in U_{j,w}}{\texttt{extremiza}} \; (C(j-1,u) \otimes \rho(u,w))$$

Aquí, la función "`decide`" es "`extremiza`", cada problema de encontrar el coste extremal C(j,w) en el subgrafo g(j,w) se descompone en los subproblemas C(j-1,u), asociados a los subgrafos g(j-1,u), $\forall u \in U_{j,w}$; y en los subproblemas elementales -arcos- de peso $\rho(u,w)$. El número de subproblemas es igual al número de subgrafos y por lo tanto al de vértices (sólo consideramos un único vértice inicial). El número de posibles decisiones en cada punto depende del número de arcos que llegan a un vértice. Si M es el número medio de arcos por vértice, la *complejidad media* espacial del algoritmo, en su versión iterativa, será $O(|\mathcal{V}| \cdot M)$, siendo $O(|\mathcal{V}|)$ la temporal.

4.4.3 Trellis

Para todo par (autómata finito, cadena de terminales), es posible asociarle un grafo multietapa ponderado, conocido con el nombre de *trellis* (palabra inglesa para "celosía"), de tal manera que el problema del análisis sintáctico de la cadena por el autómata se transforma en la búsqueda de un camino mínimo en dicho grafo.

Dado el autómata $A=(V,Q,\delta,q_0,F)$ y la cadena $\alpha=a_1,a_2,\ldots,a_n$, el trellis asociado $T(\mathcal{V},A)$ se define de la siguiente manera (figura 4.3):

- Cada etapa tiene tantos vértices como estados del autómata; hay una etapa por símbolo de la cadena, más una inicial:

$$\mathcal{V} = \bigcup_{i=0..n} \mathcal{V}_i \qquad \mathcal{V}_i = \{ (i,q) : q \in Q \}$$

- Hay un arco entre el vértice (j-1,w) de la etapa j-1 y el vértice (j,u) de la etapa j si existe una transición entre el estado u y el estado w del autómata:

$$A = \bigcup_{i=1..n} A_i$$

$$A_j = \{ ((j-1,w),(j,u)) : (j-1,w) \in \mathcal{V}_{j-1};\quad (j,u) \in \mathcal{V}_j;\quad u \in \bigcup_{\forall a \in V} \delta(w,a) \}$$

- Sólo hay un vértice inicial: el vértice de la primera etapa que corresponde al estado inicial del autómata: $I=\{(0,q_0)\}$.

- El conjunto de vértices finales está constituído por aquellos vértices de la última etapa que corresponden a estados finales del autómata: $\mathcal{F}=\{(n,q):q \in F\}$.

- Los pesos están definidos en el dominio de los booleanos, siendo "`verdad`" el peso del arco ((j-1,w),(j,u)) si el estado u pertenece a los sucesores de w cuando aparece el símbolo a_j de la cadena:

$$\rho((j-1,w),(j,u)) = \text{"verdad"} \Leftrightarrow u \in \delta(w,a_j)$$

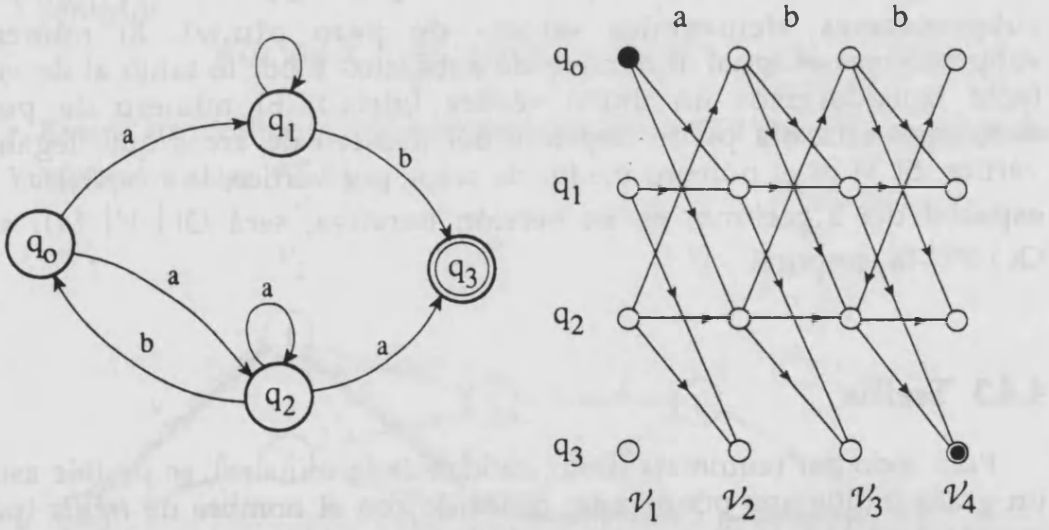


Figura 4.3 Un autómata finito y el trellis correspondiente para la cadena "abb" [Torró,89].

En la recursión de programación dinámica el operador ⊗ será el booleano ∧ ("o" lógico), y la función "`extremiza`" podrá ser simplemente el "y" lógico (∨), con lo que tendremos:

$$C(j,q) = \bigvee_{\forall u \in U_{j,q}} (C(j-1,u) \wedge \rho((j-1,u),(j,q)))$$

El costo C(j,q) será "verdad" únicamente cuando alguno de los caminos que lleguen al vértice (j,q) sea la composición de transiciones todas ellas con peso "verdad". En la versión iterativa, la complejidad temporal media de este algoritmo, es decir la complejidad de un análisis sintáctico en autómatas no deterministas, será $O(n\cdot|Q|\cdot B)$, ya que el número de vértices es $(n\cdot|Q|)$ y el número medio de arcos por vértice es $B=med|U_{j,q}|$. La complejidad espacial es $O(n\cdot|Q|)$.

4.5 El algoritmo de Viterbi

El *algoritmo de Viterbi* fue inicialmente desarrollado para encontrar, dada una secuencia de símbolos, la serie de transiciones más probable entre los estados de una cadena de Markov necesaria para producir dicha secuencia [Forney,73]. Este problema es el equivalente markoviano al análisis sintáctico en una gramática regular estocástica.

El algoritmo de Viterbi es un caso particular del algoritmo de Programación Dinámica utilizado para encontrar un camino extremal en un grafo multietapa. Al igual que en el caso del análisis sintáctico para gramáticas regulares no deterministas, se recurre a un trellis, pero en este caso se define la función peso, no el dominio de los booleanos, sino en el intervalo [0..1], puesto que ahora representa la probabilidad de una regla o transición:

$$\rho(\,(j\text{-}1,u),(j,q)\,) \in [0..1]$$

y se sustituyen respectivamente las funciones "extremiza" por "max" y $\otimes$ por el producto:

$$C(j,q) = \max_{\forall u \in U_{j,q}} (\,C(j\text{-}1,u)\cdot\rho((j\text{-}1,u),(j,q))\,)$$

Al final del proceso $C(n,|Q|)$ nos proporciona la probabilidad (de máxima verosimilitud) de que la cadena analizada pertenezca al lenguaje de la gramática.

4.6 Versión iterativa del algoritmo de Viterbi

La versión iterativa del algoritmo de Viterbi se deriva inmediatamente de la transformación recursivo-iterativa expuesta para el esquema general de Programación Dinámica. El coste temporal del algoritmo es del mismo orden que el utilizado en el caso del análisis sintáctico en autómatas no estocásticos (y no deterministas): $O(n \cdot |Q| \cdot B)$. En cuanto al coste espacial, en principio de $O(n \cdot |Q|)$, se puede reducir a $O(|Q|)$ teniendo en cuenta que en cada etapa sólo se requieren los valores de costo de la etapa anterior. Se puede entonces almacenar sólo éstos últimos olvidando los de las etapas precedentes, ya utilizados; con lo que solo es necesario emplear dos vectores que se desplazan por el trellis (llamados aquí `P` y `P'`), en vez de la matriz G completa:

```
Algoritmo VITERBI
Datos    /* τ es el tipo "estado", σ el "símbolo", A=(V,Q,q0,F,δ) */
         V:Cσ  Q:Cτ    q0:τ    F:Cτ  α=a1a2...an:Lσ
resultado R:ℛ
Auxiliar  p:τ×σ×τ→ℛ /* probabilidad de una transición */
Variables P,P':Cτ×ℛ  /* Vectores, indexados por t */
          q,q':τ
Método
  ∀q'∈Q hacer P'[q']:=0 fin∀
  P'[q0]:=1;
  para j:=1..|α| hacer
     ∀q∈Q hacer P[q]:=    max      {P'[q']·p(q',aj,q)}
                       ∀q'∈δ⁻¹(q,aj)
     fin∀
     P':=P
  finpara
  R:=max (P[q])
     ∀q∈F
fin VITERBI
```

El algoritmo, tal como se describe más arriba, sólo nos proporciona el coste final (probabilidad) de la derivación más probable. Para conocer la derivación misma es necesario apuntarse, en cada paso de la recursión, qué decisión se ha tomado, con el fin de poder luego conocer la secuencia de reglas (estados) que se han utilizado. Ello incrementa la complejidad espacial otra vez en una cantidad equivalente al numero de vértices del trellis, con lo que nos queda otra vez $O(n \cdot |Q|)$

En aplicaciones prácticas es posible reducir drásticamente (en algunos casos hasta 1/30) el coste temporal mediante la aplicación simultánea de varias técnicas diferentes como pueden ser:

- Considerar para la decisión en cada etapa únicamente los estados alcanzados.
- *Búsqueda en Haz* («Beam Search»), en la que sólo se consideran los "mejores" caminos explorados, desechando los demás.

Un estudio detallado de éstos y otros métodos se halla en [Torró,89].

5

Corrección de errores

5.1 Reglas de error y trellis

Tal como se ha mencionado en el capítulo 2, el análisis sintáctico corrector de errores involucra un paso previo, consistente en la construcción de la gramática expandida de la gramática original. Ello implica, a primera vista, no sólo una enorme multiplicación en el número de reglas a analizar, sino también un esfuerzo considerable para generarlas y un espacio no menos considerable para almacenarlas. Sin embargo, cuando se trata de gramáticas regulares (en las que nos centraremos en el resto de este trabajo), es posible fundir en un solo y único proceso la generación de la gramática expandida y el análisis sintáctico corrector de errores. Ello posible gracias a que las reglas de error son deducibles directamente de las reglas de no error, lo que permite ir construyéndolas "al vuelo".

En el capítulo anterior se ha mostrado que se puede llevar a cabo el análisis sintáctico en gramáticas no deterministas buscando un camino extremal en un grafo, el *trellis*, construído a partir de la cadena a analizar y de la gramática (o su autómata equivalente). Utilizando la misma notación y representación que en dicho capítulo, se comprueba fácilmente el que, para una regla cualquiera de la gramática característica G=(N,V,P,S) $A \rightarrow aB$; $A,B \in N$, $a \in V$, y el símbolo i-ésimo $a_i \in V$ de una cadena $\alpha=a_1,a_2,\ldots,a_n$; la arista que deberá considerarse, al recorrer el trellis correspondiente, será la mostrada en trazo fino en la figura 5.1 (sólo se presentan las etapas consideradas, $\mathcal{V}_i$ y $\mathcal{V}_{i+1}$ y los vértices afectados por la regla en cuestión).

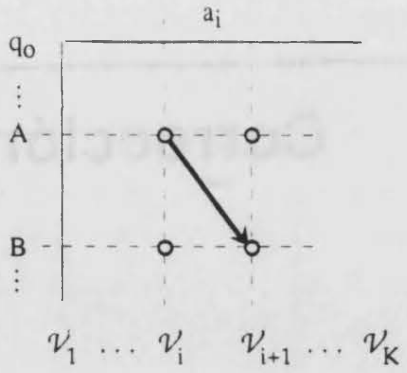


Figura 5.1 Subregión de trellis correspondiente al análisis sintáctico de la regla A→aB al presentarse el símbolo i-ésimo a_i (etapas $\mathcal{V}_i$ y $\mathcal{V}_{i+1}$).

A partir de esta representación, es inmediato construir las aristas a considerar (que corresponderían a reglas de la gramática expandida G^e) en el caso de admitir errores de sustitución, borrado o inserción (ver figura 5.2). Un error de substitución $A \rightarrow a_i B$ (a considerar sólo si $a \neq a_i$) corresponde a una arista que va entre los mismos vértices (mismos $A,B \in N$, pero en etapas consecutivas puesto que se "consume" a_i para aplicar la siguiente regla a a_{i+1}) pero con coste asociado a la substitución de a por a_i. Un error de inserción $A \rightarrow a_i A$ corresponde a una arista horizontal (se "consume" a_i, pero la siguiente regla a aplicar tiene el mismo no terminal A a la izquierda) que tendrá como coste el de insertar a_i. Por su parte, un error de borrado $A \rightarrow B$ corresponde a una arista vertical ("falta" el símbolo que permite pasar de A a B, pero la siguiente regla a aplicar ya tiene como parte izquierda a B, aunque seguirá aplicándose a a_i) y su coste será el de borrar a;

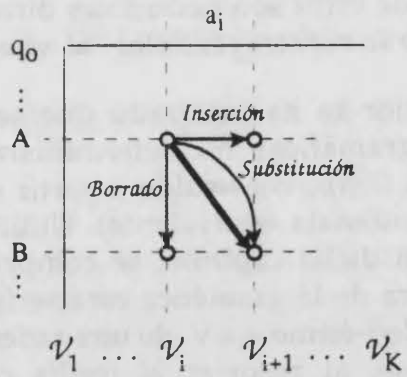


Figura 5.2 Subregión del correspondiente al análisis sintáctico de la regla A→aB al presentarse el símbolo i-ésimo a_i (etapas $\mathcal{V}_i$ y $\mathcal{V}_{i+1}$), y aristas (trazo punteado) asociadas a los errores de borrado, inserción y substitución.

En la figura 5.2 es posible observar que, si bien las reglas de sustitución (arista "paralela" a la de la regla normal) y las de inserción (arista horizontal) no presentan ninguna inconsistencia desde el punto de vista del trellis, no ocurre lo propio con las reglas de borrado (arista vertical), las cuales **incumplen la propia definición de trellis**: son aristas entre dos

vértices pertenecientes a una *misma* etapa. Esta anormalidad, a primera vista extraña, no lo es tanto si se considera que, tal como se han definido en el capítulo 2 los tipos de gramáticas, las reglas de borrado ni siquiera son regulares, y que, aunque es posible definir de una manera más general los lenguajes de tipo 3 [Saloma,87], admitiendo reglas de la forma $A \rightarrow \alpha B$; $A,B \in N$; $\alpha \in V^*$ que sí englobarían las reglas de borrado, ello no representaría ninguna ventaja a la hora de construir el trellis. El resto del capítulo se dedica a tratar de resolver esta dificultad, demostrando que no es tal si la gramática está *libre de circuitos* o si se modifica el algoritmo de `Viterbi`, añadiéndole un paso de reestimación (algoritmo `Cíclico`).

5.2 Autómatas libres de circuitos

Una gramática regular (autómata) libre de circuitos es aquella en la que ninguna secuencia de reglas aplicadas a partir de un no terminal lleva a él mismo:

$$\nexists A \stackrel{*}{\Rightarrow} \alpha A \qquad \forall A \in N, \alpha \in V^*$$

Veremos a continuación, que si una gramática cumple esta propiedad, es posible utilizar el algoritmo de `Viterbi` para llebar a cabo el análisis sintáctico corrector de errores. Veremos también que en el caso más general de una gramática regular *con* circuitos, el algoritmo **no** es aplicable.

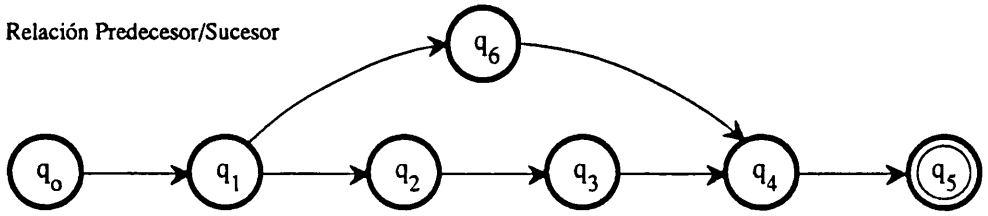
En efecto, para poder aplicar la versión iterativa del esquema de programación dinámica, la condición básica consiste en que, en un punto dado, *todos* los resultados anteriores estén calculados. En el caso del trellis, ello quiere decir que deben estar calculados todos los vértices de los cuales provenga una arista que llegue al vértice que estamos calculando. Como el cálculo progresa de etapa en etapa, es obvio que esta condición se cumple para todas las aristas provenientes de la etapa anterior, pero *nada garantiza* que sea cierta para aristas que provengan de vértices de la etapa que se está calculando, aristas que, como se ha visto, son inevitables si se consideran errores de borrado.

Según la definición de trellis, cada vértice corresponde a un estado q del autómata, y a ese vértice sólo pueden llegar aristas provenientes de vértices que corresponden a estados que sean *predecesores*[1] de q; ello independientemente de si estas aristas son debidas a transiciones normales o de error. Teniendo esto en cuenta, es evidente que, para poder utilizar el algoritmo de Viterbi considerando errores de borrado, debe de ser posible

---

[1] Un estado q_1 es predecesor de otro q si de q_1 salen transiciones que llegan a q, es decir $\exists a \in V \mid \delta(q_1,a)=q$.

ordenar los estados del autómata (y por lo tanto, los vértices en las etapas del trellis) de manera que *todos* los estados predecesores de uno dado sean anteriores a él. Si este orden es factible, implica una ordenación *parcial* del conjunto de estados y por lo tanto, la operación subsiguiente de encontrar un orden *lineal*, superconjunto de este orden parcial (con el fin de ordenar en columna los estados para formar una etapa del trellis), será lo que se conoce como *ordenación topológica* (ver figura 5.3). Es muy sencillo mostrar [Horowitz,77] que un orden parcial (y por lo tanto topológico) en un grafo dirigido sólo es posible si éste no tiene circuitos, lo que en nuestro caso implica el que el autómata (gramática) no los tenga[2].



Ordenes topológicos posibles:

q_0	q_1	q_2	q_6	q_3	q_4	q_5
q_0	q_1	q_6	q_2	q_3	q_4	q_5
q_0	q_1	q_2	q_3	q_6	q_4	q_5

Figura 5.3 Posibles órdenes topológicos de los estados de un autómata sencillo, todos compatibles con la Programación Dinámica. Obsérvese que en ningún caso el orden es total.

El siguiente algoritmo, extraído de [Horowitz,77], ordena topológicamente un grafo dirigido, escribiendo los vértices por orden y produciendo un error si esto es imposible por la presencia de algún ciclo:

```
Algoritmo OrdenTopológico
Método
  mientras haya vértices hacer
     si todo vértice tiene un predecesor
        entonces error: "Hay un ciclo" fin
     buscar un vértice v que no tiene predecesores
     escribir v
     borrar v y todas las aristas que salen de v
  finmientras
fin OrdenTopológico
```

[2] De hecho, un grafo dirigido SIN circuitos es la representación usual para un orden parcial.

5.2.1 El algoritmo ViterbiCorrector

Como ha quedado demostrado en el apartado anterior, y siempre que una gramática regular esté *libre de circuitos,* es posible utilizar el algoritmo de Viterbi para llevar a cabo el análisis sintáctico corrector de errores estocástico. Se presenta a continuación es una versión de este algoritmo, adaptada para esta aplicación en particular, mediante los añadidos necesarios para "generar" las reglas de error (la gramática expandida) a la vez que lleva a cabo el análisis sintáctico (algoritmo de *Viterbi con corrección de errores*):

```
Algoritmo ViterbiCorrector
Datos    /* τ es el tipo "estado", σ el "símbolo", A=(V,Q,q_o,F,δ) */
         V:C_σ  Q:C_τ    q_o:τ    F:C_τ  α=a_1a_2...a_n:L_σ
Resultado R:ℛ
Auxiliar                          /* ψ es el tipo "error" */
         p:τ×σ×τ→ℛ                /* probabilidad de una transición */
         p_e:τ×ψ×τ→ℛ              /* probabilidad de error en una transición */
         sig:τ→τ                  /* ordenación de estados*/

Variables P,P':C_τ×ℛ              /* Vectores, indexados por τ */
         q,q':τ
Método
  ∀q'∈Q hacer P'[q']:=0 finᐁ
  P'[q_o]:=1;
  para j:=1..|α| hacer /* Etapa del trellis */
     q:=q_o
     repetir
      P[q]:=max {
        /* reglas de no error: q->aq', a=a_j */
            max     {P'[q']·p(q',a_j,q)},
        ∀q'∈δ^-1(q,a_j)
        /* reglas de sustitución: q->a_jq', a≠a_j */
                max          {P'[q']·p_e(q',s_a|a_j,q)},
        ∀q'|q'∈δ^-1(q,a),a∈V,a≠a_j
        /* regla de inserción: q->a_jq */
        {P'[q]·p_e(q,i_a_j,q)},
        /* reglas de borrado: q->q' */
              max        {P[q']·p_e(q',b_a,q)}
        ∀q'|q'∈δ^-1(q,a),a∈V
        }
      q:=sig(q)
     hasta q=indefinido
     P':=P
  finpara
  R:= max (P[q])
     ∀q∈F
fin ViterbiCorrector
```

En el algoritmo se da un tratatamiento por separado a cada tipo de error, habiéndose escrito[3] la probabilidad de una transición (regla) de error entre los estados q y q' en cada caso como $p_e(q',s_{a|b},q)$ si es sustitución de a por b, $p_e(q,i_a,q)$ si es inserción de a y $p_e(q',b_a,q)$ si es borrado de a.

Puede observarse como en las reglas de borrado se utiliza el vector P (el que se esta calculando) en lugar del P'. Obviamente se ha supuesto que los estados están ordenados en el trellis según un orden topológico dado por la función "`sig`". El primer elemento de tal ordenación siempre será el estado inicial q_0.

5.3 Autómatas con circuitos

No es fácil encontrar en la literatura una solución completa y eficiente al problema del análisis sintáctico corrector de errores, en el caso más general de gramáticas regulares con circuitos. [Thomason,74] estudia el tema, pero no plantea una solución clara; la primera[4] solución completa parece haber sido expuesta en un trabajo muy poco conocido [Alpuente,89], y sólo recientemente [Bouloutas,91] trata el problema desde un punto de vista más general, en una exposición muy similar a la presentada a continuación, resolviendo el problema, pero sin proponer un algoritmo práctico de minimización.

5.3.1 El trellis tiene circuitos

Hemos visto que, en el caso de haber circuitos en el autómata, es inevitable la existencia de vértices del trellis cuyo cálculo depende de vértices **no** previamente calculados; es decir, que es imposible ordenar las aristas verticales (intra etapa) de manera que vayan todas en una dirección (el orden topológico); lo que implica que **hay circuitos en las etapas del trellis.**

El problema que se plantea es pues el de buscar el camino óptimo en la etapa de un trellis, la cual tiene circuitos, a partir de unos costes iniciales por vértice dados por la evaluación de las aristas provenientes de la etapa

---

[3] Nótese que estas probabilidades NO son las probabilidades de deformación, son las probabilidades *de las reglas*, posiblemente relacionadas según indica el capítulo 2.

[4] El autor no puede dejar de señalar, sin pretensión alguna de demostralo, que su programa ERRPAR, que implementa la solución presentada más adelante, ya funcionaba antes de que el problema fuera abordado por estos otros autores.

anterior. Obviamente, para ello se requiere un algoritmo que encuentre el *camino óptimo en un grafo dirigido y ponderado con circuitos.*

5.3.2 Camino óptimo en un grafo con circuitos

En lo que sigue nos situaremos en el caso de **minimización** del coste de un camino en el que el coste total es la **suma** de los costes elementales, todos positivos. Otros casos, como maximización o utilización del producto, son similares, siempre que se cumpla la condición que se discute más adelante.

La búsqueda del camino mínimo en un grafo es un problema típico de algorítmica, y su solución viene dada por el algoritmo de Dijkstra [Horowitz,77]; el cual se basa en la siguiente constatación: **Un camino por un grafo ponderado, que llega a un vértice del mismo, NO puede ser el mínimo si ya ha pasado por ese vértice.** Lo que resulta evidente, puesto que siempre será menos costoso la primera vez. En estas condiciones, a la hora de minimizar:

- Se pueden despreciar los *bucles* (aristas de un vértice a sí mismo): un camino que pasa por un bucle pasa más de una vez por el vértice.
- Para el resto de los circuitos en el grafo, se recorrerán las vueltas atrás como máximo UNA vez (y una vez como mínimo también: hay que comprobarlas).

Con lo que, para llevar a cabo la búsqueda del camino mínimo en un grafo de G=($\mathcal{V}$,A), con circuitos, se puede recurrir al siguiente algoritmo de PD:

```
Algoritmo MINCIRCUITOS
Datos    /* τ es el tipo "vértice", G=(V,A) */
         C0:C_{τ×R}          /* Vector de costos iniciales, indexado por τ */
Variables C:C_{τ×R}          /* Vector, indexado por τ */
          HayModif:B         /* Hay estados modificados */
          v,v':τ
          antes:R
Método
  C:=C0
  Repetir
     HayModif:=falso
     ∀v∈Q hacer
        antes:=C[v]
        C[v]:=    min     {C[v'] + Ce(v',v)}
              ∀v'|(v',v)∈A
        si C[v]≠antes entonces HayModif:=verdad
     fin∀
  hasta no HayModif
fin MINCIRCUITOS
```

Donde $C_e(v',v)$ es el peso de la arista (v',v). Este algoritmo es una versión del Dijkstra, en la que no se fuerza el vértice inicial (aunque ello se pueda conseguir simplemente inicializando a cero el coste acumulado del vértice inicial y a infinito el de los demás). Para poder determinar el vértice inicial del camino que llega a cada vértice, sería necesario anotarse en cada punto la decisión tomada; de la misma manera (ver capítulo 4) que se hace para determinar la derivación en el caso del análisis sintáctico con los algoritmos de PD.

Para definir la condición de fin se aprovecha implícitamente el hecho de que el camino mínimo no pasa dos veces por un mismo vértice. En efecto, a cada vuelta del bucle los caminos progresan de un vértice y se modifican los costes de aquellos vértices que son alcanzados, *si su valor precedente es mayor* (ver figura 5.4). Llegará pues inevitablemente un momento en el que el camino mínimo, o bien haya recorrido todos los estados, o bien le corresponda pasar por un estado por el que ya ha pasado; como llegado ese momento no se producirá ninguna modificación de costes, se podrá dar por terminado el proceso. La variable `Modif` del algoritmo es la que detecta si se ha producido esta modificación de costes.

Es obvio que el número de veces que se ejecuta el bucle es como máximo $|\mathcal{V}|-1$ y que, por lo tanto, la complejidad máxima es $(|\mathcal{V}|-1)|\mathcal{V}|$. En la práctica, el que se llegue o no a este máximo depende principalmente de lo similar que sea el orden en que se evalúan los vértices con respecto al orden "natural" del grafo (definido como la mejor aproximación posible a un orden topológico). Como ejemplo, se proporciona en la figura 5.4 un grafo que requiere el máximo número de reestimaciones, aunque no tiene

circuitos, lo cual se ha conseguido simplemente evaluando en sentido inverso al "natural" los estados del grafo. El autor postula, pero no demuestra, que el número de reestimaciones es igual, como máximo, al número de aristas del grafo que sean contrarias al orden de evaluación (mas una). Un ejemplo de ello se muestra en la parte inferior de la misma figura.

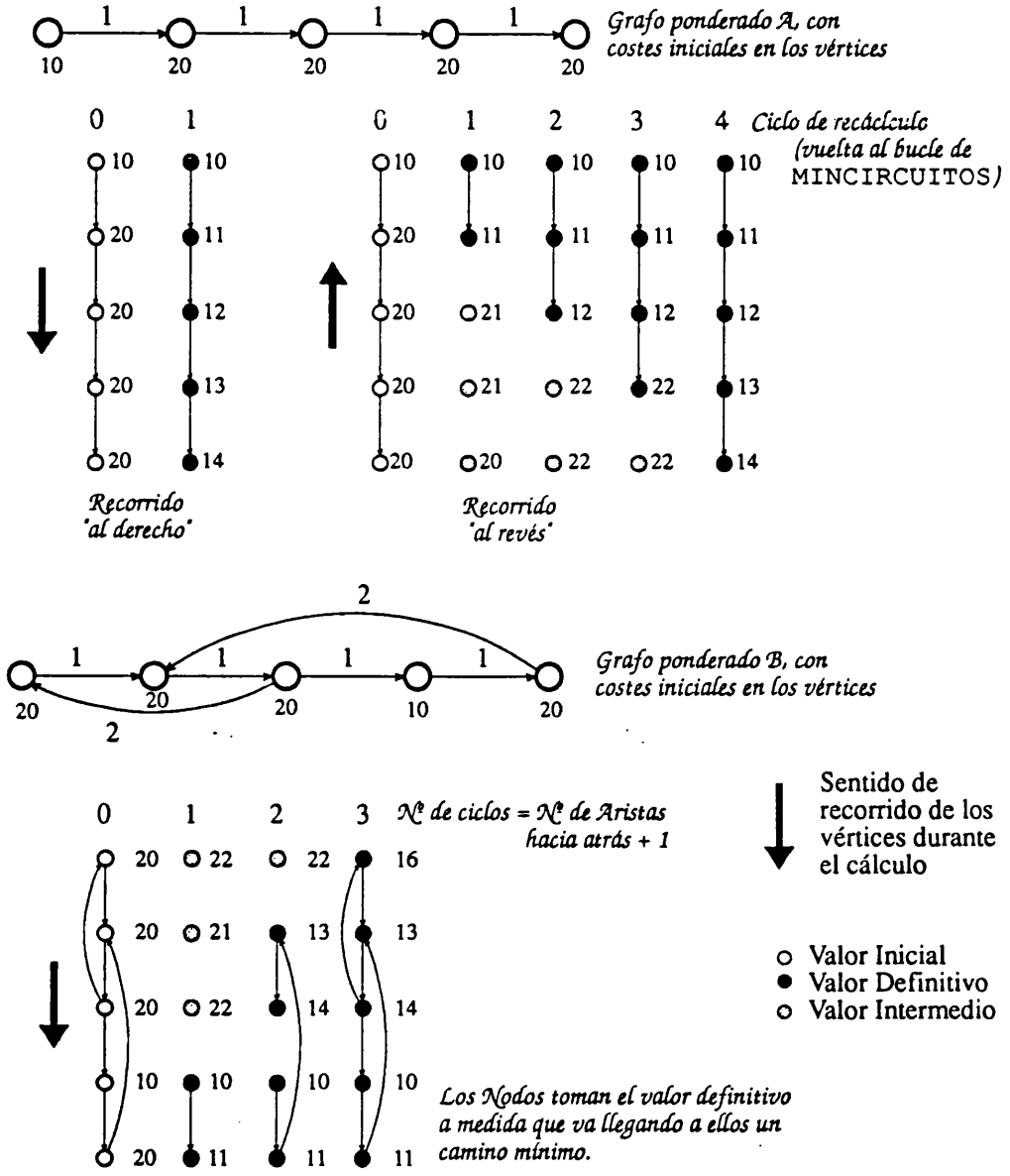


Figura 5.4 Ejemplos de recorrido por MINCIRCUITOS de los vértices de dos grafos, el primero según dos órdenes de evaluación distintos. Se muestran los grafos, con los costes iniciales en cada vértice y los pesos por arista. Debajo, se muestran los costes acumulados por vértice en cada ciclo del bucle del algoritmo: el ciclo 0 es el inicial, en los siguientes, los costes en cada vértice se calculan aplicando sucesivamente, y en el orden de evaluación, la minimización de MINCIRCUITOS. La secuencia de puntos negros indican hasta donde ha llegado el camino mínimo en cada ciclo. Ver cómo varía el número de reestimaciones (ciclos) en los primeros dos casos según el orden de evaluación y cómo depende del número de aristas en sentido contrario al de evaluación.

Por otro lado, para poder aplicar este algoritmo a problemas en el que el camino extremal buscado no sea el que minimize la suma de costes (p.e.:

maximización del producto de probabilidades), es necesario que la operación de acumulación (suma, producto, and lógico,...) sea contraria a la de decisión (mínimo, máximo, or lógico,...), es decir, que si el coste acumulado crece (decrece), la decisión busque minimizarlo (maximizarlo). Una razón sencilla para esto es obvia: si, por ejemplo, se maximiza la suma, cada vuelta a un circuito la haría crecer, con lo que el número de vueltas tendería a infinito. Exactamente lo mismo ocurriría si se plantea la existencia de pesos positivos y negativos: podrían entonces existir circuitos de coste total negativo, los cuales también harían tender el número de vueltas a infinito, aún si (por ejemplo) se minimiza la suma. Ambos casos efectivamente contradicen la hipótesis inicial, planteada al principio del apartado y en la que se basa el algoritmo.

5.3.3 El algoritmo CICLICO

La aplicación del algoritmo `MINCIRCUITOS` a un etapa de un trellis asociado a un autómata con circuitos es directa. De hecho, el algoritmo podría aplicarse a TODO el trellis en su conjunto (al fin y al cabo todo él es un grafo con circuitos), pero ello no se hace por la misma razón que no se hace en el algoritmo de Viterbi: se desaprovecharía la estructura multietapa del grafo.

Para una etapa del trellis, todos los vértices son iniciales (a todos llegan aristas de la etapa anterior) y todos son finales (de todos salen aristas a la etapa siguiente). Como los bucles son un caso particular de circuito, no es necesario considerarlos explícitamente en el que bautizaremos algoritmo `CILICO` (algoritmo para el análisis sintáctico corrector de erorres en autómatas con circuitos). Damos a continuación la versión iterativa del mismo, poniéndonos en el caso de gramáticas estocásticas, en el que el algoritmo `CILICO` es una extensión del ViterbiCorrector. Nótese que el algoritmo maximiza un producto lo cual no es problema, en contra de lo que pueda parecer después de lo dicho en el apartado anterior, puesto que todas las probabilidades son, por definición, menores que la unidad, con lo que su producto siempre decrece:

```
Algoritmo CICLICO          /*  A=(V,Q,q0,F,δ)  */
Datos    /* τ es el tipo "estado", σ el "símbolo", ψ es el tipo "error */
         V:Cσ  Q:Cτ    q0:τ    F:Cτ  α=a1a2...an:Lσ
resultado R:ℛ
Auxiliar  p:τ×σ×τ→ℛ      /* probabilidad de una transición */
          pe:τ×ψ×τ→ℛ     /* probabilidad de error en una transición */
Variables P,P':Cτ×ℛ       /* Vectores, indexados τ */
          Modif:Cτ        /* Estados modificados */
          HayModif:ℬ      /* Hay Estados modificados */
          q,q':τ     antes:ℛ
Método
  ∀q'∈Q hacer P'[q']:=0 finV;      P'[q0]:=1;
  para j:=1..|α| hacer
     Modif:=∅
     ∀q∈Q hacer          /* Etapa del trellis: Primera evaluación */
          P[q]:=max {
          /* reglas de no error: q->aq', a=aj */
               max     {P'[q']·p(q',aj,q)},
          ∀q'∈δ^-1(q,aj)
          /* reglas de sustitución: q->αjq', a≠aj */
                    max          {P'[q']·p(q',sa|aj,q)}
          ∀q'|q'∈δ^-1(q,a),a∈V, a≠aj
          /* regla de inserción: q->ajq */
          {P'[q]·p(q,ia_j,q)},
          /* reglas de borrado: q->q' */
                     max            {P[q']·p(q',ba,q)}
          ∀q'|q'∈δ^-1(q,a),a∈V,q'∈Modif
             }
          Modif:=Modif+{q}
     finV
     Repetir     /* Maximización de la etapa con circuitos */
        HayModif:=falso
        ∀q∈Q hacer
           antes:=P[q]
           P[q]:=max {
              /* reglas de borrado: q->q' */
                    max       {P[q']·p(q',ba,q)}  }
              ∀q'|q'∈δ^-1(q,a),a∈V
           si P[q]≠antes entonces HayModif:=verdad
        finV
     hasta no HayModif
     P':=P
  finpara
  R:= max (P[q])
     ∀q∈F
fin CICLICO
```

Se comprueba inmediatamente que las diferencias con el algoritmo `ViterbiCorrector` se centran exclusivamente en el tratamiento de los errores de borrado y en la supresión de la función "`sig`", ahora imposible. En la primera evaluación, se hace necesario restringir la maximización a los vértices ya calculados, especialmente al tratar con las aristas (transiciones) de borrado, únicas problemáticas; es para ello para lo que se usa el conjunto `Modif`. En la maximización de la etapa con circuitos se vuelven a comprobar estas mismas aristas de borrado, pero sólo éstas, puesto que son las únicas que pueden variar el resultado; el resto del algoritmo es idéntico a `MINCIRCUITOS`.

5.4 Camino y traza de edición

La derivación óptima, obtenida tras el análisis sintáctico corrector de errores de una cadena por una gramática (regular), está constituída por una secuencia de reglas, que el caso más general serán algunas pertenecientes a la gramática característica (reglas de no error), y otras a la expandida (reglas de error). Cada una de estas reglas ha sido utilizada para derivar un símbolo concreto de la cadena analizada, y está asociada (a través de la misma derivación óptima) a alguna de las reglas que se utilizarían para generar la cadena de la gramática más semejante (según el criterio de disimilitud utilizado, ver capítulo 2) a dicha cadena analizada. Toda esta información, directamente presente en la derivación óptima, se puede resumir de forma gráfica, habiéndose utilizado en este trabajo mayormente dos tipos de dibujo: la *traza* y el *camino* de edición.

Ambas representaciones gráficas son inmediatamente obtenibles, al igual que la derivación óptima, si durante análisis sintáctico por PD (p.e.: mediante ViterbiCorrector) se ha tomado la precaución de anotarse la decisión (transición) tomada en cada vértice del trellis. Una vez terminado el análisis, tanto la derivación óptima, como la traza y el camino de edición se extraen con sólo seguir las transiciones anotadas, empezando desde el último vértice del trellis (correspondiente al estado final y al último símbolo de la cadena), y terminando en el primero (correspondiente al estado inicial y al primer carácter de la cadena).

La **traza de edición** (ver figura 5.5) permite comparar la cadena de la gramática más semejante a la analizada con esta misma, mostrando qué símbolos de la cadena analizada han sido generados con reglas existentes en la gramática y cuáles por reglas de error. Se escriben las dos cadenas paralelamente la una a la otra, y se dibujan trazos continuos uniendo los símbolos que han sido generados por las mismas reglas (de no error). A veces también se unen, con trazos discontinuos, los símbolos generados por reglas de error, con los símbolos (de la cadena más semejante) generados por las reglas de la gramática característica asociadas a dichas reglas de error.

Alternativamente a esto último, se pueden señalar con un subrayado los símbolos borrados o insertados (pero no los substituídos) [Sankoff,83].

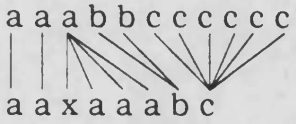


Figura 5.5 Traza de edición entre la cadena "aaabbcccccc" y la cadena "aaxaaabc". Los trazos continuos representan reglas de no error, los discontinuos reglas de error.

El **camino de edición** (figura 5.6) muestra la misma información que la traza, pero representa más claramente las reglas de error. Las dos cadenas (la analizada y la más semejante de la gramática) se escriben perpendicularmente a los lados de una rejilla[5]. La derivación óptima se representa como un camino que va entre los dos vértices extremos de la rejilla (correspondiente a los símbolos iniciales y finales respectivamente), de tal manera que una progresión en diagonal equivale a una substitución (o a una regla de no error: substitución del símbolo por sí mismo), una progresión vertical a un borrado y una horizontal a una inserción. Los símbolos substituídos, insertados o borrados se visualizan con sólo comprobar a qué símbolos corresponde el punto que se está considerando. Obsérvese que el camino de edición se asemeja a lo que se conoce en alineamiento temporal de cadenas como el *camino de alineamiento* entre dos cadenas [Wagner,74], pero en realidad corresponde a un concepto distinto [Sankoff,83].

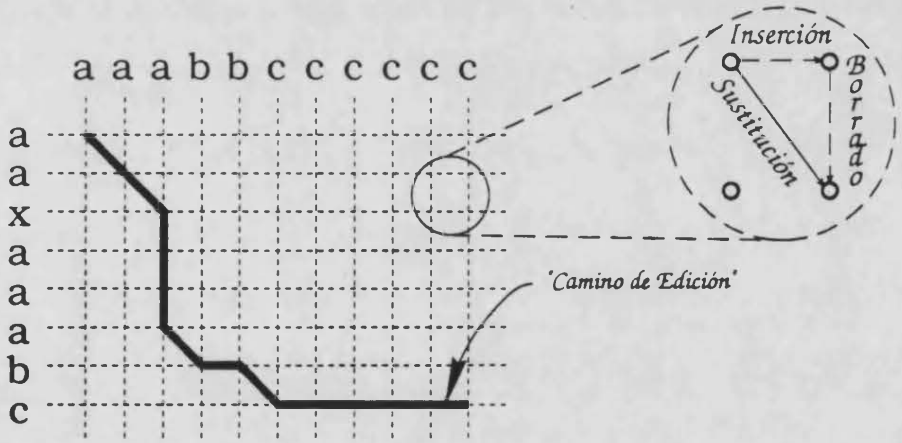


Figura 5.6 Camino de edición (trazo grueso) entre las mismas cadenas de la figura anterior ("aaabbcccccc" y "aaxaaabc"). Según un tramo del camino sea diagonal, vertical u horizontal representa un error de substitución, de inserción o borrado respectivamente. Si los símbolos a los lados de la rejilla indican que la substitución es por el mismo símbolo, se trata de una regla de no error.

---

[5] Esta rejilla y el camino de derivación equivalen (casi) al trellis que se tendría si el lenguaje de la gramática estuviera compuesto únicamente por la cadena más semejante a la analizada, y se marcara en él las reglas de la derivación óptima.

De forma arbitraria, en este trabajo se ha escogido el criterio de colocar la cadena analizada encima (en la traza de edición) o verticalmente (en el camino de edición).

6

El método ECGI

6.1 Motivación

El algoritmo de Inferencia Gramatical mediante Corrección de Errores (Error-Correcting Grammatical Inference: **ECGI**), es un método **heurístico** de inferencia gramatical, que fue diseñado en su origen para una aplicación concreta: el Reconocimiento de Palabras Aisladas. Sin embargo, las características impuestas implícitamente por ECGI (por sus heurísticos) a los lenguajes inferidos han demostrado ser comunes a muchos otros campos del Reconocimiento de Formas. Ello es debido a que ECGI intenta solventar algunos de los inconvenientes básicos que presentan, para estas aplicaciones, la mayoría de los métodos de inferencia gramatical publicados hasta el momento. Estos inconvenientes son causados por las características del extralenguaje generado por el método de inferencia, que suele ser:

- Extremadamente *recursivo*, lo que implica que ignora la posición relativa de las diferentes subestructuras en la cadena muestra, reutilizando una misma subestructura independientemente de su posición en dicha cadena (p.e.: método uv^iw [Miclet,79], método del sucesor y antecesor-sucesor [Richetin,84], lenguajes k-explorables [García,88]).

- *Infinito;* lo que significa que deriva de las muestras de aprendizaje por repetición indefinida de subestructuras de dichas muestras (p.e.: los anteriores y también todos los métodos que se basan en el árbol aceptor de prefijos: k-colas [Biermann,72], k-RI [Angluin,82], método de Levine [Muggleton,84], de comparación de finales [Miclett,80]).

Aunque en principio estas características parezcan derivar de un procedimiento natural de generalización, tienen como consecuencia el que la gramáticas inferidas sean (figura 6.1):

1) incapaces de representar características basadas en la **longitud de las subestructuras** (ver apartado 1.6), pues la gramática inferida tolera longitudes arbitrarias, al no controlar el número de repeticiones de dichas subestructuras.

2) excesivamente tolerantes con las **posiciones relativas** de dichas subestructuras.

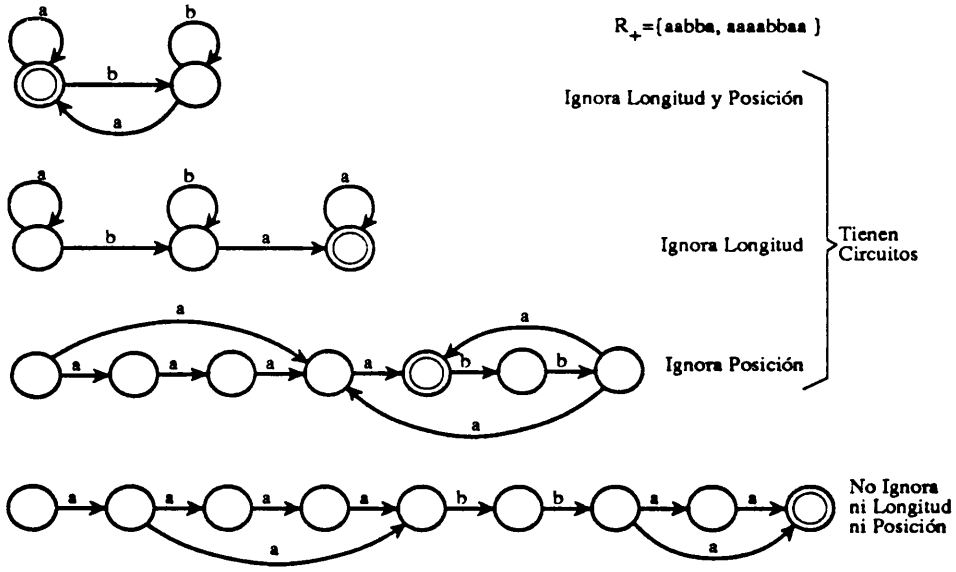


Figura 6.1 Generalizaciones que ignoran o no la longitud y posición de las subestructuras, efectuadas a partir de un par de cadenas muestra. Obsérvese que la única que no ignora ni la longitud ni la posición es la que no tiene circuitos (caso, como se vera, de ECGI), aunque ello limita fuertemente la generalización.

En lo que se refiere al primero de estos inconvenientes, la modelización de la longitud, es corriente resolverlo mediante lo que se conoce como las *aproximaciones híbridas*, en las que se añaden al modelo estructural ciertos *atributos numéricos*, encargados de representar la información complementaria. Entre estas aproximaciones, la más conocida reside en emplear gramáticas estocásticas (o equivalentemente, modelos de Markov y/o modelos de Markov con duración), ya que en ellas la modelización de longitud se halla implícita en las probabilidades de los bucles. Sin embargo, se puede demostrar que esto lleva a una muy inadecuada distribución *geométrica* de las probabilidades para distintas longitudes [Juang,85] [Rusell,85]. Por otro lado, otras posibilidades inmediatas para la modelización de longitud, consisten en introducir "contadores" y "umbrales de longitud" en los estados del autómata.

Todos estos procedimientos pueden verse como casos particulares de las *Gramáticas de Atributos*, para las cuales se ha mostrado que se puede obtener un compromiso arbitrario entre complejidad estructural (reglas) y semántica (atributos) [Fu,83].

Cabe pensar, sin embargo, que la alternativa más "limpia" se halle en un extremo de este compromiso, en el que la modelización de longitud se

incluye en la misma estructura, es decir, se representa mediante la sintaxis. Este tipo de modelización admite además una solución directa, que a la vez solventa el problema de tener en cuenta la posición relativa de las subestructuras: las gramáticas *no recursivas* (ni a derechas, ni a izquierdas), es decir, las *gramáticas sin circuitos*. Lamentablemente, esta aproximación hace inaplicable la mayoría de los métodos conocidos de inferencia gramatical, lo que conduce a proponer y estudiar un nuevo algoritmo: ECGI.

6.2 Método

El objetivo al que se destinó ECGI, desde un principio, fue el construir *incrementalmente* una gramática regular a partir de presentación positiva. La idea del método surgió de considerar qué mejoras serían aplicables al procedimiento más sencillo de inferencia gramatical: el que genera la *gramática canónica*.

6.2.1 Un algoritmo trivial

Dado un conjunto finito de cadenas muestra, existe un procedimiento trivial y bien conocido para generar incrementalmente una gramática regular. Basta para ello aplicar la siguiente regla: a cada nueva muestra $\alpha=\alpha_1..\alpha_n$ añádase a la gramática los no terminales $A_1..A_n \in N$, y las reglas $(S\rightarrow\alpha_1A_1), (A_1\rightarrow\alpha_2A_2),\ldots,(A_{n-1}\rightarrow\alpha_n)\in P$ (figura 6.2).

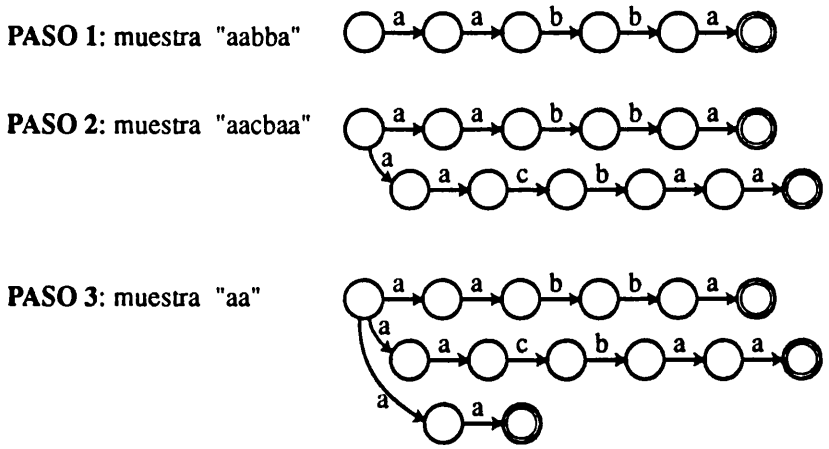


Figura 6.2 Construcción incremental de un autómata canónico.

Este procedimiento genera la *gramática canónica* del conjunto de muestras, gramática cuyo lenguaje se caracteriza por contener a todas las muestras, pero a ninguna cadena más. Como es fácilmente imaginable, el inferir una gramática de este tipo no es excesivamente útil. Al no efectuarse

ninguna generalización, la gramática no sólo es incapaz de reconocer ninguna muestra que no se le haya dado, sino que además tiene que almacenarse todas las cadenas que le han enseñado. Este es un típico caso de *aprendizaje por memorización*, caso degenerado del aprendizaje inductivo, tal como se ha presentado en el apartado 1.71.

6.2.3 ECGI

Enfocando el problema desde un punto de vista más general, si un método constructivo de inferencia gramátical M (ver capítulo 3), en un paso dado i, tiene como hipótesis, en ese paso, la gramática $G_i=(N_i,V,P_i,S)$, entonces, para realizar la hipótesis $G_{i+1}=(N_{i+1},V,P_{i+1},S)$ al recibir la siguiente cadena muestra α_i, usualmente (puesto que es la manera intuitivamente más sencilla de ser conservativo) añadirá y/o suprimirá de G_i un conjunto, relativamente reducido (también para ser conservativo), de reglas. Como, por otra parte, los métodos existentes de inferencia en general no *suprimen* reglas, sino que siempre añaden reglas nuevas (debido principalmente a que sólo emplean muestras positivas y que parten de una gramática sin reglas), tiene sentido hablar del conjunto de reglas que M añade en el paso i (reglas que "no están" en la gramática de paso i), que se escribirá $NEST=P_i \cap P_{i+1}$. Además, dado que, *excepto en el caso de generar la gramática canónica*, en NEST no están usualmente todas las reglas necesarias para generar α_i, es obvio que ello quiere decir que M ha decidido que "ya existen" cierto número de reglas en G_i que permiten generar determinadas subcadenas de α_i (puesto que si M es un método consistente de inferencia, G_{i+1} debe poder generar α_i). Con lo que, si $D(\alpha_i,G_{i+1})=r_1,r_2,\ldots,r_n;\ r_k \in P_{i+1};\ k=1\ldots n;$ es la derivación de α_i por G_{i+1}, entonces usualmente $\exists r_k \in D(\alpha_i,G_{i+1}) \mid r_k \in P_i;$ y podremos llamar $YEST=\{ r_k : r_k \in D(\alpha_i,G_{i+1}) \text{ y } r_k \in P_i\}$ al conjunto de reglas que "ya están" en G_i y por lo tanto no son añadidas por M al postular G_{i+1}.

En consecuencia, el problema principal de un método de inferencia gramatical, si quiere evitar el verse reducido a generar la gramática canónica, consiste en disponer de un método que, dada una cierta cadena muestra, le permita descubrir aquellas reglas que pertenecen a YEST (que "ya están" en la gramática).

Para ello, toda la dificultad estriba en que las cadenas muestra no llevan ninguna referencia a las reglas que las han producido, con lo que la única seguridad de que dispone un método es que dos símbolos distintos no pueden haber sido producidos por la misma regla[1]. El método ECGI, así como la mayoría de los otros métodos de inferencia gramatical, aprovechan

[1] A menos que haya errores, pero en este caso supondremos que se trata de inferir, no sólo la gramática original, sino también los errores que se producen sobre ella: las "reglas de error" reales.

esta certeza, e identifican grupos de reglas básicamente por similitud (según un determinado criterio, que no necesariamente implica igualdad) de las subcadenas de símbolos que generan. ECGI, además, asume que una subcadena que aparece en una determinada posición de la cadena muestra (principio, fin, centro, detrás de otra,...), no puede haber sido generada por las mismas reglas que otra subcadena, aunque sea similar, que aparezca en una posición distinta.

Por otra parte, muchos métodos de inferencia gramatical subdividen la cadena muestra en subcadenas que cumplen cierta característica (ser de una longitud determinada, terminar de una manera u otra,...) (k-colas, uv^iw, k-EEEI,...) y sólo entonces buscan **las subcadenas** en el lenguaje de la gramática para encontrar qué reglas pueden haberlas producido[2]. Lo que es más, esta búsqueda se realiza independientemente para cada subcadena. Sin embargo, ECGI busca, en el lenguaje de la gramática actual[3], la **cadena más similar globalmente** a la cadena muestra; aplicando un criterio de similitud de cadenas que tiene muy en cuenta la posición relativa de las subcadenas.

Una vez localizada la cadena de la gramática actual más similar a la cadena muestra, ECGI, para completar un paso de inferencia, no tiene más que asumir que las reglas que han generado subcadenas idénticas (según el criterio de similitud) en la muestra y en la cadena más similar, pertenecen a YEST (son las que "ya están"), y por lo tanto, las que pertenecen a NEST ("no están"), y que hay que añadir, son las que faltan para generar totalmente la cadena muestra.

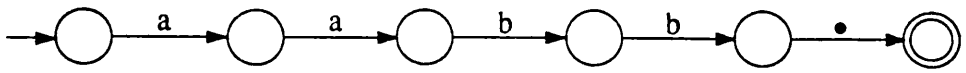
En la práctica, para buscar en la gramática actual la cadena más similar a la cadena muestra (la fase de **comparación** de ECGI) se aplican los métodos de análisis sintáctico corrector de errores descritos en el capítulo 5. La derivación óptima de la cadena muestra con la gramática expandida proporciona la secuencia de reglas de error y no error, las cuales ECGI analiza en la fase de **construcción**, para descubrir cúales son las nuevas reglas a añadir para que la gramática acepte la cadena muestra. Las reglas de no error de la derivación óptima (reglas de YEST: que "ya están" en la gramática sin expandir) generan determinadas subcadenas de la cadena muestra, entre las cuales se hallan subcadenas generadas por las reglas de error (reglas de NEST). Para que la gramática pueda aceptar totalmente la cadena muestra, basta añadir, entre cada par de subsecuencias de reglas de YEST, un conjunto de reglas que generen las subcadenas que han sido reconocidas por las reglas de error (figura 6.3).

---

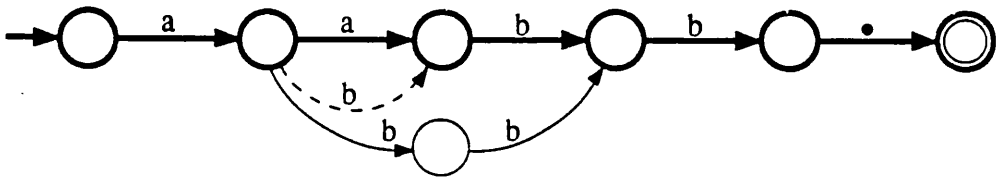
[2] En el caso de los métodos no incrementales, la búsqueda de subcadenas similares se realiza en el mismo conjunto de muestras de aprendizaje.

[3] En adelante, al considerar una nueva cadena muestra α_i, llamaremos "gramática actual" a la gramática inferida hasta ese momento por un método incremental de inferencia, es decir, a la gramática inferida por el método a partir del conjunto de cadenas $R_+=\{\alpha_1,...\alpha_{i-1}\}$.

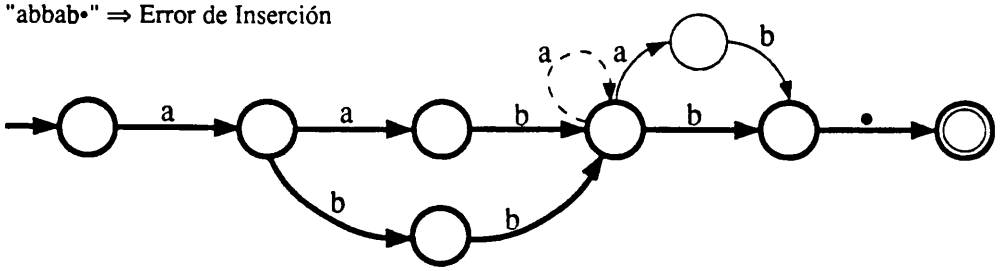
"aabb•" (Primera Cadena) ⇒ Autómata Canónico



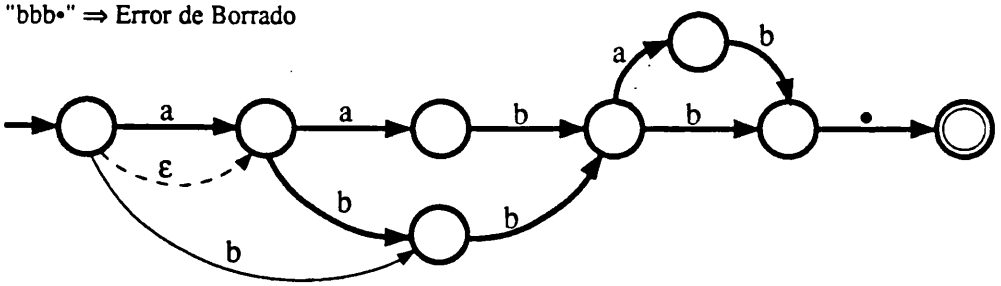
"abbb•" ⇒ Error de Substitución



"abbab•" ⇒ Error de Inserción



"bbb•" ⇒ Error de Borrado



Autómata resultante:

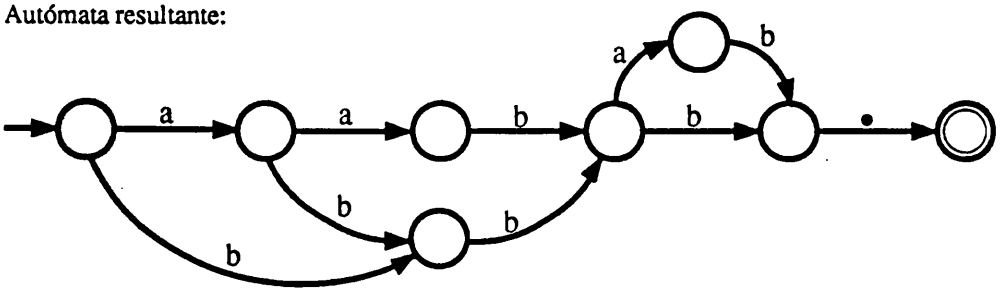


Figura 6.3 Traza de la inferencia por ECGI de una gramática a partir de R_+={aabb,abbb,abbab,bbb}. Los trazos gruesos representan, en los sucesivos autómatas, transiciones (reglas) y estados (no terminales) ya consolidadas. En cada paso, se muestran en trazo discontinuo las reglas de error, mientras que las reglas y no terminales añadidos por ECGI se muestran en trazo fino. Nótese el símbolo final que ECGI añade a todas las cadenas para forzar un único estado final.

Expresado más formalmente: el análisis corrector de errores de la cadena α por la gramática G=(N,V,P,S), proporciona la derivación óptima

$D(a,G)=r_1,r_2,..,r_k$; $r_j \in P^e$; j=1..k; en la gramática expandida $G^e=(N^e,V,P^e,S)$. Toda subsecuencia de reglas de error en la derivación óptima $r_{s+1},\ldots,r_{t-1} \notin P$ $r_j \in NEST$; j=s+1..t-1; genera una determinada subcadena $\omega=z_1,\ldots,z_h$ de la cadena muestra $\alpha=c_1,\ldots,c_s,\omega,c_t,\ldots,c_l$ que no es generable por G (hemos llamado c_s y c_t a los terminales generados por las reglas de no error r_s y r_t ; $r_s,r_t \in$ YEST y que flanquean a las reglas de error que generan ω: $D(\alpha,G)=r_1,\ldots,r_s,r_{s+1},\ldots,r_{t-1},r_t,\ldots,r_k)$. Para que la gramática G pueda generar la cadena α, deberá poder generar todas las subcadenas generadas por las reglas de no error de la derivación óptima, cosa que ya hace; y todas las subcadenas generadas por las reglas de error (p.e.:ω). Basta pues añadir a P las reglas necesarias para que esto último se cumpla. Si r_s es $(C_{s-1}\rightarrow c_sC_s)$ y r_t es $(C_{t-1}\rightarrow c_tC_t)$; estas reglas serán, para el caso de ω, $(C_s\rightarrow z_1Z_1)$, $(Z_1\rightarrow z_2Z_2)$,..., $(Z_{h-1}\rightarrow z_hZ_h)$, $(Z_h\rightarrow c_tC_t)$; donde $Z_1,\ldots,Z_h$ son nuevos no terminales generados para la ocasión. Si se da la circunstancia que ω es la cadena vacía, es decir que la secuencia de reglas de error $r_{s+1},\ldots,r_{t-1}$ no generan terminales en α, es que son reglas de borrado, siendo por lo tanto necesario generar la regla $(C_s\rightarrow c_tC_t)$, que permite evitar las reglas intermedias en G que son necesarias para reescribir C_s como C_t.

Debe resaltarse que **no** se añaden a la gramática las reglas de error de la derivación óptima, sino que se genera e inserta una nueva secuencia de reglas, que cumplen únicamente la condición de generar (suprimir) la subcadena de la cadena muestra ausente (sobrante) en la gramática. La nueva secuencia de reglas no sólo añade a la gramática la cadena muestra, sino tambien todas las posibles cadenas generadas por combinación de esta nueva secuencia con todas las otras antes añadidas, siendo éste el origen del poder de generalización de ECGI. Como se verá más adelante, la decisión de qué reglas exactamente se añaden, de entre las múltiples combinaciones que generarían (suprimirían) la misma subcadena, es la determinante principal de las características que son propias a todas las gramáticas que genera ECGI.

6.3 Algoritmo

Después de lo dicho en el apartado anterior, es posible sin más preámbulos presentar formalmente el algoritmo del método de inferencia gramatical mediante corrección de errores (la relación de orden (N,<) se discute al final del apartado):

```
Algoritmo ECGI
Datos     /* Conjunto de muestras positivas (cadenas) */
          R+={α0, α1, α2, ..., αm}; αi∈V*

Resultado Gm=(Nm, V, Pm, S)      /* Gramática regular inferida */
          (N, <)                 /* Relación de orden en N */

Inicialización    /* Gramática canónica de α0 */
          α0≡a1, a2, ..., an
          N0:={ A0, A1, ..., An-1}; S:=A0
          P0:={ Ai-1→aiAi, i=1, ..., n-1 } ∪ {An-1→an}
          G0:={N0, V, P0, S}
          (N, <):=A0<A1<...<An-1

Inferencia
  ∀ i=1..m hacer
     Comparación
        Obtener /* Derivación óptima de αi en Gei */
        D(αi, Gi)≡r1, r2, ..., rk;  rj∈Pei; j=1...k
     Construcción
        ∀ rs, rs+1, ..., rt-1, rt∈D(αi, Gi)
          tales que rs, rt∈Pi  /* NO son de ERROR */
          ∧ rs+1, ..., rt-1∉Pi  /* SON de ERROR */
        hacer
          sea
             rs≡(Cs-1→csCs);   rt≡(Ct-1→ctCt)
             xi≡c1, ..., cs, ω, ct, ..., cl;   ω≡z1...zh
             (Ni, <)≡S<...<Cs<...<Ct<...
          si ω=λ
          entonces
             Pi+1:=Pi ∪ { (Cs→ctCt) }
          sino
             Pi+1:=Pi ∪ { (Cs→z1Z1), (Z1→z2Z2),
                  ...(Zh-1→zhZh), (Zh→ctCt) }
             Ni+1:=Ni ∪ { Z1, ..., Zh }
             (Ni+1, <):=S<...<Cs<...<Z1<...<Zh<...<Ct<...
          finsi
        finpara
  finpara
fin ECGI
```

En el algoritmo aparecen bien diferenciadas las dos etapas, "comparación" (en la que se busca la derivación óptima mediante análisis sintáctico corrector de errores) y "construcción". Se comprueba, como ya se dijo en el apartado anterior, que el algoritmo de construcción genera y añade una secuencia de reglas totalmente nueva, diferente de las reglas de error propias a la derivación óptima. La construcción **no** aprovecha no terminales que pudieran estar en las reglas de error (en una sustitución por ejemplo) y ni siquera los tiene en cuenta para escoger qué reglas añadir. Sólo se

preocupa de enlazar (dentro del **si-entonces-sino**) con las reglas de no error a cada extremo de la nueva subcadena que debe generar la gramática.

La complejidad del algoritmo se encuentra en su casi totalidad concentrada en el análisis corrector de errores necesario para obtener la derivación óptima. Ya se ha visto que esto supone, para cada cadena α_i de longitud $|\alpha_i|=n$, un coste temporal $O(n\cdot|Q|\cdot B)$ y un coste espacial $O(n\cdot|Q|)$; donde $B=B_0\cdot(2\cdot|V|+1)$ debido al modelo de error utilizado. B_0 (*branching factor* o número de reglas promedio con el mismo no terminal a la izquierda) depende de la complejidad de la gramática inferida.

Hay que notar la especial precaución que se ha tenido, a la hora de añadir no terminales, en construir y conservar un orden en el conjunto de los mismos. Recuérdese (capítulo 5) que este orden es fundamental para poder realizar en análisis corrector de errores con el algoritmo `ViterbiCorrector` y no tener que recurrir a `Cíclico`. Se consigue un orden que evita el que un no terminal se pueda derivar de uno que sea anterior a él, simplemente (y siempre que no haya circuitos) insertando los nuevos no terminales en cualquier lugar entre los no terminales C_s y C_t de la primera y última reglas añadidas. C_s y C_t pertenecen a reglas de no error, y por lo tanto, existen ya en la gramática y están correctamente ordenados. En la práctica, los nuevos no terminales se insertan justo antes de C_t.

6.4 Propiedades de la gramática inferida

A partir de la descripción del algoritmo ECGI, es inmediato extraer algunas de las propiedades comunes a todas las gramáticas que infiere. Es evidente, pues, que si una gramática ha sido inferida mediante ECGI a partir de un conjunto de muestras R_+:

- Su lenguaje contiene a R_+.
- Es regular.
- Es **no** determinista (aunque, como se verá en el capítulo 12, puede conseguirse que sea determinista).

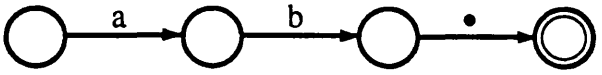
También, menos evidentemente, y como se irá justificando más adelante:

- Es ambigua.
- No tiene circuitos.
- El lenguaje que genera es finito.

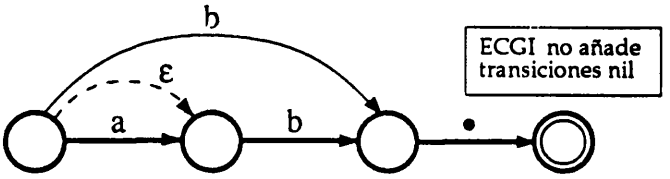
Y debido a la implementación realizada:

- Es representable mediante un autómata de estados etiquetados (LAS, ver capítulo 2).

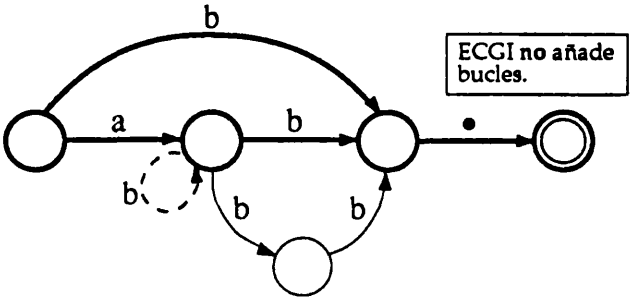
"ab•" (Primera Cadena) ⇒ Autómata Canónico:



"b•" ⇒ Error de Borrado:



"abb•" ⇒ Error de Inserción:



"bbb•" ⇒ Error de Substitución:

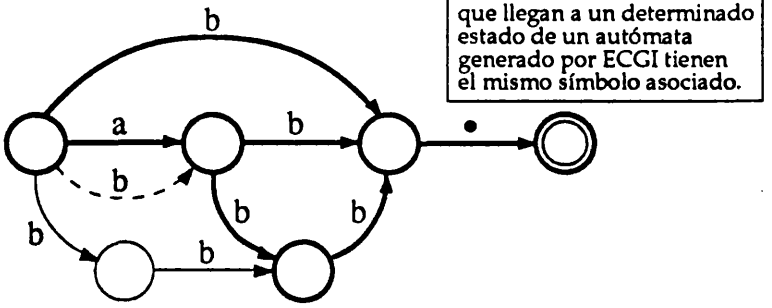


Figura 6.4 Traza (similar a la de la figura anterior) del proceso de inferencia por ECGI a partir de R_+={ab,b,abb,bbb}. Se muestran algunas peculiaridades del algoritmo de construcción. Nótese además que en c) el punto de inserción podría haber sido el estado siguiente, y que el autómata final d) es no determinista.

6.5 Heurísticos

Dedicaremos los siguientes apartados a un estudio de los heurísticos implícitos en el algoritmo de construcción, y a una clarificación del *porqué* de las características que caben esperar de una gramática inferida por ECGI (figura 6.4).

6.5.1 Las asunciones de ECGI

De la descripción realizada en el apartado anterior, es posible deducir que dada la gramática actual del paso i de inferencia G_i, la cadena muestra α en ese paso, y la cadena $\beta \in L(G_i)$ más similar a α, en ECGI se hacen las siguiente asunciones:

1) Todas las reglas que han servido para generar α, que no son de error, son reglas que han servido para generar β. Es decir, $\forall r_k \in D(\alpha, G_{i+1})$, $r_k \in YEST \Leftrightarrow r_k \in D(\beta, G_i)$.

2) De las reglas que han servido para generar β por G, sirven para generar α **todas** aquellas que generen subcadenas de α similares (en un sentido a definir) a subcadenas de β, que se hallen **en la misma posición relativa** en α y β. Es decir, sea una descomposición arbitraria de α en n subcadenas $\omega_k \in V^*$; $k=1...n$; $\alpha=\omega_1\omega_2...\omega_s...\omega_t...\omega_n$; y una de β en m subcadenas $\varphi_k \in V^*$; $k=1...m$; $\beta=\varphi_1\varphi_2...\varphi_u...\varphi_v...\varphi_m$; $\varphi_k \in V^*$; $k=1...m$; sea $Simil(\omega,\varphi): V^* \times V^* \rightarrow \mathcal{B}$ una función que devuelve "verdad" si ω y φ son similares, y $R_{\varphi l}=r_1,...,r_{|\varphi l|}$ la secuencia de reglas que ha servido para generar la subcadena φ_l de β, entonces, dadas un par de subsecuencias $\varphi_u, \varphi_v \in \beta$; $u<v$; entonces $R_{\varphi_u} \in YEST \wedge R_{\varphi_v} \in YEST \Leftrightarrow \exists \omega_s, \omega_t;\ s<t : Simil(\omega_s, \varphi_u) \wedge Simil(\omega_t, \varphi_v)$.

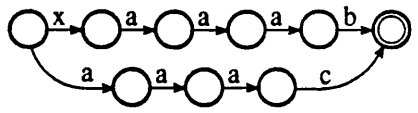
Estas dos asunciones constituyen el heurístico principal de ECGI, y junto con el procedimiento de búsqueda mediante programación dinámica (corrección de errores), caracterizan a ECGI como algoritmo de inferencia gramatical.

De la primera asunción se deduce inmediatamente el corolario (figura 6.5):

C.1: Ninguna regla de la gramática actual, distinta de las que han participado en la generación de β, puede pertenecer a YEST.

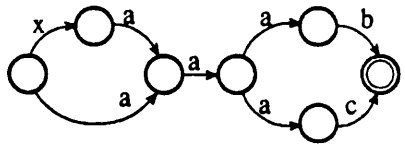
y de la segunda asunción:

C.2: Si en α y β hay dos subcadenas similares, no es posible que no hayan sido generadas por las mismas reglas (si están en la misma posición relativa).



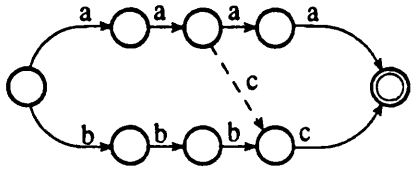
Incumple la *segunda* asunción (corolario C.2):

Si R_+={"xaaab","aaac"} esta gramática
NO es inferible por ECGI: cadenas parecidas *deben* tener reglas comunes.



Incumple la *segunda* asunción (corolario C.2):

Si R_+={"xaaab","aaac"} esta gramática
NO es inferible por ECGI: Hay subcadenas comunes que se generan con reglas diferentes para cada cadena.



Incumple la *primera* asunción:

Si lo que está en trazo continuo es la gramática actual, la regla punteada NO es inferible por ECGI al presentarse la cadena "aacc": todas las reglas que se identifican deben pertenecer a la cadena más próxima: "aaaa".

Lo que inferiría ECGI en el último caso:

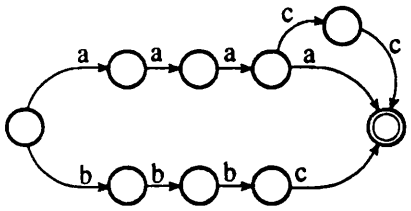
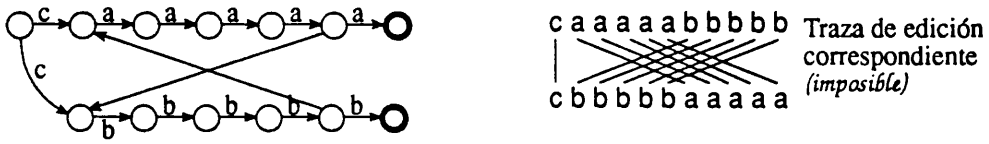


Figura 6.5 Gramáticas (autómatas) no inferibles por ECGI al incumplir alguna de sus asunciones.

La segunda asunción (se *identifican* –pertenecen a YEST– las reglas que generan subcadenas de β *similares* a las de α), implícitamente supone que es posible determinar qué reglas han generado una cadena muestra con sólo disponer de dicha cadena, hipotetizando que la similitud entre cadenas (en un sentido a definir) es signo de que han sido generadas (por lo menos en parte) por las mismas reglas (lo cual hoy en día es el heurístico base de toda la inferencia gramatical). En el caso de ECGI la similitud queda definida por el algoritmo de corrección de errores que se aplica en la fase de comparación, el cual, no sólo justifica el nombre de ECGI, sino que permite variar el comportamiento del método en función de varias posibles definiciones alternativas de dicho algoritmo de corrección (ver apartado 6.6). Por otra parte, es fácil comprobar que la segunda asunción, en su exigencia de conservar la ordenación relativa de las subcadenas, es también la responsable de que ECGI no pueda generar circuitos (ver figura 6.6).

R_+={"caaaaabbbbb","cbbbbbaaaaa"}

Gramática imposible de generar por ECGI (viola restricción de posición relativa):



Gramática generada por ECGI en este caso:

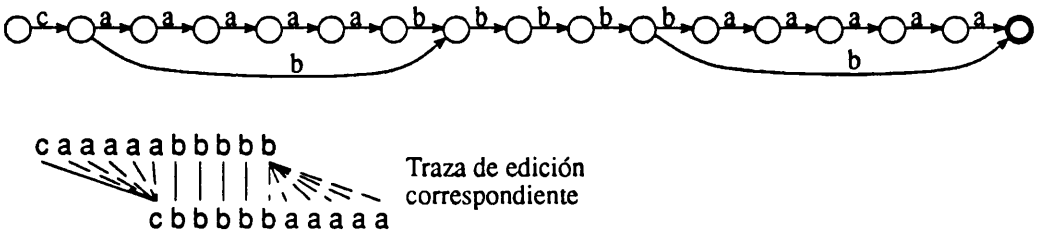


Figura 6.6 Identificación imposible para ECGI (por restricción de posición relativa) y la correspondiente permitida. Autómatas inferidos en cada caso.

La segunda asunción es la que determina el poder de generalización de ECGI. La primera asunción restringe mucho esta generalización, y el corolario C.1 (ninguna regla, fuera de las que generan la cadena más próxima β, puede haber participado en la generación de la muestra α) provoca a veces que ECGI no generalize cuando podría haberlo hecho. Por otro lado, esta suposición es también la mayor responsable de que la gramática inferida por ECGI (y su lenguaje) dependa del orden de presentación de las muestras (ver figura 6.7).

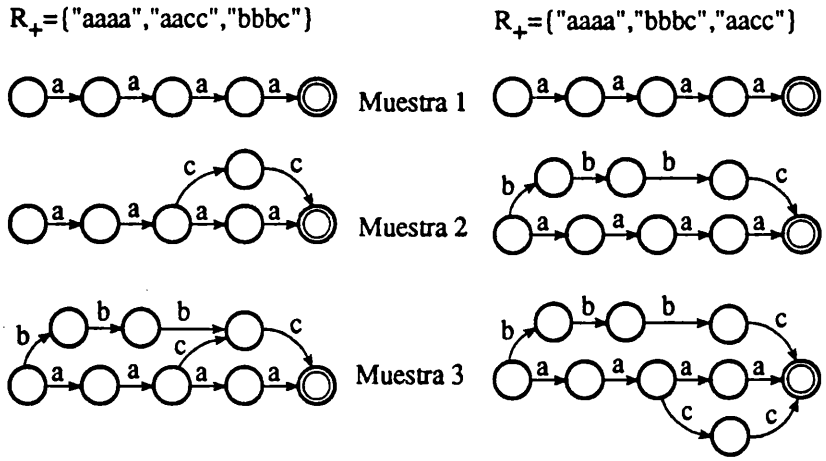


Figura 6.7 ECGI infiriendo dos gramáticas diferentes para un mismo conjunto de muestras ordenado de dos maneras distintas.

Recientemente, en [Miralles,91] se ha propuesto cambiar las asunciones de ECGI, autorizándole a identificar subcadenas que no pertenencen a la cadena más similar β, y que ni siquiera *pertenecen a una misma cadena del lenguaje* de la gramática actual. Para ello es necesario cambiar el modelo de

error, definiendo reglas de error capaces de saltar de una cadena a otra completamente distinta del lenguaje, eso sí, siempre teniendo en cuenta la ordenación de estados (figura 6.8). Los experimentos llevados a cabo indican una disminución en el tamaño de los modelos (gramáticas) inferidos, así como un (muy leve) empeoramiento de resultados de reconocimiento con respecto al modelo aquí presentado,

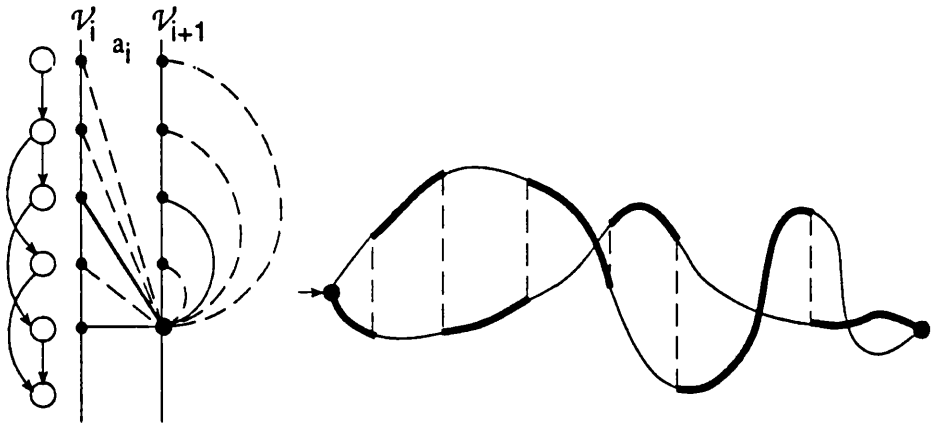


Figura 6.8 Propuesta para aumentar las capacidades de identificación de ECGI [Miralles,91]. Reglas de error suplementarias (trazo punteado) que ello implica, con respecto a las de error usualmente utilizadas (trazo fino) para una regla de no error (trazo grueso). Las reglas se dibujan en dos etapas del trellis, correspondientes al análisis del carácter i-ésimo a_i de la cadena a analizar, por el autómata de 6 estados representado a la izquierda. Derivaciones (trazo grueso) con "saltos" (trazo punteado) que ello autoriza, mostradas en un autómata en el que sólo se representan las transiciones (trazo fino).

6.5.2 No se añaden las reglas de error

Utilizando la representación gráfica, es posible comprobar inmediatamente el porqué no se añaden sin más en la gramática las reglas error (ni derivaciones directas de ellas): ello provocaría la aparición de bucles (reglas de inserción), de reglas con el símbolo nil (reglas de borrado) o de secuencias de ellas, así como de una no separación de caminos diferentes (no se separarían los no terminales correspondientes a secuencias de reglas que generan subcadenas completamente distintas), todo lo cual se puede observar en la figura 6.9.

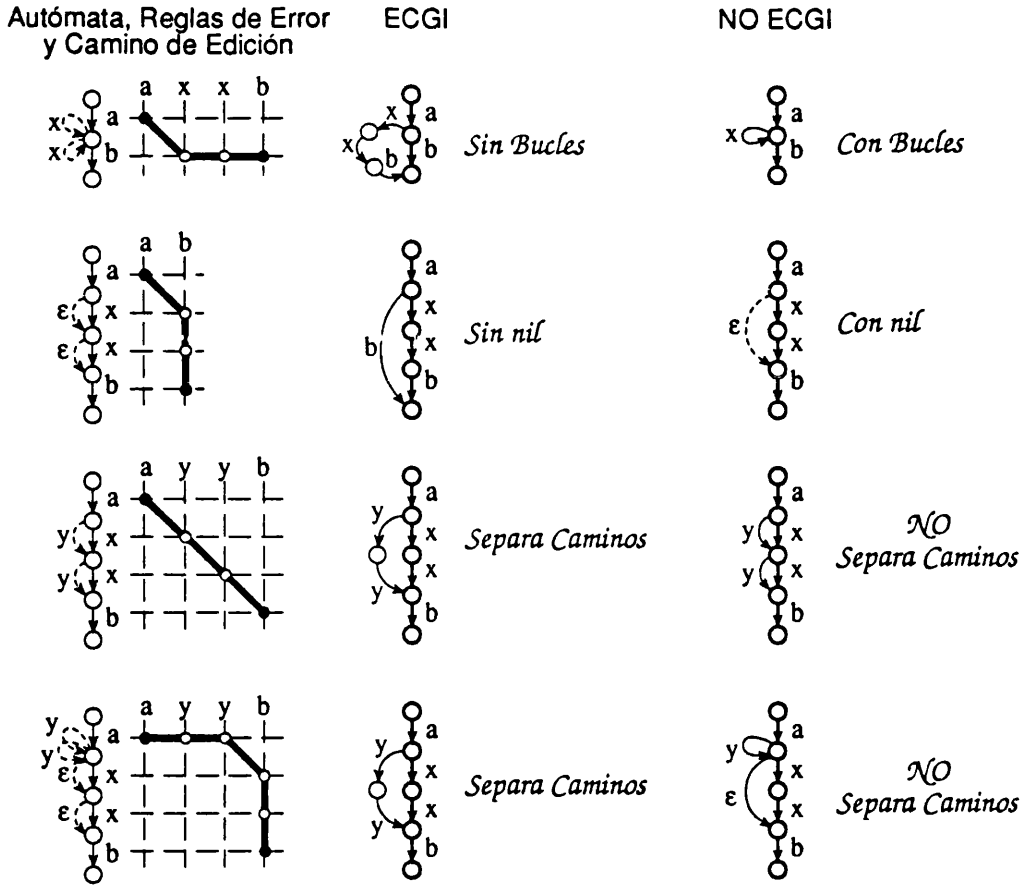


Figura 6.9 Casos típicos de derivación. Se muestra lo que hace ECGI y la alternativa más inmediata (añadir las reglas de error o similares). El caso d) comparado con el c) muestra la dependencia del algoritmo de construcción del de comparación y cómo ECGI la evita en algunos casos.

Se vuelve a comprobar pues que una gran parte de la carga heurística de ECGI se halla en el mismo algoritmo de construcción, en el cual se han tomado una serie de decisiones "razonables" a la hora de escoger qué reglas son convenientes y qué reglas no lo son. En conclusión: *ECGI está específicamente diseñado para no generar ni bucles ni reglas que empleen el símbolo nil.*

6.5.3 Heurísticos no esenciales

Dada la naturaleza no caracterizable, y por lo tanto eminentemente práctica de ECGI, es conveniente examinar hasta qué punto su proceso de inferencia es adecuado (en el sentido de que infiere estructuras "intuitivamente" correctas y adecuadas a los problemas a resolver), teniendo en cuenta las restricciones impuestas por el método y los heurísticos de construcción. Este examen además nos permitirá tomar conciencia de otro conjunto de heurísticos implícitos en ECGI.

Para el análisis se irán considerando distintos problemas de inferencia en orden creciente de complejidad, estudiándose el comportamiento de ECGI en cada caso. En cada problema se examinará la derivación óptima, analizando el camino o la traza de edición entre la cadena más próxima y la nueva cadena, y comprobando si las reglas que esta derivación induce en la construción son o no adecuadas.

6.5.3.1 Prioridad a la substitución

En el caso más sencillo, las dos cadenas están formadas por la misma secuencia de terminales diferentes, pero que se repiten más o menos en una u otra cadena. En la figura 6.10 se muestra un ejemplo, junto con la traza de edición y la construcción correspondiente por parte de ECGI.

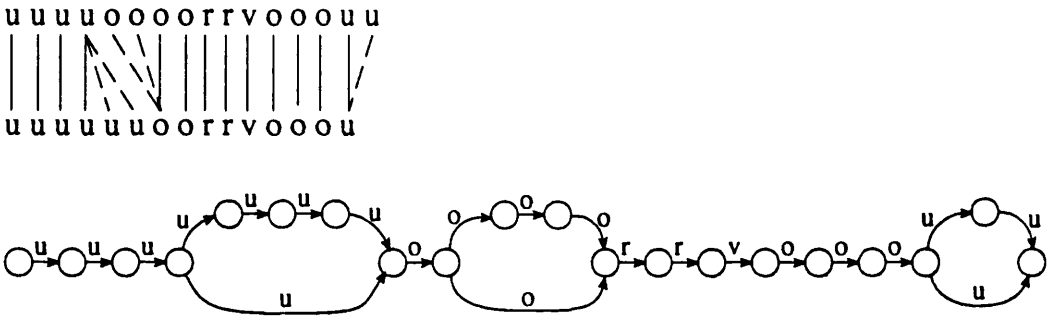


Figura 6.10 Traza de edición y construcción en el caso más sencillo.

Obsérvese la *simetría entre la inserción y el borrado* que, en buena lógica, siempre debe de estar presente: debe obtenerse el mismo lenguaje cuando primero se da la cadena con muchos símbolos y luego se borra, que cuando se primero se da la cadena con pocos símbolos y se inserta.

Por otra parte, en la figura 6.11 se comprueba que, aún en este caso sencillo, la derivación óptima queda indefinida: hay más de una derivación que conduce al mismo número de errores. Se representan cuatro posibles caminos de edición (en trazo grueso en el trellis) para dos cadenas: "uuuu" y "uu":

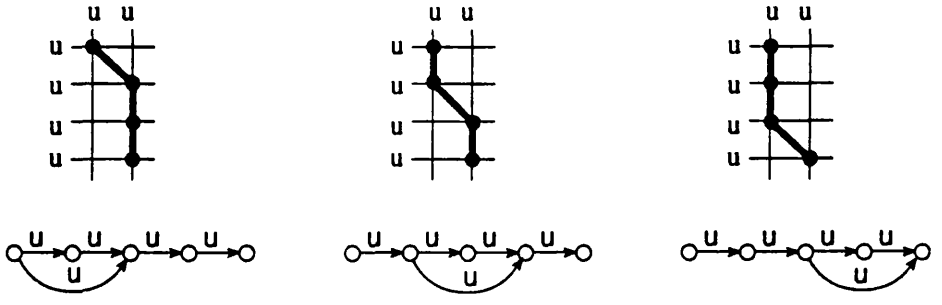


Figura 6.11 Varios posibles caminos de edición posibles entre las cadena "uu" y "uuuu", todos ellos equivalentes para la comparación , pero no para la construcción.

Una buena definición del algoritmo de construcción debe llevar a la misma estructura (la antes presentada) para todos estos posibles casos. Sin embargo, el algoritmo de construcción de ECGI, tal como se ha descrito, podría generar en cada caso un conjunto distinto de reglas (la inserción o borrado se produciría en un lugar distinto de la cadena más próxima). Una posible aproximación, para obviar el problema, es obligar de alguna manera a que el algoritmo encuentre siempre uno solo de estos tipos de derivación, no pudiendo presentarse nunca ninguno de los demás. Ello se consigue aplicando la optimización en un determinado orden, para que cuando haya que escoger entre varias decisiones de igual coste, se escoja siempre la primera (por ejemplo) que se compara. Esta es la aproximación de ECGI, implícita en el algoritmo ViterbiCorrector tal como se ha presentado. Por lo tanto: *ECGI, en caso de que haya multiplicidad de derivaciones con el mismo coste, da prioridad, entre los otros errores, a la sustitución.* Si hay más de una sustitución equivalente, escoge la última en el orden de análisis. Esto lleva, en el caso general, a insertar en el primero de los posible lugares de inserción en la cadena más próxima (ver figura 6.12).

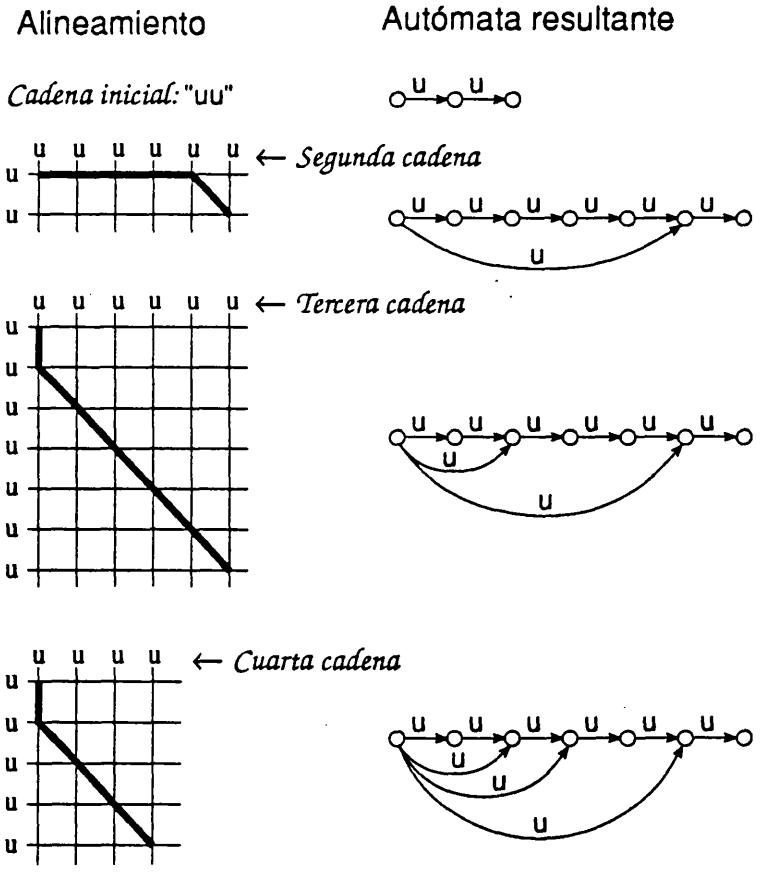


Figura 6.12 Ejemplo de inferencia de ECGI, en el que se ve que priorizar la substitución (diagonal del camino) al tomar las decisiones, conduce siempre a insertar y borrar en el mismo lugar. Ello no ocurriría si no se priorizara alguno de los errores, pues cualquier camino sería posible en rejillas como las presentadas. Nótese que el análisis del camino procede de atrás hacia delante.

6.5.3.2 Inserción en paralelo

El caso de mayor dificultad ocurre cuando en un segmento de la cadena se han suprimido algunos símbolos y se han insertado otros nuevos. Ello produce una ambigüedad sobre el lugar a insertar la nueva subcadena: ¿antes, después, en medio, o en paralelo con la subcadena que se ha borrado? (ver figura 6.13).

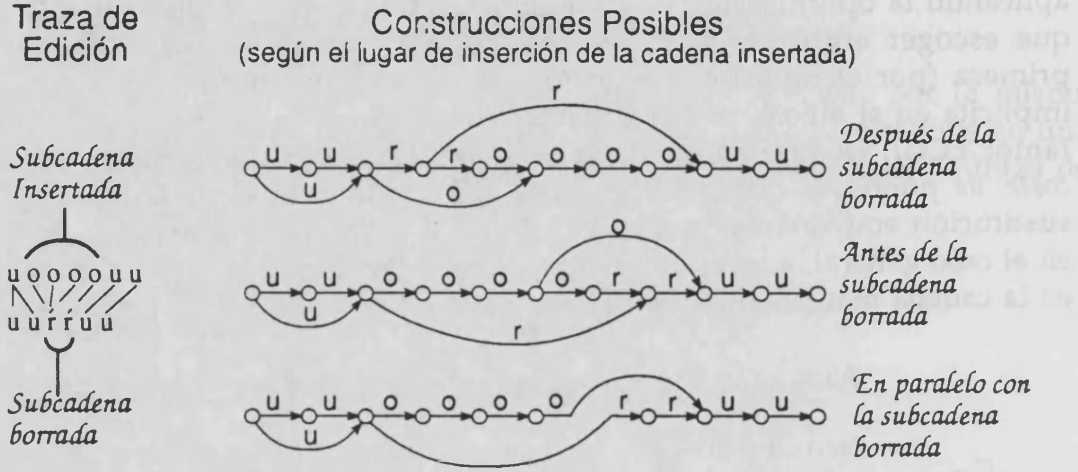


Figura 6.13 Ambigüedad fundamental cuando hay inserción y borrado simultáneos.

La primera impresión es que la solución más adecuada es realizar la inserción en paralelo. Esto es lo que hace ECGI, aunque a menudo le pueda llevar a generar una gran cantidad de estados innecesarios (ver figura 6.14). En efecto, la inserción "en paralelo" implica que ésta se hace de manera que las subcadenas borrada e insertada no puedan pertenecer a la misma cadena del lenguaje de la gramática inferida (y por lo tanto supone que las dos subcadenas nunca aparecerán a la vez en alguna de las siguientes cadenas muestra).

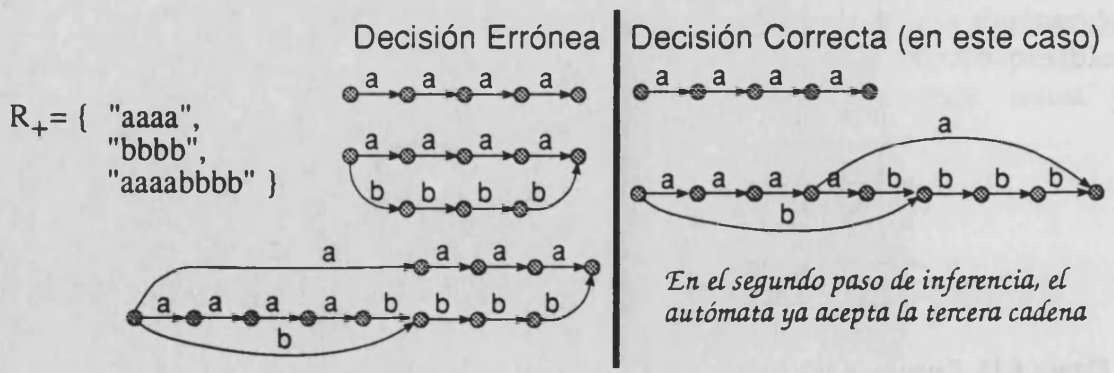


Figura 6.14 Decisión errónea tomada por ECGI cuando se le presentan tres cadenas en un determinado orden.

No existe una solución definitiva a este problema, aunque un paliativo podrían consistir en presentar las cadenas en orden decreciente de longitud.

Por otro lado, si en un caso concreto, la solución "en paralelo" resulta ser extremadamente inadecuada, no sería muy complicado modificar el algoritmo de construcción de ECGI para obtener una inserción después (o antes), .

En resumen: En caso de borrado e inserción simúltáneos en un punto, *ECGI procede a la inserción de la nueva subcadena en paralelo con la borrada.*

6.5.3.3 Otras consideraciones

En general, ECGI, al realizar el análisis de una derivación óptima se encontrará con una serie de casos de borrado, inserción y substitución de una subcadena. En lo que sigue nos centraremos en los casos de borrado, ya que los casos de inserción son simétricos a éstos, y el caso de substitución de una cadena por otra ya se ha visto en el apartado 6.5.3.2. Siempre que se produce un borrado, la subcadena borrada ω se encontrará entre entre otras dos subcadenas φ_1 y φ_2. Los distintos casos con los que se encontrará el algoritmo de construcción se definen en función de la igualdad o no de los terminales que componen ω, φ_1 y φ_2. Los casos posibles serán entonces $\varphi_1\omega\varphi_2$ = uuu, unu, urr, y unr; donde cada trío define los símbolos que componen la subcadena correspondiente (p.e.: unu quiere decir que entre dos subcadenas compuestas por terminales u, se borra una subcadena compuesta por terminales distintos n). Entonces, el caso uuu (borrado de una subcadena entre dos subcadenas compuestas por los mismos que terminales que ella) es el caso sencillo, considerado en el apartado 6.5.3.1, y los otros tres casos restantes son los que se muestran en la figura 6.15, junto con la decisión que tomará la etapa de construcción de ECGI en cada uno de ellos, a la hora de decidir qué reglas debe añadir.

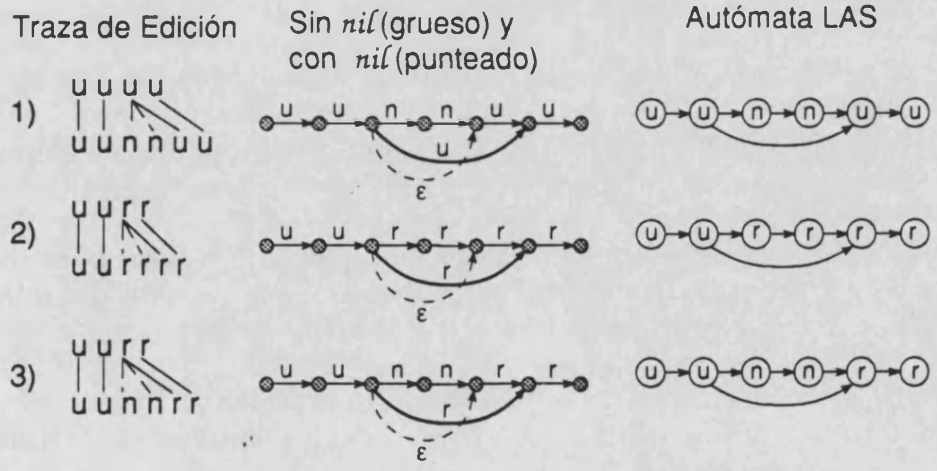


Figura 6.15 Tres casos de borrado de subcadenas y las correspondientes reglas generadas por ECGI. Se muestra la traza de edición en cada caso, el autómata generado por ECGI, el que generaría si autorizara símbolos "nil" y el autómata generado por la versión de ECGI que genera autómatas LAS.

Aunque correctas, las decisiones tomadas en los casos 1) y 3) son inadecuadas si se pretende realizar a posteriori algún tipo de segmentación sobre los autómatas inferidos por ECGI (p.e.: inferir gramáticas que describen palabras, e intentar luego simplificar y extraer subautómatas que describen fonemas). En efecto, tanto en 1) como en 3) las reglas que generan φ_1 y φ_2 se han visto "mezcladas" con las que generan ω, y por lo tanto resultan difícilmente separables si se intenta particionar el autómata. Ello es debido, como se puede observar en la misma figura, a la negativa por parte de ECGI a generar transiciones con el símbolo "nil". En la implementación práctica, el inconveniente se ha evitado, debido a que ECGI infiere autómatas de estados etiquetados (LAS: ver capítulo 2). En estas condiciones, como se puede comprobar en la última serie de autómatas presentados en la figura 6.15, ya no se produce ninguna "mezcla" de s[illegible]adenas.

6.6 (Di)similitudes

Aunque nada obligue a ello, en todas las implementaciones actuales de ECGI se ha recurrido a un único modelo de error: el descrito en el capítulo 2 para gramáticas regulares y que autoriza la inserción, borrado y/o sustitución de un símbolo en cualquier lugar de la cadena. Como también se vio en el mismo capítulo, es posible definir a partir de este modelo de error, múltiples criterios de (di)similitud entre cadenas, similares todos ellos, pero que a la hora de un análisis sintáctico corrector de errores conducen a derivaciones óptimas que pueden ser muy diferentes. Por otra parte:

- No todos estos criterios son minimizables óptimamente mediante programación dinámica (ViterbiCorrector, capítulo 5), aún restrinjiéndose a las definibles a partir de un modelo de error tan sencillo como el utilizado.

- El coste de la optimización (búsqueda de la derivación óptima) puede depender fuertemente del criterio utilizado (el coste de ViterbiCorrector es $O(|Q|\cdot n\cdot B)$, siendo la operación elemental el cálculo de un elemento de (di)similitud y su comparación).

Tres son los criterios que se han estudiado en este trabajo para definir la (di)similitud entre una cadena α y la gramática G, en función del número de reglas de error y no error de la derivación óptima:

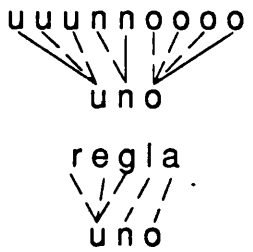
minE: similitud(α,G) = Número de reglas de error.
minEL: similitud(α,G) = Número de reglas de error / Número de reglas utilizadas.
maxA: disimilitud(α,G) = Número de reglas de **no** error (de *aciertos*).

6.6.1 Menos errores.

El criterio minE es el más intuitiva: una cadena se parece tanto más a otra cuantos menos modificaciones (errores) son necesarios para pasar de una a otra. Como se mencionó en el capítulo 2, este criterio es el que comúnmente se utiliza en corrección de errores, con el nombre de distancia de Levensthein, y es el que se utilizó en las primeras versiones de ECGI

6.6.2 Relativamente menos errores

El criterio minEL pretende mejorar el anterior, teniendo en cuenta las características del habla (la "elasticidad" de ciertos sonidos), aunque resulta ser una mejora aplicable a muchos otros problemas. Viene a decir: lo que importa es número *relativo* de errores con respecto a la longitud de la derivación. Ello puede resultar adecuado, o no, según la aplicación considerada (ver figura 6.16). El criterio minEL se utilizó en todos los experimentos de ECGI, excepto en los últimos correspondientes al reconocimiento de imágenes.



6 inserciones +
3 no errores ⇒ minEL = 6/9 = 0.67 ; minE = 6

5 errores ⇒ minEL = 5/5 = 1 ; minE = 5

Figura 6.16 Comparación de la cadena "uno" con otras dos cadenas utilizando normalización de longitud y sin utilizarla. Obsérvese como una cadena mucho más larga es a pesar de todo la más próxima si se utiliza la normalización.

El inconveniente mayor que presenta un criterio con *normalización por la longitud de la derivación* es debido a que los algoritmos de programación dinámica expuestos hasta ahora sólo permiten minimizarlo de manera **subóptima** (aunque suficiente para que ECGI dé buenos resultados) (ver figura 6.17).

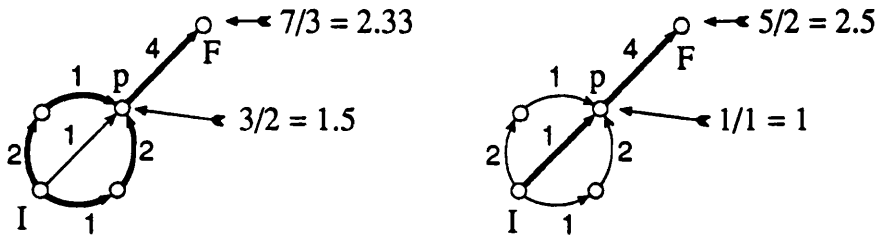


Figura 6.17 Problema de camino mínimo con coste minEL entre los vértices I,F de un grafo dirigido y ponderado. Solución óptima (izquierda) y solución subóptima (derecha) encontrada por programación dinámica. El trazo grueso señala el camino óptimo escogido en cada caso; las flechas indican el coste acumulado en los vértices señalados. Obsérvese que la PD, al tomar una decisión local en el punto P, escoge la solución equivocada (1/1 en vez de 3/2, lo que lleva a 2.5 en vez de 2.33 en el coste final).

Recientemente se ha propuesto un algoritmo [Marzal,91a] que permite obtener la derivación óptima. Lamentablemente, la optimalidad se consigue a costa de un aumento considerable de complejidad: se pasa de $O(|X|\cdot|Y|)$ temporal y $O(\min(|X|,|Y|))$ espacial a $O(|X|\cdot|Y|\cdot\min(|X|,|Y|))$ y $O(\min(|X|,|Y|^2)$ respectivamente (el algorimo se aplicó a similitudes entre cadenas, no habiéndose extendido aún a similitudes gramática-cadena). Bien es cierto que, aún utilizando un algoritmo clásico de PD (y por lo tanto subóptimo para este caso), este criterio es más costoso de evaluar, tanto temporalmente (requiere una normalización en cada comparación en el trellis) como espacialmente (es necesario anotarse también la longitud acumulada por cada camino en el trellis).

6.6.3 Más aciertos

El criterio maxA se deriva de la observación de que la etapa de construcción de ECGI se basa única y exclusivamente en las reglas **no** erróneas. ECGI genera tantos menos no terminales y reglas, cuantas más reglas *no* erróneas haya en la derivación (para una longitud de muestra dada), puesto que añade una regla por cada símbolo de la muestra que no ha podido ser generado por una regla de no error. En teoría, pues, el criterio maxA debe llevar a ECGI a generar menos reglas, y ello independientemente del problema considerado. Es el criterio que se utilizó en la tarea de reconocimiento de imágenes.

Sin embargo, a pesar de que en principio genere menos estados, la maximización de aciertos no impide que, en algún caso, se produzcan efectos indeseables que una normalización por la longitud de la derivación, o una minimnización de errores, hubieran evitado (ver figura 6.18).

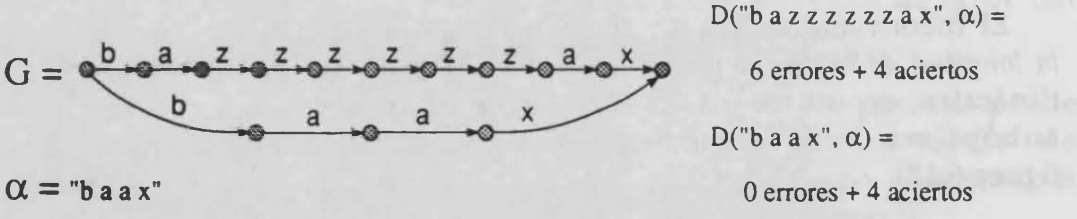


Figura 6.18 Comparación de una cadena con una gramática. Obsérvese que si sólo se maximizan aciertos ambas derivaciones son equivalentes, siendo obviamente una de ellas inadecuada.

6.6.4 Relación entre criterios

Intuitivamente se plantea de inmediato la duda de si las distancias expuestas anteriormente son o no equivalentes; especialmente, si no será lo mismo minimizar el número de errores que maximizar el número de

aciertos. En este apartado examinamos la relación formal existente entre estos y otros criterios.

Sea L, E, A respectivamente el número de reglas, el número de reglas de error y el número de reglas de no error de una derivación. Los criterios a considerar se obtienen evaluando (d es una derivación cualquiera):

$$\text{minE} = \min_{\forall d} (E_d)$$

$$\text{minEL} = \min_{\forall d} \left(\frac{E_d}{L_d}\right)$$

$$\text{maxA} = \max_{\forall d} (A_d)$$

Comparando el criterio minE con el maxA, se comprueba inmediatamente que **no es lo mismo minimizar errores que maximizar aciertos**. En efecto, para toda derivación, se cumple L=E+A, y para dos derivaciones distintas 1 y 2:

$$E_1 < E_2 \Rightarrow L_1\text{-}A_1 < L_2\text{-}A_2 \Rightarrow L_1\text{-}L_2 + A_2 < A_1$$

de lo cual, es imposible deducir $A_1 > A_2$, puesto que L_1 y L_2 son cualesquiera (ver figuras 6.18 y 6.19).

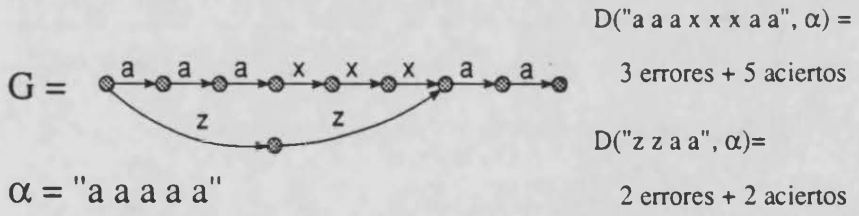


Figura 6.19 Comparación de una cadena con un autómata. Obsérvese como según se minimizen errores o se maximizen aciertos se escoge una derivación distinta como óptima.

Por otro lado, suponiendo que se dispone de un algoritmo para evaluar de manera óptima minEL, es fácil comprobar que **si se normaliza por la longitud del camino, es lo mismo minimizar errores que maximizar aciertos,** ya que:

$$\frac{E_1}{L_1} < \frac{E_2}{L_2} \Rightarrow \frac{L_1\text{-}A_1}{L_1} < \frac{L_2\text{-}A_2}{L_2} \Rightarrow 1 - \frac{A_1}{L_1} < 1 - \frac{A_2}{L_2} \Rightarrow \frac{A_1}{L_1} > \frac{A_2}{L_2}$$

Y análogamente:

$$\frac{A_1}{L_1} > \frac{A_2}{L_2} \Rightarrow \frac{E_1}{L_1} < \frac{E_2}{L_2}$$

Lo que hace innecesario definir y experimentar el criterio:

$$D=\max_{\forall d}\left(\frac{A_d}{L_d}\right)$$

6.6.5 Nadie es perfecto

Debe observarse que ninguno de los criterios definidos en los apartados anteriores lleva a cabo una alineación perfecta de las cadenas (en el sentido más intuitivo), si se asume la "elasticidad" de las subcadenas. En concreto, todos estos criterios asumen que resulta igual de costoso insertar (borrar, por simetría) el mismo símbolo ya presente que uno completamenta distinto, tal como se comprueba en la figura 6.20. Asimismo, se observa en la misma figura que, ECGI, en el caso más intuitivo, agruparía en subredes los segmentos más parecidos, al contrario de lo que hace la presente versión. Sin embargo como también se ve, ello conllevaría un incremento en el número de reglas añadidas.

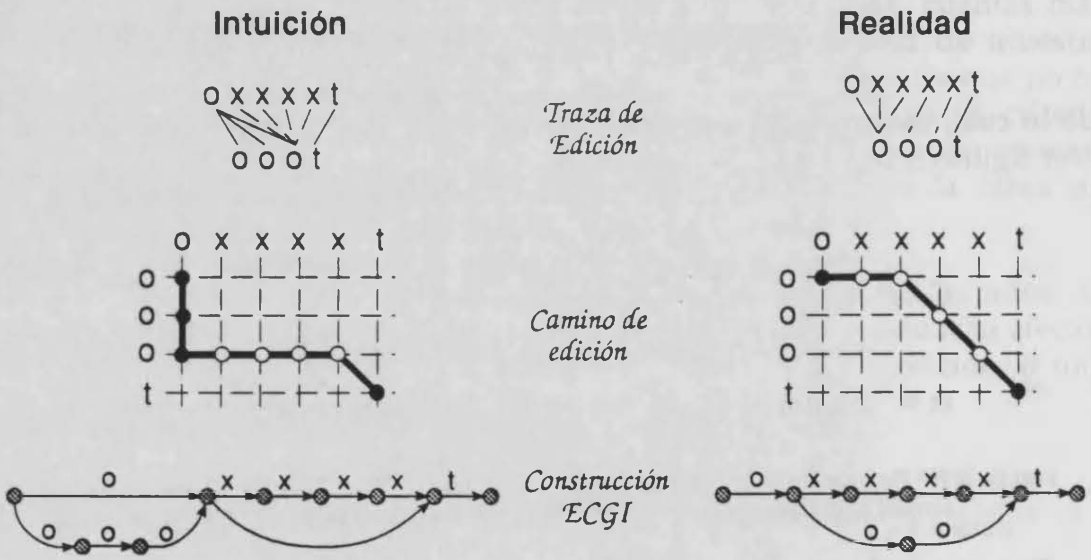


Figura 6.20 Ejemplo de caminos de edición entre dos cadenas que contradice la idea "intuitiva" cuando hay "elasticidad" de subcadenas. Nótese que la construcción mostrada para ECGI en el caso intuitivo **no** es la que haría el método ECGI actual, que generaría un autómata idéntico al del caso real

Se concluye, pues, que un criterio concreto no resuelve todos los posibles problemas que se presenten a ECGI. De hecho, la definición del criterio de (di)similitud es el punto donde más fácilmente se puede ajustar ECGI a un problema particular. Es de notar, sin embargo, que los criterios utilizados se encuentran entre los más sencillos, y que toda mejora supondrá probablemente un incremento de complejidad.

Finalmente, para evaluar los tres criterios de (di)similitud aquí considerados, desde el punto de vista de su eficacia y adecuación *para ECGI*, se han realizado una serie de experimentos de reconocimiento de formas, que se presentan en el capítulo 9.

6.7 Reconocimiento

Las gramáticas, se hayan obtenido o no mediante un método de inferencia a partir de un conjunto de muestras, pueden aplicarse de diversas maneras al reconocimiento de formas (modelización de conocimientos linguísticos, traducción, clasificación, etc...). De entre todas estas maneras, la más sencilla consiste en emplear las gramáticas como clasificadores.

Toda gramática representa (genera) un lenguaje, y siempre existe un reconocedor de las cadenas de ese lenguaje asociado a la gramatica. En los casos más sencillos, las gramáticas regulares, ese reconocedor es un autómata finito. Un autómata se comporta realmente como un reconocedor de formas: dado un objeto (una cadena) es capaz de decidir si ese objeto pertenece o no a la forma que representa: el lenguaje de la gramática. Nada impide utilizar las gramáticas inferidas por ECGI aplicando esta idea, reduciéndose entonces la operación de reconocimiento a un análisis sintáctico, con una decisión final si/no de pertenencia al lenguaje. Sin embargo, y como ya se ha mencionado en el capítulo 2, esta aproximación, aunque sencilla, es poco factible en la práctica.

La razón de esta infactibilidad se halla cuando se constata que las gramáticas inferidas por ECGI, en un caso real, no modelizan la totalidad del lenguaje que tienen la misión de reconocer. Ello es debido, principalmente, a que la presentación de ejemplos nunca puede ser completa. En concreto, en las tareas mínimas de reconocimiento de formas planteadas a ECGI, existen dos fuentes principales de variabilidad en las cadenas del lenguaje buscado, que limitan seriamente la completitud del conjunto de muestras R_+:

- El "ruido", que altera las cadenas introduciendo una serie de errores más o menos aleatorios. Estos errores inducen una variabilidad casi ilimitada, que impide toda convergencia final del aprendizaje, al ser muy alta la probabilidad de que aparezca un error, aún no modelizado por no hallarse presente en R_+.

- Las variaciones de longitud de las subestructuras, que es casi imposible que se den en toda su variedad en R_+.

Es pues obvio que la inferencia debe detenerse en un punto razonable, en el que las principales características estructurales de R_+ han sido registradas, junto con los errores (debidos o no al ruido) más frecuentes y las variaciones de longitud más usuales. Un criterio razonable de detención de la inferencia en estos casos podría ser esperar a que la distancia promedio a la gramática de las últimas (n) muestras baje por debajo de un umbral[4].

---

[4] En la práctica, debido a la perenne escasez de muestras que sufren los estudiosos del reconocimiento de formas, el aprendizaje se detienen cuando ya no quedan más muestras (nótese además que en este caso las muestras no pueden ser reutilizadas en un proceso de reestimación).

Por otra parte, si se detiene el aprendizaje antes de que la gramática inferida genere la totalidad del lenguaje buscado, es evidente que es imposible utilizar sin más dicha gramática (su autómata) en reconocimiento, pues muchas cadenas que deberían ser aceptadas no pertenecen al lenguaje inferido. En reconocimiento sintáctico de formas, la solución que usualmente se da a esta dificultad, tal como se mencionó en el capítulo 2, consiste en extender la gramática inferida recurriendo a métodos de *análisis sintáctico corrector de errores*. Esta aproximación cuenta con la ventaja de que la gramática inferida sólo tendrá que dar cuenta de las características estructurales más importantes, y como mucho, de los errores más probables, puesto que el modelo de de error (el que se emplea en reconocimiento, no confundirlo con el que emplea e ECGI en inferencia) da cuenta de los errores y/o variaciones menores.

Concretando todas estas consideraciones:

1) En los experimentos llevados a cabo para este trabajo, se han utilizado las gramáticas inferidas por e ECGI como *clasificadores* de dígitos (hablados, manuscritos e impresos) o letras (habladas).

2) Cada forma (dígito) se ha representado mediante una única gramática, inferida por ECGI.

3) El reconocimiento se efectúa mediante un *análisis sintáctico corrector de errores* de la muestra por cada una de las gramáticas inferidas, lo que proporciona un conjunto de (di)similitudes muestra-gramática. Para la decisión, se recurre a la regla del vecino más próximo (NN) [Duda,73], en su versión más sencilla: se escoge aquella clase cuya distancia a la muestra sea mínima (o cuyo parecido sea máximo).

Por otra parte, como se verá en el capítulo 7 ("extensión estocástica del ECGI"), es posible almacenar la información estadística asociada a la frecuencia de utilización de las reglas, de manera que las gramáticas inferidas por ECGI sean estocásticas. En este caso, el análisis sintáctico corrector de errores será también estocástico, y para la decisión se aplicará la regla de *máxima verosimilitud* (basada en la regla de Bayes [Duda,73]): se escoge la clase que maximiza la probabilidad de pertenencia de la cadena a la gramática.

6.8 Aprendizaje continuo

Dada la naturaleza absolutamente incremental del proceso de aprendizaje de ECGI, es perfectamente factible implementarlo de manera que simultánee el reconocimiento de la nueva muestra y su aprendizaje,

permitiendo la evolución del modelo inferido aún cuando ya está en plena utilización.

Por otro lado, a cada nueva cadena muestra, el primer paso del algoritmo de inferencia de ECGI consiste en obtener la derivación óptima de la muestra con respecto a la gramática actual. Nada impide que el análisis sintáctico corrector de errores, que supone el buscar esta derivación, proporcione al mismo tiempo una (di)similitud muestra-gramática utilizable para el reconocimiento de la misma. Ello autoriza una integración de los procesos de inferencia y reconocimiento, que no sólo resulta conceptualmente atractiva, sino que permitiría reducir fuertemente el coste computacional en caso de utilizarse ECGI en una aplicación que requiera aprendizaje continuo.

El siguiente algoritmo ilustra este proceso continuo de aprendizaje y la posible reducción de coste computacional mediante integración inferencia-reconocimiento. En él, un operador solicita la realización de una tarea mediante presentación de una forma representada por la cadena α. ECGI efectúa un reconocimiento, presenta la forma reconocida y solicita confirmación. Si la respuesta proporcionada por el operador es positiva, ECGI efectúa la tarea solicitada e integra la nueva cadena a la gramática inferida para la clase reconocida. Si la respuesta es negativa, se solicita al operador cuál es la clase correcta (mediante un medio más seguro), y se actualiza la gramática de esa clase, obviamente, después de haber de realizado la tarea solicitada. El proceso es en principio indefinido, y se detiene sólo cuando el operador ya no tenga más tareas que realizar:

```
Algoritmo Integración
Datos    M                  /* número de clases a reconocer */
         G_oi i=1..M        /* M gramáticas iniciales */
Método
  hacer indefinidamente
     Aceptar una nueva muestra α
     /* Análisis sintáctico corrector de errores */
     Obtener  D(α,G_i) y              /* Derivación óptima y */
              dist(α,G_i) i=1...M     /* disimilitud α,G_i */
     Postular clase de α:=k=argmin (dist(α,G_i))
                                 i=1..M
     Solicitar confirmación  /* al operador */
     si correcto
     entonces
        Realizar Tarea(k)
        construcción(G_k,D(α,G_k))
     sino
        Solicitar t=clase correcta
        Realizar Tarea(t)
        construcción(G_t,D(α,G_t))
     fin si
  fin hacer
fin Integración
```

En esta óptica, se entrena inicialmente el sistema -por ejemplo, utilizando el mismo algoritmo, pero sin efectuar el paso "`Realizar Tarea`" las N primeras veces (N reducido≈30)- con relativamente pocas muestras, aprovechando la rápida convergencia inicial de ECGI. Seguidamente se entra en fase de producción, dejando que el refinamiento posterior, mucho más lento, se haga durante la utilización. Esto no sería excesivamente molesto para el usuario (la tasa de reconocimiento fácilmente superaría el 80% en un principio), con la ventaja de disponer en todo momento de una adaptación a las circunstancias (cambio de locutor, de útil de escritura, catarro, etc.).

Dos observacions adicionales, en caso de plantearse una aproximación mediante `Integración`:

- Sería imprescindible una simplificación periódica de los autómatas (gramáticas) inferidos, para evitar un crecimiento indefinido (ver capítulo 10).
- El modelo de corrección de errores y el algoritmo de análisis sintáctico deberían ser el mismo en comparación y reconocimiento; en caso contrario se pierde la ventaja computacional, aunque se puede seguir aprovechando la capacidad de aprendizaje incremental de ECGI.

6.9 Sólo substituciones

Se ha subrayado que ECGI (lo mismo que cualquier algoritmo de inferencia gramatical cuando trata muestras ruidosas y distorsionadas), infiere no sólo la gramática buscada, sino también un modelo de los errores más frecuentes (probables) que se producen en las muestras. El papel del modelo de error aplicado en reconocimiento es únicamente el de dar cuenta de aquellos errores imposibles de inferir por su poco significado estructural y su infinita variedad.

Por otro lado, si no se opta por una aproximación integradora de la inferencia y el reconocimiento, es posible aplicar modelos de error totalmente distintos en aprendizaje y reconocimiento. Dado que en reconocimiento suele requerirse un mínimo coste computacional, cabe pensar en utilizar durante esta fase un modelo de error menos costoso. Si se recuerda que el coste del algoritmo de reconocimiento es estrictamente el de ViterbiCorrector, $O(n \cdot |Q| \cdot B)$, con B dependiendo directamente del número de reglas aplicables a cada no terminal, una reducción drástica en el número de las reglas de la gramática expandida se obtiene **autorizando únicamente errores de sustitución**. El número de reglas por cada regla de la gramática no expandida se reduce en más de la mitad (de $1+2 \cdot |V|$ a $|V|$).

Pero la ventaja computacional que se obtiene, autorizando únicamente errores de substitución, no sólo es debida a la reducción en el número de reglas de la gramática expandida. Desarrollos recientes de ECGI [Torró,90] [Alfonso,91], que utilizan técnicas subóptimas para reducir drásticamente (a menos de un 10% de la original) la complejidad temporal de un reconocimiento, sólo son posibles si no se autorizan las reglas de borrado.

En la práctica, autorizar tan sólo sustituciones al efectuar la búsqueda de la cadena más similar en la gramática, equivale a suprimir toda posibilidad de estiramiento o compresión de las subcadenas de la muestra con respecto a las muestras de aprendizaje. Puesto que se está suponiendo que la gramática inferida modeliza suficientemente los errores más probables que se producen en las muestras, se espera que esto no resulte una penalización excesiva para la eficacia del reconocimiento; esperanza que se ve confirmada por la experiencia, como demuestran los resultados resumidos en el capítulo 9. Desde luego, llevando la idea hasta un extremo, podría pensarse en suprimir totalmente el modelo de error en reconocimiento; pero en este caso, como ya se ha hecho notar anteriormente, un error, por mínimo que sea, provocaría el rechazo inmediato de la cadena. Por otra parte, existe una posibilidad incluso menos drástica que la de autorizar tan sólo sustituciones, y que consiste en sólo prohibir las reglas de borrado. Esta aproximacíon, que reduciría a $|V|+1$ en vez de a $|V|$ el número de reglas, no se ha considerado, al resultar, en principio, satisfactorios los resultados obtenidos con sólo substituciones.

6.9.1 Consideraciones prácticas

La variabilidad en logitud tolerada cuando sólo hay sustituciones está limitada por la longitud de las cadenas más larga y más corta generables por la gramática. Ello, si no se toman precauciones, provoca, o bien el rechazo (si la implementación lo tiene previsto), o bien la obtención de resultados incoherentes; esto último debido a que la búsqueda en el trellis se detiene inesperadamente en uno de sus lados: se acaba la cadena y no se llega a un estado final, o todos los caminos posibles han llegado a un estado final sin haber llegado al final de la cadena.

En esta implementación, para evitar el rechazo y obtener a pesar de todo resultados coherentes, se han adoptado las dos reglas siguientes:

- Si una cadena es más larga que la máxima generable, se trunca.
- Si una cadena es más corta que la mínima generable, la distancia final es la del estado alcanzado que tenga la mínima distancia al detenerse la búsqueda (basta para ello buscar el nodo con la distancia acumulada mínima en la última columna del trellis).

Estas reglas suponen implícitamente que la parte final de una cadena muy larga no contendrá información esencial para poder decidir a qué clase pertenece (lo cual es usualmente cierto), y para aplicarlas es necesario conocer la longitud de las cadenas más corta y más larga, ambas obtenibles mediante los algoritmos detallados en el capítulo 10.

6.10 Implementación

Desde un principio, se adoptó para ECGI una representación no convencional de las gramáticas regulares inferidas. Esta representación, en forma de autómatas de estados etiquetados (LAS: labelled states automata, ver capítulo 2) fue impuesta inicialmente por razones de facilidad de implementación, ya que permite una cómoda visualización de la derivación óptima y una representación gráfica sencilla de los autómatas inferidos; aunque posteriormente se comprobó, como se explica en el apartado 6.5.3.3, que en algunos casos conduce a generalizaciones más acordes con la intuición. Resumiendo: **ECGI infiere (en la presente implementación) autómatas de estados etiquetados (LAS).**

Por otra parte, para no tener que trabajar con multitud de estados finales y para conseguir que la búsqueda de la derivación óptima consista realmente en la búsqueda del camino mínimo entre sólo dos vértices del trellis, en la implementación de ECGI se ha definido un *símbolo final* (•). De esta manera: **en los autómatas que infiere, ECGI añade, para cada estado final del LAS q_a, una transición $(q_a,\bullet,q_\bullet)$, que va de ese estado a un único *estado final* $q_\bullet$, etiquetado con •.** En estas condiciones, para que la condición de reconocimiento se siga cumpliendo, resulta necesario *añadir el símbolo final a todas las cadenas a reconocer por el autómata.* Además, por simetría, para simplificar la implementación y no tener que tratar de modo especial al estado inicial, se añade también a las cadenas el terminal correspondiente al estado inicial: ~. Esto no altera los autómatas inferidos, pero implica modificar la definición dada en el capítulo 2, de «lenguaje reconocido por un LAS», puesto que se supondrá que *el símbolo inicial es reconocido por el estado inicial* q_0. Con esta modificación, el lenguaje reconocido por un autómata LAS de ECGI, inferido para la gramática G, será (figura 6.21):

$$L_{ECGI}(G)=\{\ \sim\alpha\bullet:\ \alpha\in L(G)\ \}$$

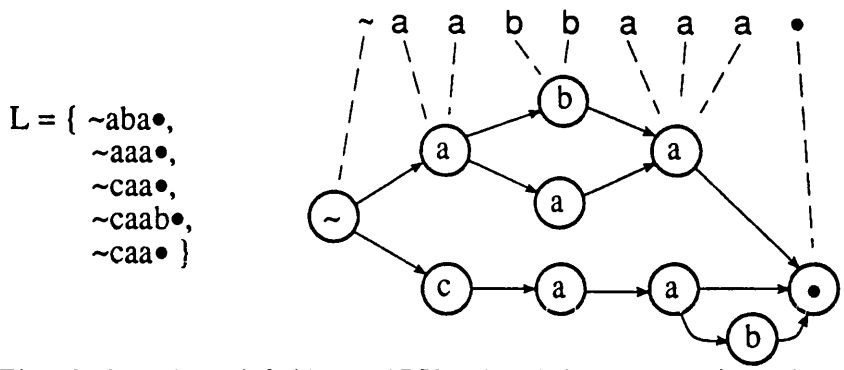


Figura 6.21 Ejemplo de autómata inferido por ECGI en la práctica: estados etiquetados, símbolo final. Lenguaje. Traza de edición con una nueva cadena: "aabbaaa•".

6.10.2 Representación Gráfica de un LAS de ECGI

Aprovechando que un LAS tiene etiquetados los estados, y que los autómatas inferidos por ECGI no tienen circuitos, lo cual permite ordenar los estados topológicamente, es posible una visualización muy descriptiva de dichos autómatas:

Para representar un LAS inferido por ECGI se dibuja un eje de estados (horizontal) 1..|Q| donde 1 es el primer estado y |Q| el último. Perpendicularmente a este eje, se coloca otro eje en el que figuran los terminales 1..|V|, donde |V|-1 es el primer terminal y |V| es el último. Los restantes símbolos se ordenan de manera que los símbolos más similares (según la física del problema o cualquier otro criterio adecuado) estén más cerca. En estas condiciones, cada estado q_i, de número de orden i (según el orden topológico), viene representado por un punto (i,eti(q_i)) (recuérdese que eti(q_i) es la etiqueta –terminal– asociado al estado), y las transiciones se pueden dibujar como rectas que unen los estados. La representación asegura que no hay dos estados en la misma vertical, con lo que únicamente pueden confundirse las transiciones que van entre dos estados con el mismo símbolo. Por otra parte, la confusión que es fácilmente evitable si se dibujan estas transiciones curvadas (aunque no se ha hecho en este trabajo).

Un dibujo de este tipo (véase un ejemplo en la figura 6.22) permite visualizar muy bien la estructura inferida, poniendo en relieve su linealidad, así como las secuencias de estados que representan subestructuras o subcadenas formadas por los mismos símbolos o similares (que corresponden a varios estados seguidos en la misma horizontal o próximos a ella).

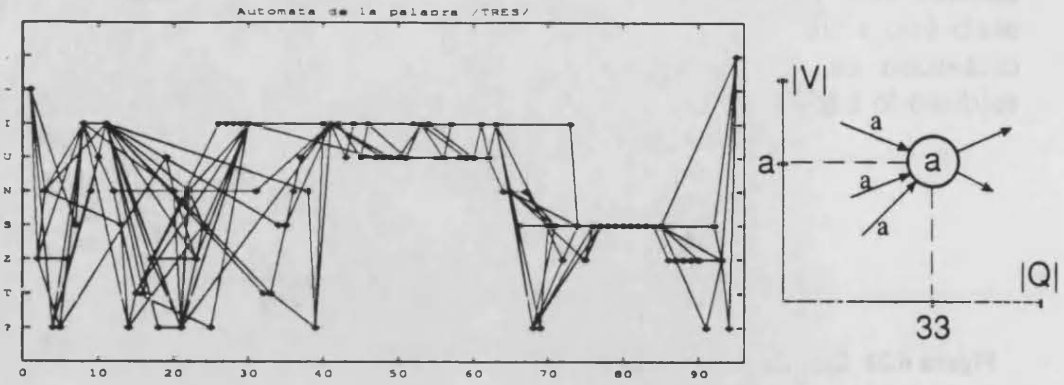


Figura 6.22 Representación de un autómata inferido por ECGI.

6.10.3 Version LAS de la construcción

El cambio de representación que suponen los LAS, aunque no afecta a la filosofía de ECGI, influye fuertemente en su implementación. En particular, el algoritmo de construcción adquiere un aspecto distinto, que es conveniente exponer.

El algoritmo, tal como se muestra a continuación, es una transcripcción literal del utilizado en la práctica, y por lo tanto, trata con un autómata LAS y no con una gramática. La principal diferencia, con respecto a la versión presentada previamente, reside en que se ha explicitado el algoritmo de análisis de la derivación óptima, así como las comprobaciones que se realizan sobre esta para detectar las reglas de no error. En efecto, el algoritmo de análisis sintáctico corrector de errores proporciona la derivación óptima *como una secuencia de vértices del trellis* $p=(q,a)\in Q\times V$, es decir, como una función $pred:Q\times V\rightarrow Q\times V$, asociada a cada vértice $p\in Q\times V$, que da el siguiente vértice del trellis según la derivación. Para detectar una regla de inserción ($A\rightarrow aA$), basta comprobar que el no terminal (el estado) es el mismo en ambos lados de la regla; es decir, hay que comprobar en la derivación que, al pasar de un vértice de la derivación (q,a) al siguiente (q_1,a_1), no cambia el estado asociado a dichos vértices, o sea que $q=q_1$. Para ello se define la función $q:Q\times V\rightarrow Q$, que al ser aplicada a un vértice $(q,a)\in Q\times V$ proporciona $q\in Q$. Similarmente, una regla de borrado puede detectarse comprobando que no cambia el terminal asociado a dos vértices del trellis, consecutivos según la derivación (o sea $a=a_1$), para lo cual se define la función $v:Q\times V\rightarrow V$ que dado $(q,a)\in Q\times V$, proporciona $a\in V$. Finalmente, en una regla de sustitución cambian tanto el estado como el terminal ($q\neq q_1$, $a\neq a_1$), siendo una regla de no error si además la etiqueta del estado (proporcionada por la función $eti:Q\times V\rightarrow V$) es igual al primero de los terminales (es decir, a) (recuérdese que el autómata es un LAS).

En la práctica, no es necesario comprobar si el terminal o el estado cambian al pasar de un vértice a otro del trellis. El algoritmo de análisis

corrector de errores, proporciona la información sobre tipo de regla de error de la que se trata en cada vértice del trellis. Para detectar una regla de no error, basta entonces comprobar sólo si la etiqueta del estado asociado al vértice es igual al terminal asociado al mismo. Sin embargo, la explicación anterior será útil para entender, tanto el algoritmo ECGI-AS, como su verisón modificada, que se presenta un poco más adelante.

```
Algoritmo ECGI-LAS (construcción)
Datos     Gi              /* Gramática actual */
          αi              /* muestra */
          D(αi,Gi)        /* Derivación óptima de xi en Gi */
Resultado Gi+1            /* Gramática inferida */
Auxiliar  /* las 3 primeras son funciones sobre un vértice del trellis */
       q:Q×V→Q            /* estado asociado */
       v:Q×V→V            /* terminal asociado */
       pred:Q×V→Q×V       /* vértice anterior según D */
       eti:Q→V            /* etiqueta de un estado (LAS) */
Variables
       p,pu∈Q×V           /* vértice examinado y último vértice en
                             que se ha identificado */
Inicialización
   p=(q•,•); pu=p; /* Se comienza desde el último punto del trellis */
Método
mientras p≠(q0,~) hacer
   p=prev(p)                 /* Se toma el siguiente vértice */
   si v(p)=eti(q(p))
   entonces                  /* Puede no ser error */
       sea αi=b1...bsβbt...bl;
       donde bs=v(p); bt=v(pu); β=z1...zh; bi,zi∈V,∀i
       si TipoRegla(p,pred(p))=Substitución
       entonces
          si αi=λ entonces
            si no existe ya, crear (q(p),bt,q(pu))
          sino crear
           qz1,qz2,...,qzh
           (q(p),z1,qz1),(qz1,z2,qz2),...,(qzh,bt,q(pu))
          pu=p
          fin si
       fin si
   fin si
fin mientras
fin ECGI-LAS (construcción)
```

Obsérvese la variable "pu", que permite recordar en qué vértice del trellis se ha encontrado la última regla de no error.

Aún sin ser necesario, se ha explicitado el sentido de recorrido de la derivación óptima, de atrás hacia delante, como recordatorio de cómo se ha realizado la implementación actual y de cómo se obtendría si se aplica el algoritmo `ViterbiCorrector` tal como se ha expuesto. Es posible, sin

ningún coste adicional, evaluar los sucesores en vez de los predecesores en cada vértice del trellis [Torró,89], con lo que se podría recorrer la derivación empezando desde el primer estado en vez del último.

Se ha prescindido de los detalles del algoritmo destinados a preservar el orden en el conjunto de estados. La regla es exactamente la misma que en el caso de la gramática y consiste simplemente en insertar los estados antes de q(pu).

Es posible visualizar muy sencillamente el proceso de construcción en un LAS, representándolo sobre el camino de edición (ver figura 6.23).

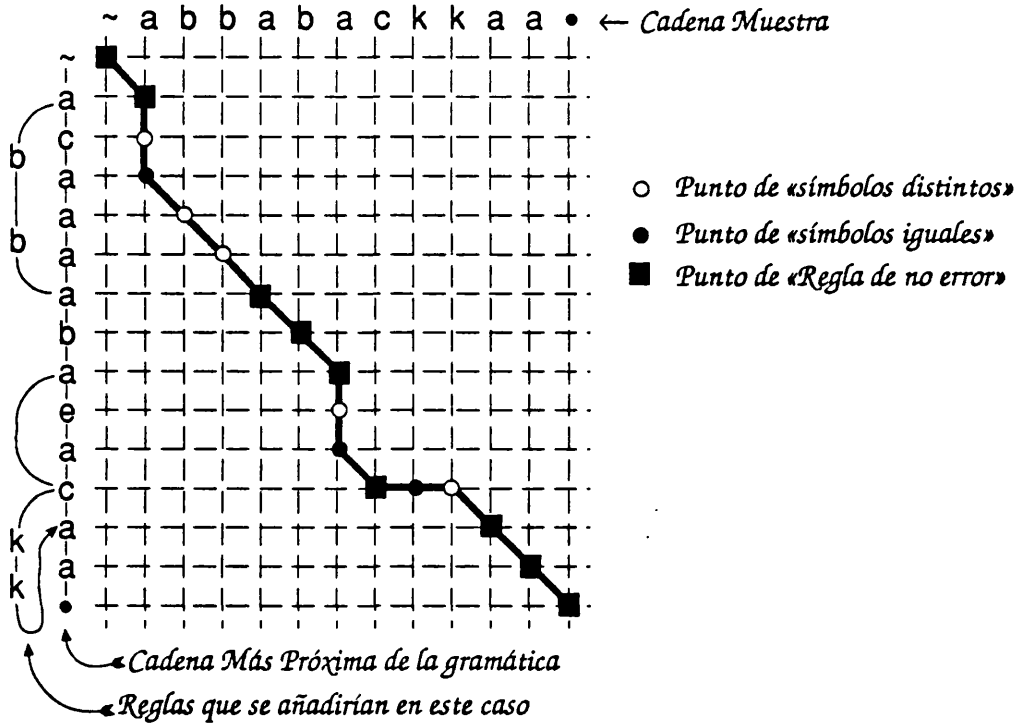


Figura 6.23 Algoritmo de construcción utilizado por ECGI visto sobre el camino de edición. El autómata es un LAS. Los puntos en los que los símbolos de ambas cadenas son iguales o distintos se muestran con circulos oscuros o claros respectivamente. Los cuadros negros señalan los puntos en los que hay una regla de no error. A la izquierda se muestran las reglas añadidas por ECGI en este caso.

Esta versión del algoritmo de construcción de ECGI se ha utilizado con buenos resultados, principalmente en la versión determinista de ECGI (ver capítulo 11), en la que era imprescindible. Sin embargo, para todos los restantes experimentos se ha utilizado una versión más refinada que intenta minimizar el número de no terminales añadidos. En esta versión, que llamaremos «definición 2» (la 1 es la original), se sustituye en el algoritmo anterior la línea:

```
|       si TipoRegla(p,pred(p))=Substitución     (1)
```

por :

```
|       si q(p)≠q(pu)  y   /* no pueden haber bucles */     (2)
            s≠t            /* no hay transición nil */
```

En la definición 2, se observa que no hay ninguna referencia directa al tipo de reglas (de error o no) que se analiza en cada momento. **No** debe creerse que esto equivale al método, explicado al principio del apartado, en el que se averigüaba el tipo de regla de error mediante comprobación del cambio de terminal o estado asociado, al pasar, en la derivación, de un vértice del trellis al siguiente. Aquí la comprobación se hace, no entre un vértice y el siguiente de la derivación, sino entre un vértice y *el último vértice en el que se ha producido identificación* (diremos que se produce *identificación* cuando el algoritmo toma las acciones correspondientes a la detección de una regla de no error en la derivación: aquí cuando v(p)=eti(q(p)) y q(p)≠q(pu) y s≠t, y en el caso original cuando v(p)=eti(q(p)) y TipoRegla(p,pred(p))=Substitución).

La definición 2, puede verse como un heurístico que modifica la definición de "identificación" para el caso de un autómata LAS, en un intento de aprovechar las características propias de estos autómatas. El comportamiento diferente de las dos definiciones se puede comprobar en la figura 6.24. Se observa que la definición 2 genera un estado menos. Nótese que la situación presentada en la figura es poco usual, máxime si se recuerda que uno de los heurísticos de ECGI otorga preferencia a la substitución.

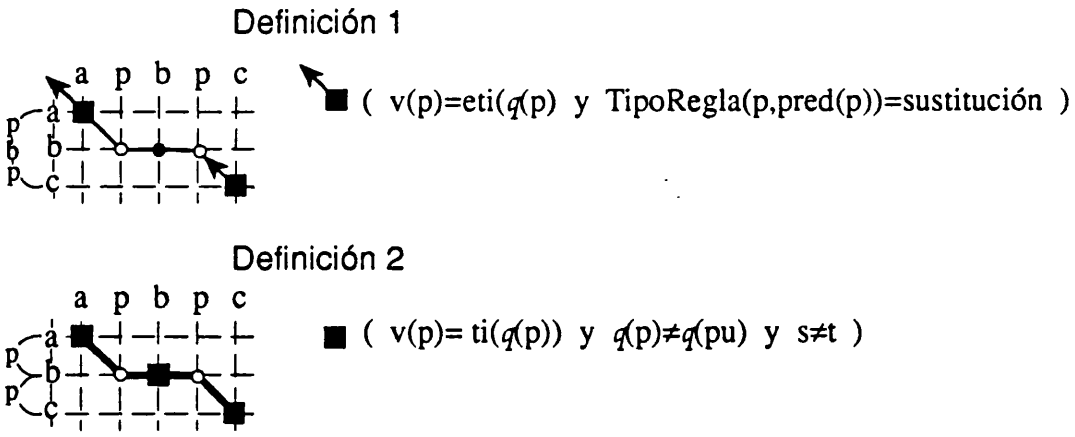


Figura 6.24 Camino de edición entre las cadenas "abc" y "apbpc". A la izquierda se muestran las transiciones y estados añadidos por ECGI, para cada una de las dos posibles definiciones de "identificación". Se observa que la definición 2 genera un estado menos.

Con la definición 2, el algoritmo no busca la siguiente regla de no error, sino el siguiente vértice del trellis en el que son iguales la etiqueta del estado como y terminal asociados a dicho vértice. Una vez encontrado este último, procede sin más a añadir las reglas necesarias para dar cuenta de los terminales de la cadena que hay entre este vértice y anterior que ha cumplido la misma condición. La única precaución que toma el algoritmo antes de proceder a modificar la gramática (autómata), es la de no producir reglas no permitidas (ni bucles ni nil) y de no duplicar transiciones (esto último se comprueba en el caso $\alpha=\lambda$, que es el único que puede crear una transición ya existente).

7

ECGI Estocástico

ECGI, tal como se ha presentado en el capítulo anterior, es una herramienta muy eficaz que ya en los experimentos piloto demostró su gran poder de modelización. Proporcionó, desde un primer momento, resultados del orden del 97% aciertos, en experimentos de reconocimiento llevados a cabo sobre dígitos hablados (monolocutor), los cuales se hallaban representados mediante cadenas de símbolos muy poco elaborados (ver capítulo 8: experimentos con corpus piloto).

Sin embargo, aunque mejorable (utilizando símbolos más elaborados, e incluso una representación continua), esta tasa de reconocimiento es aún insuficiente en aplicaciones prácticas; más cuando se aspira a resolver tareas mucho más difíciles, en las que las tasas de aciertos de los mejores métodos descienden drásticamente (multilocutor, grandes diccionarios, diccionarios difíciles, ...). Afortunadamente, existe una vía inmediata para mejorar las capacidades de modelización de las estructuras inferidas por ECGI, la cual se deriva directamente de la constatación de que los autómatas inferidos no han retenido ninguna información relacionada con la importancia relativa de las distintas (sub)estucturas inferidas, es decir, ninguna información relativa a las *frecuencias de utilización* de las distintas reglas, no terminales y terminales[1]. Esta, es precisamente la información incorporada por los métodos estocásticos, la cual estaría disponible si ECGI infiriera una *gramática estocástica*.

En los siguientes apartados se expone cómo se han incorporado probabilidades a las reglas de las gramáticas generadas por ECGI, y cómo se han aprovechado estas probabilidades en reconocimiento.

---

[1]Esto no es del todo cierto como se comprobará cuando se defina el *tráfico* y el *número de caminos*. en el capítulo 10.

7.1 Ambiguedad y probabilidad

En el caso de que la gramática inferida por ECGI sea no ambigua se pueden obtener, mediante el procedimiento expuesto en el capítulo 2 y con todas las garantías de estocastiscidad, las probabilidades de utilización de las reglas, con sólo llevar la cuenta del número de veces que se utilizan en la generación de las muestras de R^+. Sin empargo, en la realidad las gramáticas generadas por ECGI son ambiguas (ver capítulo 9), con lo que en principio se hará necesario disponer de un número mucho mayor de muestras (o una reconsideración repetitiva de R^+) y utilizar algún método de reestimación de probabilidades (p.e.: reestimación por Viterbi [Levinson,83] [Rabiner,89]) o también Baum-Welch) que asegure la convergencia y fiabilidad de dicha reestimación (ver también capítulo 2). Una aproximación de este tipo se ha planteado y llevado a la práctica en [Castaño,90] proporcionando resultados satisfactorios, aunque no necesariamente mejores que los aquí obtenidos.

No obstante, estos métodos, relativamente costosos, no son imprescindibles en la práctica y, como se mostrará a continuación, una aproximación no estricta (y por lo tanto subóptima) al problema permite obtener tasas de reconocimiento equivalentes (e incluso, en primera aproximación, mejores) a las de un método estrictamente correcto. En consecuencia, en todo el resto del trabajo actuaremos como si las gramáticas generadas por ECGI no fueran ambiguas, asumiendo que todos los resultados que se obtengan serán probablemente subóptimos.

7.2 Construcción estocástica

La utilización de la información estadística para mejorar la estructura inferida por ECGI, es decir, su empleo para realizar la búsqueda de la derivación óptima sobre la que luego se basará la construcción, tropieza con dos dificultades:

- La información frecuencial se renueva a cada muestra.
- No sólo es necesaria la información probabilística referente a las reglas de la gramática que se ha inferido, sino que también se debe disponer de la misma información sobre las demás reglas de la gramática *expandida* (sobre las reglas de error), puesto que todas ellas se utilizan para el análisis sintáctico de las muestras.

La modificación de las frecuencias de uso de las reglas a cada muestra implica que, tras cada de ellas, es necesario renormalizar toda la gramática (es decir, convertir las nuevas frecuencias en probabilidades). Aunque ello, en contra de lo que pueda parecer, no resulta excesivamente gravoso comparado con la complejidad del análisis sintáctico corrector de errores, se

ha renunciado (en primera aproximación) a esta posibilidad. Por lo tqnto, en todo el resto de este trabajo, el modelo estocástico se estudiará únicamente con miras a mejorar las capacidades de reconocimiento de ECGI, **no utilizándose en ningún caso las probabilidades en el proceso de aprendizaje de la *estructura* del modelo.** Conviene notar que ésta es la sistemática más frecuente en reconocimiento de formas: estimación de probabilidades de un modelo estructural fijo, obtenido sin hacer uso alguno de estas probabilidades.

Una consecuencia de esta decisión, entre otras, es la de imposibilitar la integración aprendizaje-reconocimiento en el caso del aprendizaje continuo, al emplearse algoritmos de optimización distintos en cada etapa.

Por otra parte, hay que notar que el procedimiento de renormalizar a cada muestra no es el único factible si se desea utilizar la información estocástica durante la inferencia de la estructura del modelo generado por ECGI. Existe otra posibilidad intermedia, estudiada en [Castaño,90], que consiste en particionar (o reutilizar) R^+ para reestimar las probabilidades. En vez de proceder a una renormalización a cada muestra, se reconoce y construye con las probabilidades estimadas en un paso anterior hasta reunir cierto número de muestras. En ese momento es cuando se procede a la renormalización. Los resultados obtenidos por M.A.Castaño muestran una ligera pérdida (no explicada) en la eficiencia del reconocimiento, pero muestran la factibilidad de la idea.

En cuanto al segundo de los puntos arriba mencionados, que destaca la necesidad de disponer del modelo estocástico también para la gramática expandida, es fácil comprobar que ello es una dificultad inherente al proceso *incremental* de aprendizaje de ECGI. Renunciando a una aproximación incremental, sería posible prescindir de las probabilidades asociadas a las reglas de error. En efecto, al estar incluído el conjunto de muestras R_+ en el lenguaje de la gramática inferida, se pueden estimar las probabilidades de las reglas a partir de R_+, puesto que no resulta necesario expandir la gramática para realizar el análisis sintáctico.

De todas maneras, y como se verá en los apartados siguientes, es posible estimar de manera fiable un modelo de error estocástico y muy eficaz.

7.3 Reconocimiento estocástico

La estimación de las probabilidades de las reglas de error de la gramática expandida de una gramática G=(N,V,P,S), supone una serie de dificultades que se agudizan más, si cabe en el caso particular de ECGI. En efecto, la misma idea en que se basa el método hace imposible estimar realmente la probabilidad de una regla de error concreta: siempre que alguna es utilizada,

el proceso de construcción de ECGI se encarga inmediatamente de anularla generando las reglas mínimas imprescindibles para que dicha regla de error no sea necesaria para reconocer (generar) la nueva muestra (y otras similares).

Por otra parte, aún en el caso de que fuera posible, el enorme número de reglas de error[2]: $(2\cdot|V|+1)\cdot P$, exigiría disponer de un número prohibitivo de muestras de aprendizaje para poder efectuar una estimación aceptable de las probabilidades correspondientes.

Para soslayar ambos problemas, no queda más remedio que renunciar efectivamente a la estimación individual de *todas* las probabilidades de las reglas de error, y recurrir a una aproximación más "global", agrupando dichas reglas por *clases* y asignar a cada regla la probabilidad de su clase. Se hace entonces la asunción de que **la probabilidad de una regla de error no depende de los no terminales asociados a dicha regla** (de la "posición" de la regla). En estas condiciones, en la probabilidad de una regla de error sólo interviene el tipo de ésta (inserción, borrado o sustitución) y el símbolo (no terminal) que tiene asociado, por lo que se pueden agrupar las reglas de error en $2\cdot|V|+|V|^2$ clases, siendo éste el número de probabilidades que efectivamente hay que estimar.

Al no estar estas probabilidades asociadas a una regla en concreto, se puede considerar que una estimación de ellas se puede obtener a partir del conteo de la ocurrencia *de las de su clase* durante la construcción y/o inferencia de la gramática a partir de R^+ (incluso aunque muchas de las reglas que hayan intervenido en el conteo queden anuladas por el proceso de construcción). En consecuencia, ECGI rellena durante la construcción una tabla, que denominaremos *Tabla de Sustitución* (al considerar inserciones y borrados como sustituciones en las que interviene el símbolo nil ε), que contiene la frecuencia de sustitución de cada uno de los terminales por otro: $f_s(b,a)$ (substitución de 'a' por 'b'), $f_b(a)$ (borrado de 'a') y $f_i(b)$ (inserción de 'b'). $f_s(a,a)$ nos da la frecuencia de las reglas NO erróneas que tienen asociado el símbolo 'a'. $f(\varepsilon,\varepsilon)$ se define 0.

La tabla de sustitución será por lo tanto una matriz de $(|V|+1)^2$ elementos tales que (En tabla 7.1 se muestra un ejemplo):

$$T_s(b,a) = f_s(b,a)\ \forall a,b\neq\varepsilon\ ;\quad T_s(b,\varepsilon) = f_i(b)\ ;\quad T_s(\varepsilon,a)=f_b(a);\quad T_s(\varepsilon,\varepsilon)=0$$

El conteo de estas frecuencias se realiza simultáneamente con el análisis de la derivación óptima durante la construcción, con sólo añadir al contador de los errores de cada clase aquellos errores que aparecen en dicha derivación.

---

[2] Como se verá más adelante, y debido a las particularidades del algoritmo ViterbiCorrector, el número de reglas de error es en realidad menor al presentado: todas las reglas de inserción de las reglas con el mismo no terminal a la izquierda son comunes.

En la implementación práctica, debido a que se genera un autómata de estados etiquetados (LAS, ver capítulo 2) y a que ECGI añade los símbolos inicial y final (ver capítulo 6), la tabla incorpora también a estos últimos, con frecuencias $f_s(\sim,\sim)$ y $f_s(\bullet,\bullet)$ iguales al número de cadenas de R^+, puesto que nunca se insertan, sustituyen o borran.

Tabla 7.1 Tabla de errores de sustitución obtenida por ECGI durante la inferencia de un modelo para el dígito hablado /cuatro/ a partir de 150 muestras. Los terminales son V={E,I,K,N,O,R,S,T,U,Z,e,n,o,r,v}. Obsérvese que además están incluídos el símbolo inicial y final y que el símbolo ε se representa como /. El símbolo # es una provisión por si apareciara un símbolo desconocido, y su frecuencia debe de ser siempre 0.

	E	I	K	N	O	R	S	T	U	Z	e	n	o	r	v	#	~	.	/
E	555	0	1	0	5	4	0	1	0	2	0	0	0	3	8	0	0	0	13
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	32	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	2
N	0	0	0	5	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
O	4	0	2	0	870	0	0	0	1	0	0	0	0	0	3	0	0	0	8
R	0	0	0	0	0	67	0	3	0	0	0	0	0	0	0	0	0	0	3
S	1	0	0	0	0	0	22	1	0	1	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	729	2	3	0	3	1	0	0	0	0	0	8
U	0	0	1	2	7	0	0	10	946	0	0	0	8	0	1	0	0	0	3
Z	2	0	0	0	1	1	2	2	0	275	1	0	0	3	1	0	0	0	7
e	2	0	0	1	0	0	0	0	0	0	11	0	0	0	0	0	0	0	2
n	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
o	0	0	1	0	4	0	0	2	2	0	0	0	348	0	2	0	0	0	5
r	3	0	0	0	0	2	0	0	0	1	0	1	0	63	2	0	0	0	3
v	2	0	1	0	4	4	0	2	1	2	2	0	1	1	484	0	0	0	12
#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
~	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	150	0	0
.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	150	0
/	11	0	2	0	8	3	1	6	6	3	2	0	4	0	2	0	0	0	0

Aún con la drástica reducción aplicada, el número de muestras es relativamente pequeño respecto al de parámetros a estimar, por lo que gran parte de las frecuencias en la tabla de sustitución de cada gramática (autómata) son de valor muy reducido o nulo. Ello conduce a una estimación extremadamente pobre de las probabilidades. Para mejorar dicha estimación, se procede a *sumar* las tablas de sustitución de todas las clases a reconocer, reduciéndolas a una *única tabla de sustitución*, común a todas ellas. Dicho de otro modo, **la extensión estocástica de ECGI supone que las probabilidades de los distintos tipos de reglas de error son las mismas para todas las clases a reconocer**, e iguales a la probabilidad de cada tipo en el conjunto de las clases. Esto es conveniente a menos de disponer de un muy elevado número de muestras: en [Prieto,88], experimentos de reconocimiento de letras habladas proporcionaron en general una tasa de reconocimiento de 0.5% mejor utilizando las tablas de substitución promediadas (p.e. 65% en lugar de 64.5%) (ver apéndice A).

7.4 La estocasticidad de la gramática expandida

7.4.1 Las probabilidades de deformación

Una vez inferida por ECGI una gramática, es necesario obtener, a partir de la correspondiente tabla de sustitución y de las frecuencias de uso en R^+ de las reglas, las probabilidades de éstas y las de las reglas de la gramática expandida; todo ello asegurando la estocasticidad, tanto de la gramática sin expandir como de la expandida.

Utilizando las definiciones dadas en el capítulo 2, si el no terminal A toma parte de las m reglas $A \rightarrow x_k B_k, k=1...m$; de la gramática característica y n_k es el número de veces que se utiliza una de ellas r_k en R^+ basta calcular:

$$p(r_k) = \frac{n_k}{\sum_{j=1..m} n_j}$$

para obtener su probabilidad, asegurando simúltáneamente la estocasticidad de la gramática:

$$\sum_{k=1..m} p(r_k) = 1$$

En el caso de la gramática expandida, para cada una de estas reglas hay que añadir las reglas de error de sustitución (con probabilidad de deformación $p_s(b \mid x_k)\ \forall b \in V$), de borrado ($p_b(\varepsilon \mid x_k)$) y de inserción ($p_i(b \mid x_k)\ \forall b \in V$). A partir de la tabla de sustitución, que nos proporciona las frecuencias $f_s(b,a)$, $f_b(a)$ y $f_i(b)$, es directo obtener una estimación de cada una de estas probabilidades:

$$p_s(b \mid x_k) = \frac{f_s(b,x_k)}{N_{x_k}}; \qquad p_b(\varepsilon \mid x_i) = \frac{f_b(x_k)}{N_{x_k}}; \qquad p_i(b \mid x_k) = \frac{f_i(b)}{N_{x_k}};$$

$$N_{x_k} = \sum_{\forall b \in V} f_s(b,x_k) + f_b(x_k) + \sum_{\forall b \in V} f_i(b)$$

que cumple la condición para que las probabilidades de deformación sean consistentes.

Sin embargo, esta estimación no es válida para el caso del ECGI. Un primer signo de ello se tiene si se observa que, debido a que en la tabla de sustitución las frecuencias de inserción $f_i(b)$ no dependen del símbolo

anterior (el de la regla), la expresión de $P_i(b \mid x_k)$ depende de x_k sólo a través de la normalización. Ello obviamente es consecuencia directa del modelo de error utilizado, y se refleja en el algoritmo ViterbiCorrector, el cual, al construir las reglas de error en el trellis, únicamente añade una regla de inserción por cada símbolo y **por cada no terminal a la derecha** (y no por cada regla, como obligaría un modelo de error más complejo y que permitiría una dependencia real de x_k). Es decir, en el trellis se tendrán las siguientes reglas de error para las m reglas $A \rightarrow x_k B_k$, k=1...m:

$A \rightarrow bB_k$	$\forall b \in V; b \neq x_k$; k=1...m	(sustitución)
$A \rightarrow B_k$	k=1...m	(borrado)
$A \rightarrow bA$	$\forall b \in V$	(inserción)

Esta peculiaridad del modelo de error utilizado por ECGI[3], impide que se pueda utilizar el hecho de que las probabilidades de deformación sean consistentes como garantía de la estocasticidad de la gramática expandida (ver capítulo 2), lo que hace necesario adaptar la condición a este caso particular:

Recuérdese que si G es *estocástica* y el no terminal A toma parte en las m reglas $r_k = A \rightarrow x_k B_k$ de probabilidad $p(r_k)$, k=1,...,m; se cumple:

$$\sum_{k=1..m} p(r_k) = 1$$

Al construir G^e, para cada una de estas reglas asociadas a A, hay que añadir las reglas de error de sustitución (con probabilidad de deformación $p_s(b \mid x_k)$ $\forall b \in V$) y la de borrado ($p_b(\varepsilon \mid x_k)$) correspondientes; y para todas ellas en conjunto hay que añadir las reglas de inserción ($p_i(b)$ $\forall b \in V$). La probabilidad de una regla de substitución o borrado se puede entonces definir como la probabilidad combinada de que se produzca el error en cuestión y de que se utilize la regla original r_k; es decir, la probabilidad de una regla de substitución sería $p_s(b \mid x_k) \cdot p(r_k)$ y la de una regla de borrado $p_s(\varepsilon \mid x_k) \cdot p(r_k)$, mientras que la probabilidad de inserción es independiente de la regla r_k aplicada. La suma de todas estas reglas deberá ser la unidad para poder asegurar la estocasticidad de G^e, con lo que la condición se puede escribir (recuérdese que $p_s(x_k \mid x_k) \cdot p(r_k)$ es la probabilidad de la regla NO errónea):

$$\sum_{k=1..m} p(r_k) \cdot \Big(\sum_{\forall b \in V} p_s(b \mid x_k) + p_b(\varepsilon \mid x_k) \Big) + \sum_{\forall b \in V} p_i(b) = 1$$

---

[3] Modelo, por otro lado, muy utilizado en análisis sintáctico corrector de errores en gramáticas regulares.

En los siguientes apartados se expone el método que emplea ECGI para estimar las probabilidades de deformación.

7.4.2 Estocasticidad según ECGI

ECGI separa totalmente las probabilidades de las reglas de inserción de las de las demás reglas de error. Para ello, se define la probabilidad de inserción de un símbolo 'b', como:

$$p_i(b) = \frac{f_i(b)}{\sum_{\forall a,b\in V} f_s(a,b) + \sum_{\forall a\in V} f_b(a) + \sum_{\forall b\in V} f_i(b)}$$

es decir, como la relación entre el número de veces que se ha insertado 'b' en R_+ y el número total de errores y no errores que se han producido en el mismo R_+.

Las otras probabilidades de error se definen siguiendo la filosofía usual, pero aplicando un factor corrector $(1\text{-}P_i)$ para asegurar la estocasticidad:

$$p_s(b \mid x_k) = (1\text{-}P_i)\cdot\frac{f_s(b,x_k)}{N_{x_k}}; \qquad p_b(\varepsilon \mid x_k) = (1\text{-}P_i)\cdot\frac{f_b(x_k)}{N_{x_k}};$$

$$N_{x_k} = \sum_{\forall b\in V} f_s(x_k,b) + f_b(x_k)$$

$$P_i = \sum_{\forall b\in V} p_i(b)$$

P_i representa la *probabilidad de inserción de cualquier símbolo* en cualquier posición de la gramática, es decir la probabilidad de que se utilize una regla de inserción. $(1\text{-}P_i)$ obviamente es la probabilidad de que se emplee cualquier otro tipo de regla. Nótese que P_i también se puede obtener directamente como la la relación entre el número total de inserciones N_i y el número total de errores N_e en R_+:

$$P_i = \frac{N_i}{N_e}$$

$$N_i = \sum_{\forall b\in V} f_i(b); \qquad N_e = \sum_{\forall a,b\in V} f_s(a,b) + \sum_{\forall a\in V} f_b(a) + \sum_{\forall b\in V} f_i(b))$$

Es fácil comprobar que con estas definiciones la estocasticidad se cumple:

$$\sum_{k=1..m} p(r_k)\cdot\Big(\sum_{\forall b\in V} p_s(b \mid x_k) + p_b(\varepsilon \mid x_k)\Big) + \sum_{\forall b\in V} p_i(b) =$$

$$\sum_{k=1..m} p(r_k)\cdot(1-P_i) + P_i = 1\cdot(1-P_i) + P_i = 1$$

En resumen, y utilizando las definiciones anteriores, las reglas de error utilizadas por ECGI para el análisis sintáctico corrector de errores estocástico, con sus probabilidades, se derivan de la gramática característica a partir de cada grupo de m reglas $A\rightarrow x_k B_k$ k=1...m con el mismo no terminal A a la izquierda, de la siguiente manera:

$p(\mathbf{A\rightarrow bB_k}) = p(r_k)\cdot p_s(b \mid x_k)$;	$\forall b\in V$; k=1...m	*(substitución y no error)*
$p(\mathbf{A\rightarrow B_k}) = p(r_k)\cdot p_b(x_k)$	k=1...m	*(borrado)*
$p(\mathbf{A\rightarrow bA}) = p_i(b)$	$\forall b\in V$	*(inserción)*

7.4.3 En la práctica

Realizando una pequeña conversión, es posible escribir de otra manera la probabilidad $p_i(b)$ de inserción del símbolo 'b':

$$p_i(b) = \frac{f_i(b)}{N_e} = \frac{N_i}{N_i}\cdot\frac{f_i(b)}{N_e} = \frac{N_i}{N_e}\cdot\frac{f_i(b)}{N_i} = P_i\cdot\frac{f_i(b)}{N_i} = P_i\cdot p_{ri}(b)$$

con lo que podemos ponerla en función de la probabilidad de inserción de cualquier símbolo y de la *probabilidad relativa de inserción* del símbolo 'b' (relación entre la frecuencia de inserción de 'b' y el número total de inserciones):

$$p_{ri}(b) = \frac{f_i(b)}{N_i}$$

Esto permite un tratamiento más unificado de los valores en la tabla de sustitución T_s. La obtención de las probabilidades a partir de T_s implica simplemente normalizar la tabla por líneas (figura 7.1): a cada elemento se le asigna el valor resultante de dividir el valor original por la suma de los valores en la misma línea:

$$T_{ps}(b,a) = \frac{T_s(b,a)}{\sum_{\forall x} T_s(x,a)} \begin{cases} p_s(b|x) = \frac{f_s(b,x)}{N_x} & a \neq \varepsilon \\ p_b(a) = \frac{f_b(a)}{N_x} & b = \varepsilon \\ p_{ri}(b) = \frac{f_i(b)}{N_i} & a = \varepsilon \end{cases}$$

$$N_x = \sum_{\forall b \in V} f_s(x,b) + f_b(x); \qquad N_i = \sum_{\forall b \in V} f_i(b)$$

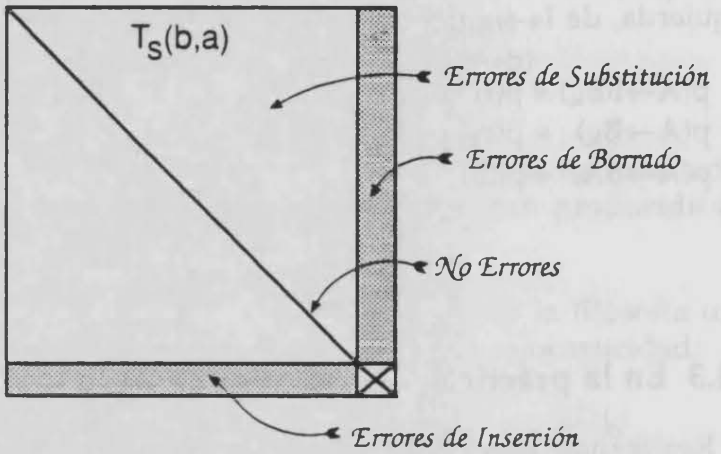


Figura 7.1 Estructura de $T_S(b,a)$ y $T_{ps}(b,a)$, tabla de frecuencias de error y tabla de probabilidades de error respectivamente.

Nótese que en la implementación real se ha permitido que los símbolos inicial y final del LAS intervengan en la normalización, no teniendo ello importancia: todos los valores que tienen asociados son 0 excepto los de la diagonal. Sólo queda afectada la estimación de P_i ya que se contabiliza el número de no errores asociado a ambos símbolos, lo que rebaja ligeramente la probabilidad relativa de una inserción.

Debido al número limitado de cifras significativas de las que dispone un calculador real, y a que en el mismo la rapidez de la adición es usualmente muy superior a la del producto, normalmente se realiza todo el cálculo utilizando los logaritmos de las probabilidades (esto, en su caso, también permitiría trabajar únicamente con enteros):

$$T_{ls}(b,a) = -\log(T_{ps}(b,a))$$

siendo esta pues la tabla que se utilizará en la realidad. También se ha procedido a un cambio de signo (por comodidad: son todos negativos), y se tomará la precaución de saturar al máximo número representable cuando el argumento del logaritmo es 0.

El mismo proceso se aplica a las probabilidades de las reglas P(r). La contabilización de las frecuencias n_k de las reglas $r_k=(A \rightarrow x_k B_k)$ se realiza

también al recorrer la derivación óptima, simultáneamente con la contabilización de las frecuencias de utilización de los no terminales (estados) f(A). Como todas las cadenas que hacen uso de un estado hacen uso de alguna de las transiciones que salen de ese estado, las probabilidades de las reglas se pueden calcular directamente como:

$$p(r_k) = \frac{n_k}{f(A)}$$

Resumiendo todo lo dicho hasta ahora, la relación de recurrencia que emplea ECGI durante el reconocimiento de una cadena muestra $\alpha=a_1,...a_n$ para calcular la derivación óptima (ver capítulo 5, algoritmo `ViterbiCorrector`), se puede escribir como[4]:

$$D(A_j,t)=\min\begin{cases} D(B_k,t-1) + L_{ni} + T_s(a_t,x_k) + L(B_k\rightarrow x_kA_j) & k=1...m \\ D(B_k,t) + L_{ni} + T_s(\varepsilon,x_i) + L(B_k\rightarrow x_kA_j) & k=1...m \\ D(Aj,t-1) + L_i + T_s(a_t,\varepsilon) \end{cases}$$

$$L_{ni} = -\log(1-P_i); \qquad L_i = -\log(P_i); \qquad L(r)= -\log(p(r));$$

donde D(A,t) representa el coste acumulado en el vértice correspondiente al estado A y al terminal t de la cadena en el trellis del análisis sintáctico. Se ha minimizado al tratarse de los logaritmos *negados* de las probabilidades a maximizar.

Nótese que se ha utilizado la representación *hacia atrás* (se miran los m predecesores de A_j en lugar de los sucesores). Todo el razonamiento es absolutamente análogo para este caso, y como ya se ha mencionado, es la representación que utiliza la presente implementación de ECGI.

En el capítulo 9 se presenta una tabla resumen de experimentos, que permite comprobar como la aportación de la información estadística mejora sensiblemente las prestaciones de ECGI en reconocimiento de formas. En el mismo capítulo se efectuán también una serie de simplificaciones sobre el modelo aquí presentado, con el fin de comprobar empíricamente cuál es la importancia relativa de la información estadística que aportan las frecuencias de las reglas de la gramática sin expandir y las frecuencias de los distintos errores del modelo de error.

---

[4] En la práctica $L_{ni}+L(B_k\rightarrow x_kA_j)$ se precalculan antes de la optimización, y lo mismo podría hacerse con $L_i+T_s(b,\varepsilon)$ para utilizar directamente el $-\log(p_i(b))$.

7.5 Sólo sustitución

Al igual que en el caso no estocástico en el caso estocástico se ha estudiado la posibilidad de utilizar sólo errores de substitución en reconocimiento. Si no se permiten los errores de inserción ni de borrado los problemas de estocasticidad se simplifican grandemente. Si consideramos que las probabilidades de inserción y borrado son nulas (y por lo tanto ignoramos sus respectivas frecuencias), se podrán obtener las probabilidades de deformación con el procedimiento usual. Para las m reglas $A \rightarrow x_k B_k$; k=1..m:

$$p_s(b \mid x_k) = \frac{f_s(b,x_k)}{N_{x_k}}; \qquad p_b(\varepsilon \mid x_k) = 0; \qquad p_i(b \mid x_k) = 0;$$

$$N_{x_k} = \sum_{\forall b \in V} f_s(x_k,b)$$

estando asegurada la estocasticidad:

$$\sum_{i=1..m} p(r_i) \cdot \left(\sum_{\forall b \in V} p_s(b \mid x_i) \right) = 1$$

En la práctica, basta poner a cero las frecuencias de borrado en T_s y obtener T_{ls} de la misma manera que anteriormente. En la programación dinámica no se generarán las transiciones de borrado ni las de inserción.

Hay que notar que los heurísticos, adoptados para obtener un resultado aunque la cadena muestra sea más larga que la más larga de la gramática o más corta que la más corta (ver capítulo 6), no garantizan la estocasticidad si llegan a aplicarse (¡equivalen a aceptar una cadena que no está en la gramática expandida!). Obviamente ello es relativamente poco importante, considerando la ya gran "informalidad" del mismo heurístico.

Los resultados comparativos entre el modelo de error completo y el de sólo sustitución, en el caso no estocástico y estocástico se muestran en el capítulo 9.

8

Resultados experimentales

En este capítulo se exponen con detalle los experimentos de reconocimiento de formas que se han realizado a lo largo de este trabajo para comprobar la viabilidad y buen funcionamiento del algoritmo ECGI, así como de sus extensiones.

En primer lugar se describen los conjuntos de datos a partir de los cuales se organizaron los experimentos. Seguidamente, se detallan éstos y se tabulan sus resultados.

En este capítulo no se incluyen los resultados de los experimentos de obtención de cadena media, de simplificación de autómatas, de obtención de autómatas deterministas y de comparación entre los distintos criterios, todos los cuales se hallan a continuación, en los capítulos correspondientes (capítulos 9, 10 y 11). Tampoco se pretende exponer *todos* los experimentos realizados, pues se han realizado muchos que luego han quedado obsoletos por mejoras (a veces mínimas) en la parametrización, se han visto incluídos en otros por ampliación del conjunto de datos, etc.; sin contar la gran cantidad de pequeñas pruebas que requiere la puesta a punto de un método heurístico como lo es ECGI.

Se han empleado cinco grupos distintos de datos, cada uno de ellos correspondiente a un grupo distinto de experimentos.

De los cuatro grupo de experimentos, dos involucraron dígitos hablados, uno letras habladas y otros dos utilizaron imágenes. De estos últimos el primer grupo correspondía a dígitos manuscritos y el otro a dígitos impresos. El primer grupo de datos de dígitos hablados, basado en una parametrización y etiquetados elementales, se utilizó como experimento piloto cada vez que se desarrollaba un nuevo prototipo de ECGI, pues representaba un auténtico desafío por lo poco elaborado de los símbolos utilizados. Sin embargo, el corpus de datos de letras habladas es, con margen, el más difícil (sobre todo el subconjunto de EE-letras), dada la extrema similitud de las formas a reconocer.

Casi todos los experimentos se realizaron en un ordenador Hewlett-Packard HP9300, con sistema operativo UNIX (HP-UX)[1].

8.1 Reconocimiento del Habla

Se describen a continuación los experimentos realizados en el campo para el cual fue diseñado ECGI en un principio: el reconocimiento del habla. Se presentan primero los corpus de datos (el piloto, el principal y las letras), seguidos de la descripcción de los distintos experimentos y de los resultados obtenidos en cada caso.

8.1.1 Representación simbólica de la señal vocal

Los corpora de los dígitos hablados están formados por la pronunciación repetida frente a un micrófono, por parte de uno o varios locutores (según el corpus), de los diez dígitos *castellanos* (tabla 8.1).

Tabla 8.1 Los diez dígitos castellanos.

Palabra	Transcripción fonética	Nº de sílabas
cero	/θéro/	2
uno	/úno/	2
dos	/dós/	1
tres	/trés/	1
cuatro	/kwátro/	2
cinco	/θínko/	2
seis	/seís/	1
siete	/sjete/	2
ocho	/óco/	2
nueve	/nwéβe/	2

El corpus de las letras habladas está formado por las treinta letras del alfabeto castellano (tabla 8.2).

[1]Los experimentos iniciales se llevaron a cabo en un Eclipse C-350 de Data-General, aunque luego se repitieron en el HP9300. Por aquel entonces, el ECGI, integrando inferencia y reconocimiento, y junto con los autómatas inferidos, cabía en 64Kbytes de memoria (¡que tiempos aquellos!). Los últimos experimentos se realizaron con la última versión de ECGI, escrita en C y que funciona en un RISC 6000 de IBM (S.O. AIX).

Tabla 8.2 Las letras del alfabeto castellano habladas y su transcripción fonética.

A	/a/	J	/xóta/	R	/ére/
B	/be/	K	/ka/	RR	/érre/
C	/θe/	L	/éle/	S	/ése/
CH	/ce/	LL	/éλe/	T	/te/
D	/de/	M	/éme/	U	/u/
E	/e/	N	/éne/	V	/úβe/
F	/éfe/	Ñ	/éɳe/	W	/úβedóble/
G	/xé/	O	/o/	X	/ékis/
H	/áce/	P	/pe/	Y	/ígriéγa/
I	/i/	Q	/ku/	Z	/θéta/

Llamaremos "EE-letras" al conjunto de 9 letras {F, L, LL, M, N, Ñ, R, RR, S} cuya pronunciación se diferencia estrictamente por la consonante *intermedia* (no confundir con las E-letras {B,C,D,E,G,P,T y –en inglés– V y Z}, que se diferencian por la consonante *inicial*).

Las palabras fueron adquiridas en una habitación no especiamente aislada, mediante un micrófono de proximidad (relación señal/ruido ≈ 40 dB). Se trata de experimentos de reconocimiento de *palabras aisladas*, por lo que existía una gran pausa entre palabra y palabra, con el fin de trivializar su extracción y segmentación del fondo. El proceso de conversión de estas palabras en cadenas de símbolos es distinto para cada uno de los corpora presentados, aunque sigue el esquema adquisición, parametrización y etiquetado presentado en el capítulo 1.

8.1.1.1 Corpus piloto: Dígitos monolocutor

El grupo de datos que conforman el corpus piloto fué adquirido y cuantificado linealmente mediante un conversor A/D de 12 bits a una frecuencia de muestreo de 8533 Hz.

Cada palabra adquirida se sometió posteriormente a un submuestreo en el que se aplicó una ventana rectangular de longitud 256 muestras (20 mseg.) con una frecuencia de 66.6 Hz.. De cada ventana se extrajeron tres parámetros muy elementales: la amplitud media, la densidad de cruces por cero y la densidad de cruces por cero de la primera derivada de la señal (preénfasis) (ver [Casacuberta,87] para una descripción mas completa de estos parámetros), que permiten detectar con cierta seguridad segmentos fricativos y realizar una clasificación "tosca" de las vocales (figura 8.1).

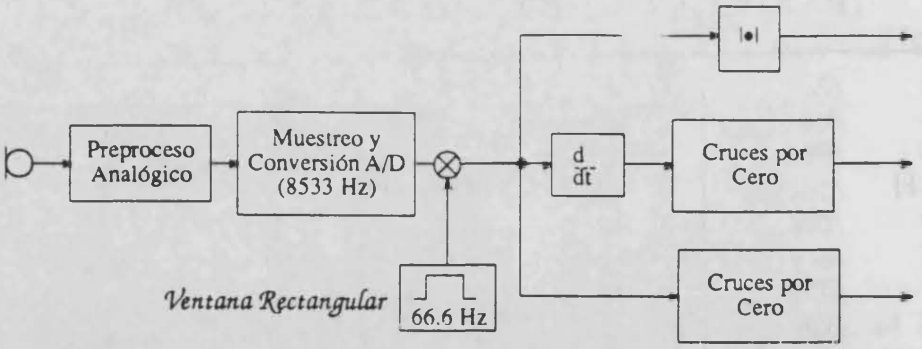


Figura 8.1 Adquisición y parametrización para los experimentos piloto: amplitud media, densidad de cruces por cero y densidad de cruces por cero de la derivada.

El conjunto de símbolos (aproximadamente) fonéticos era el mostrado en la tabla 8.3.

Tabla 8.3 Símbolos para las cadenas del experimento básico.

I	Vocal anterior
U	Vocal posterior
N	Sonora débil
S	Fricativa fuerte
Z	Fricativa débil
T	Oclusiva.

Cada ventana, descrita por estos tres parámetros se sometió a un procedimiento de etiquetado *difuso* (descrito en [Vidal,85]), para luego asignarle el símbolo dè la etiqueta difusa más evidente, o el símbolo "?" si ninguna era los suficientemente evidente.

Un único locutor pronunció (en varias sesiones) 58 repeticiones de cada uno de los dígitos. El corpus se compone por lo tanto de un total de 580 cadenas, habiendo resultado la longitud media de las mismas ser 43 caracteres (figura 8.2).

/CERO/	ZZZZZZIIIIIIIZIIIUUUIN??ZZTZZ
/UNO/	ZUUUUUUUN?NNIUUIIINNN?ZZZZ
/DOS/	ZIIIUUIUUUIISSSSSSSSSSZ
/TRES/	NIINIIIIIIIIIUINSSSSSSSSSSSSZ
/CUATRO/	ZZZUUUUUUUU?TTZNIIZ?NUUUNN??Z
/CINCO/	ZZZZZZZ?II?NNNNNNNN?TTT?ZIUIUUUINNN?ZZZZ
/SEIS/	?SSSSSSS?IIIIIIIIIIIIIIN?SSSSSSSSSSS
/SIETE/	ZSSSSSSSZIIIIIIIITTTTTZIIIIINNN?Z??
/OCHO/	ZZZ?IIUUUITTTTSSSSSIIIIIINNN?ZZZZ
/NUEVE/	ZZNNIIUUUUUUUUUUIIIINN?SZZZZ

Figura 8.2 Ejemplo de las cadenas utilizadas en el experimento básico de dígitos hablados (una por clase).

8.1.1.2 Corpus principal: Dígitos

El grupo de datos que conforman el corpus principal fue adquirido y cuantificado linealmente mediante un conversor A/D de 12 bits a una frecuencia de muestreo de 8533 Hz.

Cada palabra adquirida se sometió posteriormente a un submuestreo en el que se aplicó una ventana de Hamming [Casacuberta,87] de 30 msg. (128 muestras) con una frecuencia de 66.6 Hz. (128 muestras). De cada ventana se extrajeron 11 parámetros, 10 de ellos correspondían a 10 *coeficientes cepstrales* y el undécimo a la energía media en la ventana (convenientemente normalizada). En [Benedí,89] se dan los detalles del procedimiento para calcular los coeficientes cepstrales, que se resume en (figura 8.3):

- Obtención de la *transformada (rápida) de Fourier,* que proporciona 128 puntos complejos correspondientes a las frecuencias de 0 a 4267 Hz.
- Obtención del módulo de la transformada, elevando al cuadrado dichos valores complejos.
- Submuestreo frecuencial, mediante aplicación de 18 ventanas trapezoidales distribuídas según la *escala de Mel*[2].
- Codificación de cada uno de los 18 valores de este *banco de filtros,* a 8 bits con escala logarítmica.
- Análisis *cepstral* de estos 18 valores, es decir aplicación a ellos de una tranformada inversa de Fourier. La información contenida en cada uno de los 18 coeficientes así obtenidos decrece con su índice, reteniéndose los más significativos para el problema concreto: en nuestro caso los 10 coeficientes cepstrales.

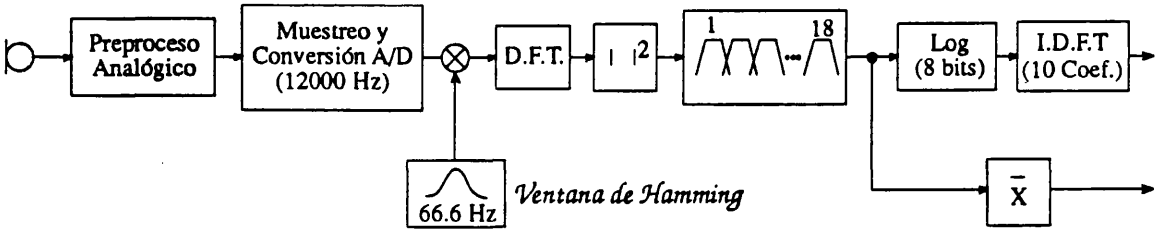


Figura 8.3 Esquema del proceso de adquisición y parametrización: obtención de los 10 Coeficientes Cespstrales y la Energía media.

[2]La escala de Mel es una escala, basada en el estudio de la membrana basilar del oído humano, aproximadamente lineal en las bajas frecuencias y logarítmica a medias y altas.

Cada ventana, descrita por estos once parámetros, se sometió a un etiquetado, clasificándola según la regla k-NN (k=7 vecinos) en la clase fonética más próxima de entre 15 clases. Para la definición de estas 15 clases "fonéticas" se recurrió a un algoritmo de agrupamiento no supervisado (*clustering* o *cuantificación vectorial*) del tipo C-Medias [Duda,73]. Así se obtuvieron 255 grupos a partir de un conjunto de vectores de parámetros (coeficientes ceptrales obtenidos como arriba descrito). Los prototipos (centroides) de estos 255 grupos se volvieron a agrupar, mediante el mismo algoritmo, en las 15 clases fonéticas mencionadas; convirtiéndose cada par (prototipo,clase) en uno de los representantes de la clase. Finalmente, mediante un procedimiento puramente manual y heurístico, se eligieron 15 símbolos para etiquetar cada una de las clases de acuerdo con sus características "perceptivas" acústico-fonéticas.

El conjunto de vectores para el agrupamiento estaba constituído por 4171 vectores extraídos de 150 palabras, las cuales corresponden a las primeras 5 repeticiones de cada dígito, pronunciadas por 3 de los locutores, 1 femenino y 2 masculinos (locutores 1,7,8). Estos locutores se escogieron porque no aparecían en las muestras de test y sí en las de aprendizaje en la mayoría de los experimentos.

Once locutores pronunciaron, en varias sesiones, 10 repeticiones de cada uno de los dígitos castellanos. El corpus se compone pues de un total de *1100 palabras*, que se distribuyeron de distintas maneras para realizar los experimentos. La longitud media de las cadenas es de 28,3 caracteres (figura 8.4).

/CERO/	TZZZZZvEEEEEEEErZEEvvoooUUT
/UNO/	UUUUUUUooNNNNvvvoooUUUUT
/DOS/	ZvvOOOOOOOvvvSSSSSSSSSSSST
/TRES/	ZrEErrEEEeveeeenrSSSSS
/CUATRO/	UUoUUOOOOOvNTTTZZZSZSZvooUUnTTT
/CINCO/	TZZrellllINNNNNNnTTTKTooooUUTTT
/SEIS/	SSSSSSSZrvEEEEEeeelInnnrSSSSS
/SIETE/	SSSSSSrIIIIEEEenTTTTZZeeeennTTT
/OCHO/	ToOOOOOvvTTTTZZSSSSevvUUUUTTT
/NUEVE/	TnNNvEOOvEEEEEvNNeeeeennnnnTT

Figura 8.4 Ejemplos de las cadenas del corpus principal de los dígitos hablados (una por clase).

Todos los experimentos son *multilocutor* o *independientes del locutor*, estando garantizada la variedad por la presencia de 5 voces femeninas y 6 masculinas.

8.1.1.3 Corpus difícil: letras

La parametrización para el corpus de letras siguió un procedimiento similar al de los dígitos (corpus principal), si se exceptúa la utilización de una frecuencia de muestreo mayor (133.3 Hz.), para disponer de una mayor finura de análisis de fonemas individuales (con una frecuencia de submuestreo de 66.6 Hz. es muy posible que una transición quede representada por un único símbolo), y un clustering que proporciona 32 símbolos en lugar de 15; estando todos estos cambios justificados por la mucho mayor dificultad de la tarea de reconocimiento planteada.

/F/	uUooJJJJJJJJJJeeeMVUUUUUUUUUUUUUUVeeeeooooeIIIAAAAAA
/L/	UoJJJJJJJJJJJJJJMRRReTTVTVMMooooooooIIIAAAAAuu
/LL/	UoJJJoeooooooooooMMVVVVVMMeMMMMooooooooooooooIIIAAAA
/M/	uUoJJJJJJJJJJJJJJeeeVVVVVVEEeJooooIIIIIIAAAAA
/N/	UoJJJJJJJoJJJooooeeIIISSSIIIIooooooooooooooIIIIAAAA
/Ñ/	uUoJJJJJJJJJJJooooeIIIIIIIIIIIIIZZFFIIIGGGGGoIIIIIAAAAA
/R/	KoJJJJJJJJoeJJJJJJJJoeVUUooooooooooooooIIIAAAuuuuuuu
/RR/	UJJJJJJJJJJJJJJJJJJJNNEXENNEENSSSSSEEEeeoeeeeIIIAAAAAuuuur
/S/	uIooJJJJJJJJJJJJeeMGHHHHHHHHHHHHHHHHPIIeeeeeIIIIIAA

Figura 8.5 Ejemplos de cadenas utilizadas en los experimentos de reconocimiento de letras habladas (una por clase del subconjunto de EE-Letras).

10 locutores (5 femeninos y 5 masculinos) pronunciaron 10 veces cada una de las letras, consiguiéndose de esta manera un corpus de *3000 palabras.*, representadas por cadenas de 56,8 símbolos de longitud media (figura 8.5).

8.1.2 Experimentos de reconocimiento

Para cada corpus se realizaron una serie de experimentos, ya partiéndolo de distintas maneras en conjuntos de aprendizaje y test, ya utilizando distintas variantes de ECGI (no estocástico, estocástico, sólo autorizando sustituciones: capítulo 7). Las variantes «Ignorando las frecuencias de los errores» e «ignorando las frecuencias de las reglas» se describen con detalle en el capítulo 9.

8.1.2.1 Corpus piloto

En todos los experimentos realizados con el corpus piloto se utilizaron las 38 primeras muestras de cada dígito como conjunto de aprendizaje, y las 20 siguientes como muestras de test. Los autómatas se infirieron con lel criterio minEL (ver apartado 6.6). Las tasas de reconocimiento de los experimentos efectuados, variando el tipo de algoritmo de reconocimiento fueron las mostradas en la tabla 8.4 (cada experimento implica el reconocimiento de 200 muestras).

Tabla 8.4 Resultados de los experimentos piloto (todos monolocutor, 380 cadenas de aprendizaje, 200 de test). En el capítulo 9 se detalla el procedimiento seguido para ignorar las frecuencias de las reglas y/o de los errores.

Algoritmo de Reconocimiento	%Aciertos
No estocástico	
Modelo de error completo	97,5
Sólo substituciones	96,5
Estocástico	
Modelo de error completo	99,5
Ignorando frecuencia de los errores	98,5
Sólo substituciones	99,5
Sólo substituciones, ignorando frecuencia de las reglas	99,5
Sólo substituciones, ignorando frecuencia de los errores	99
Número medio de estados	1063

A causa de la baja calidad de los parámetros, los resultados son inferiores a los que se obtienen con el corpus principal (ver exposición de éste más adelante), variando la diferencia entre los porcentajes de aciertos de 0 a -0.5 en el caso estocástico y de un -1.5 a -3 en el caso no estocástico (téngase en cuenta que los experimentos del corpus piloto son monolocutor). La relación entre los distintos tipo de reconocimiento se mantiene, discutiéndose en el apartado dedicado a los resultados del corpus principal.

En un apéndice se dan el tamaño de cada uno de los autómatas inferidos, que varía de 73 a 136 estados.

Como ejemplo visual de cómo distintos modelos representan realmente distintas estructuras, se adjunta la figura 8.6 con los 10 autómatas inferidos en con este corpus de datos.

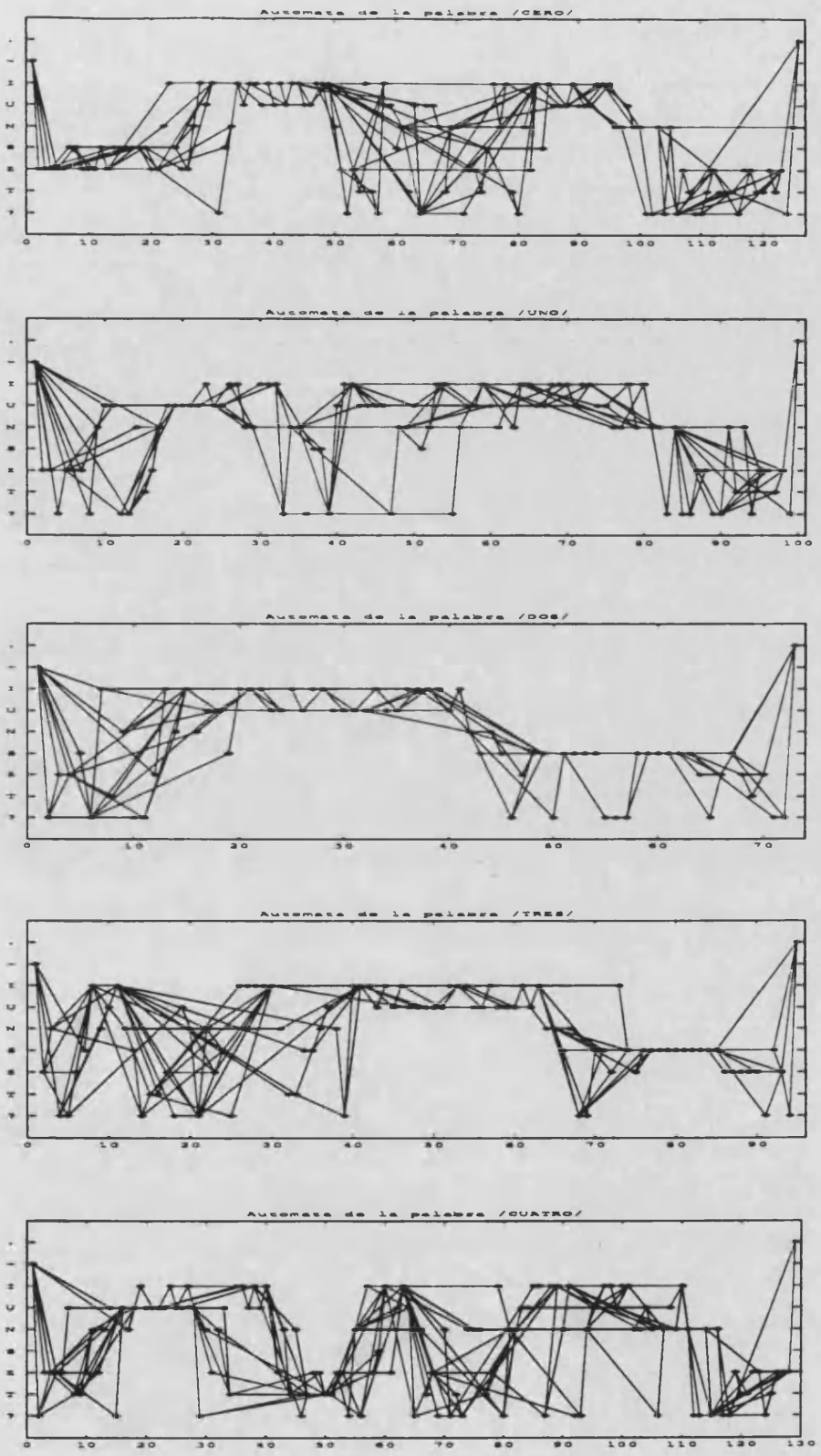


Figura 8.6 (a) Los 10 autómatas inferidos en el experimento piloto, del /cero/ al /cuatro/.

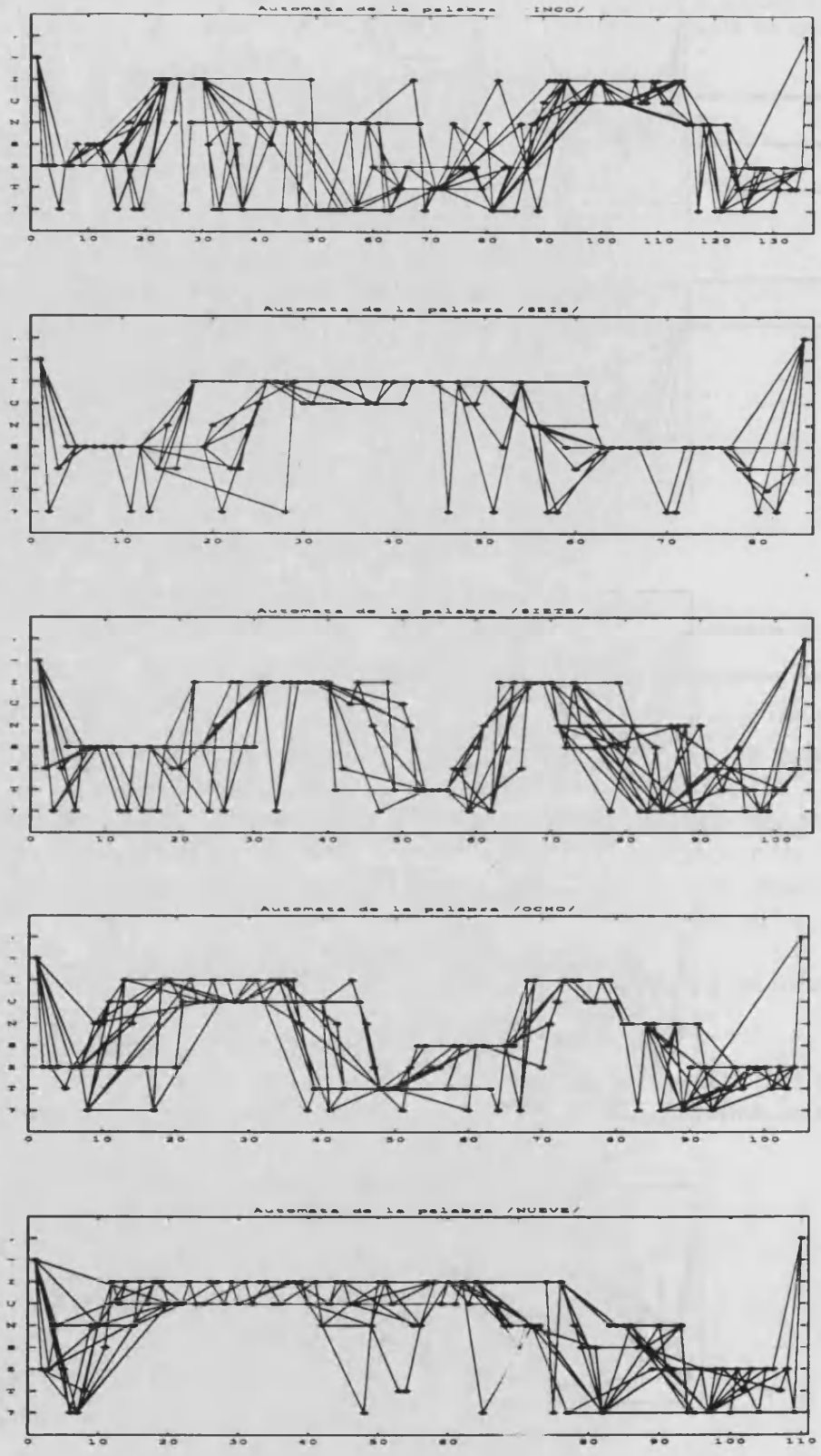


Figura 8.6 (b) Los 10 autómatas inferidos en el experimento piloto, del /cinco/ al /nueve/

8.1.2.2 Corpus principal

El corpus principal, de 1100 dígitos pronunciados por 11 locutores (numerados del 1 al 11), se utilizó en 6 experimentos sencillos y dos experimentos en los que se utilizó la técnica "*Leaving-k-out*" (cross-validation) [Raudys,91] para suplir la relativa pequeñez del corpus.

8.1.2.2.1 Experimentos sencillos

Los experimentos sencillos se resumen en la tabla 8.5. Se encuentra un experimento multilocutor (H2) y varios independientes del locutor. H1,H3,H4 pretender estudiar cómo afecta la mayor o menor cantidad de muestras de aprendizaje con respecto a las utilizadas posteriormente para el test de reconocimiento. H5, realizado posteriormente utiliza el reparto más adecuado con todas las muestras disponibles. H6 sustituye el locutor 10 de H5, que demostró estar adquirido en condiciones inadecuadas (saturación, palabras cortadas, etc...).

Tabla 8.5 Descripcción de los experimentos sencillos efectuados con el corpus principal de dígitos hablados. El número de muestras se da como locutores*repeticiones_locutor*clases. F,M son locutores femeninos y masculinos respectivamente.

Exp.	Aprendizaje	Test	Descripción
	Id. deLocutores, Sexo, NºMuestras	Id. de Locutores, Sexo NºMuestras	
H1	1,3,4,7,8 2F,3M 5*10*10=500	2,5,9,10 3F,2M $5*5_{(primeras)}*10=250$	Independiente del locutor, mitad y mitad de las muestras
H2	1,2,3,4,5,6,7,8,9,10 5F,5M $10*5_{(primeras)}*10=500$	1,2,3,4,5,6,7,8,9,10 5F,5M $10*5_{(siguientes)}*10=500$	Multilocutor
H3	1,2,5,6,7,8,9,10 4F,4M $8*6_{(primeras)}*10=480$	3,4 1M,1F 2*10*10=200	Independiente del locutor, más locutores en aprendizaje
H4	3,5,6,9 2F,2M 4*10*10=400	1,2,4,7,8,10 3F,3M $6*3_{(primeras)}*10=180$	Independiente del locutor, pocos locutores en aprendizaje
H5	1,2,4,7,8,10 3F,3M 6*10*10=600	3,5,6,9 2F,2M 4*10*10=400	Independiente del locutor, todas las muestras, locutor 10 mal adquirido.
H6	1,2,4,7,8,11 3F,3M 6*10*10=600	3,5,6,9 2F,2M 4*10*10=400	Independiente del locutor, todas las muestras.

Cada uno de estos experimentos se repitió con varios tipos de algoritmos de reconocimiento (no todos en todos los casos), proporcionanado la tabla 8.6.

Tabla 8.6 Resultadados de los experimentos sencillos de reconocimiento de dígitos hablados (% de aciertos). Corpus de 1000 dígitos, distribuídos diferentemente según los experimentos. Ver capítulo 9 para una descripcción de cómo se ignoran las frecuencias de los errores y/o de las reglas.

Algoritmo de Reconocimiento	H1	H2	H3	H4	H5	H6
No estocástico						
Modelo de error completo	–	–	–	–	98,5	99
Sólo substituciones	–	–	–	–	99,5	99,5
Estocástico						
Modelo de error completo	99,2	–	–	–	100	100
Ignorando frecuencia de errores	97,6	–	–	–	–	–
Sólo substituciones	98	100	100	98,8	99,7	99,8
Sólo substituciones, ignorando frecuencia de reglas	96,8	100	99,5	96,6	99,5	99,8
Sólo substituciones, ignorando frecuencia de errores	97,6	99,8	100	98.8	99	99
Número medio de estados	153	172	159	146	171	158

Se observa como la información que aporta la extensión estocástica simpre mejora el reconocimiento, bastando a veces con su aportación para alcanzar el 100% de reconocimiento.

En en caso no estocástico, es interesante comprobar que prohibir errores de inserción y borrado mejora el reconocimiento, llevándolo del 99% al 99,5%. Posiblemente debido a que (como indica el caso de utilizar sólo substituciones, pero ignorando las frecuencias de error en el modelo estocástico) (ver capítulo 9) las reglas de error incorporadas al autómata toman demasiada importancia si se les superpone las del modelo de error, a menos que se hallen presentes las probabilidades de las reglas para suavizar el efecto.

8.1.2.2.2 Leaving-k-out

Como es posible comprobar con el experimento H4 (40 muestras por autómata frente a 50..60 en los demás experimentos), una disminución de muestras en aprendizaje conlleva un empeoramiento notable de la efectividad de los modelos en aprendizaje.

Por otro lado, el número de muestras utilizadas en reconocimiento (500 en el mejor de los casos) tampoco es realmente suficiente para estimar la

capacidad de reconocimiento de ECGI. En [Duda,73] se estudia formalmente esta cuestión y se presenta una tabla en la que aparece el intervalo de confianza de una estimación del error medio real (y desconocido) p de un clasificador, en función del error estimado $\hat{p}$ (por máxima verosimilitud) y del número de muestras utilizadas para la estimación n. Aún en el caso de que $\hat{p}$ sea nulo (no hayan errores de reconocimiento), si se han utilizado 250 muestras, p se halla (con un 0.95 de probabilidad) entre el 0 y 0.6%. Con 1000 muestras todavía podría estar entre el 0 y 0.2%.

Para paliar el inconveniente de disponer tan sólo de un conjunto reducido de muestras, a la hora de estimar las capacidades de un reconocedor, se suele recurrir, en reconocimiento de formas, a la técnica de "leaving-k-out" [Raudys,91]. Esta técnica consiste en repetir varias veces los experimentos de reconocimiento intercambiando cada vez parte del conjunto de aprendizaje con parte del de test.

En nuestro caso se han repetido 5 veces un experimento de reconocimiento, intercambiando cada vez las muestras pertenecientes a 2 locutores (uno masculino y otro femenino). De esta manera, cada experimento sigue siendo independiente del locutor y se disponen cada vez de 800 muestras de aprendizaje (80 por autómata) y 200 de test, dando lugar a un conjunto de test efectivo de 1000 muestras. A lo largo de este trabajo hemos llamado HLKO11 este conjunto de experimentos (tabla 8.7).

Tabla 8.7 Experimentos "leaving-k-out" (HLKO11) para los dígitos hablados. El número de muestras se da como Locutores*muestras_locutor*clases. F,M son significan "femenino" y "masculino" respectivamente.

Exp.	Aprendizaje			Test		
	Locutores	Sexo	NºMuestras	Locutores	Sexo	NºMuestras
H11	2,4,5,6,7,8,9,11	4F,4M	8*10*10=800	1,3	1F,1M	2*10*10=200
H22	1,3,4,5,6,8,9,11	4F,4M	8*10*10=800	2,7	1F,1M	2*10*10=200
H33	1,2,3,5,6,7,9,11	4F,4M	8*10*10=800	4,8	1F,1M	2*10*10=200
H44	1,2,3,4,6,7,8,11	4F,4M	8*10*10=800	5,9	1F,1M	2*10*10=200
H55	1,2,3,4,5,7,8,9	4F,4M	8*10*10=800	6,11	1F,1M	2*10*10=200

Los resultados, utilizando el criterio minEL (definida en 6.6) durante la generación de autómatas, fueron los mostrados en la tabla 8.8.

Tabla 8.8 Resultados de los experimentos "leaving-k-out" (HLKO11) para los dígitos hablados (% de aciertos). Cada experimento involucra 800 muestras de aprendizaje y 200 de test.

Algoritmo de Reconocimiento	H11	H22	H33	H44	H55	Total
No estocástico						
Modelo de error completo	99,5	99	99,5	100	99,5	99,5
Sólo substituciones	99,5	99,5	99	100	99,5	99,5
Estocástico						
Completo	100	99,5	100	100	99,5	99,8
Sólo substituciones	100	99,5	100	100	99,5	99,8
Número medio de estados	181	200	199	186	198	192

El resultado final de 99,8%, presentado en la última columna de la tabla 8.8, representa tan sólo dos errores en 1000 operaciones de reconocimiento (reconocer /cuatro/ en vez de /ocho/ y /ocho/ en vez de /cuatro/).

Es muy notable el que, al aumentar las muestras de aprendizaje, los resultados del modelo de error completo y del obtenido prohibiendo los errores de borrado e inserción coinciden. Ello confirma que los modelos inferidos han ido incorporando los errores más frecuentes, e implica evidentemente un aumento de la complejidad espacial de los autómatas. Este aumento no es sin embargo excesivo, pues, en el peor de los casos, ha sido sólo de un 26% (comparando H4, de sólo 400 de muestras de aprendizaje y H22, de 800), y en el mejor 7.5% (H2, de 500 muestras, con H44).

Una idea de la estructura de los autómatas inferidos se puede extraer de la tabla 8.9 (todas las cantidades son medias para los 50 autómatas).

Tabla 8.9 Estadísticas (promedios) para los 50 autómatas inferidos en el experimento HLKO11 (Tamaño del lenguaje, Número de estados, Factor de ramificación, número de reglas, longitud promedio de las cadenas del corpus, longitud mínima, media y máxima de las cadenas aceptadas).

Tam. Len.	NºEstados	F.Ramific.	NºReglas	Long.Cad. aprendizaje	Long.Cad. Aceptadas min.	med.	max.
$6{,}25 \cdot 10^{14}$	192,8	1,97	387	28,3	15,4	19,9	55,1

El número de estados de los autómatas varía de 126 a 340 (ver apéndice B para detalles).

Los mismos experimentos, pero utilizando el locutor 10, demuestran que éste efectivamente es muy distinto a los demás (mala adquisición, comprobada examinando la señal en el dominio del tiempo), pues los resultados empeoran sensiblemente al estar este locutor en el conjunto de test (H55) (tabla 8.10).

Tabla 8.10 Resultados (% de aciertos) de los experimentos "leaving-k-out" (HLKO10) cuando se sustituye el locutor 11 por el 10 (mal adquirido).

Algoritmo de Reconocimiento	H11	H22	H33	H44	H55	Total
Estocástico						
Modelo de error Completo	100	99	100	100	97,5	99,3
Sólo substituciones	100	99	100	100	97,5	99,3
Número medio de estados	210	210	209	196	198	205

8.1.2.3 Corpus de Letras

El corpus de 3000 letras habladas se utilizó en 4 experimentos sencillos y uno utilizando la técnica de "Leaving-k-out", con la misma filosofía que la aplicada en el corpus principal de dígitos hablados.

8.1.2.3.1 Experimentos sencillos

De los 4 experimentos sencillos, los 3 primeros fueron multilocutor y el cuarto independiente del locutor. En los dos primeros se utilizó un subconjunto del corpus, una vez escogiendo un rango cualquiera de letras ('H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q') y la otra, el especialmente difícil conjunto de las EE-Letras. En los dos últimos se experimentó con las 30 clases posibles, una vez multilocutor y la otra independiente del locutor (ver tabla 8.11).

Tabla 8.11 Descripcción de los experimentos sencillos efectuados con el corpus de letras habladas. El número de muestras se da como locutores*repeticiones_locutor*clases. F,M son locutores femeninos y masculinos respectivamente.

Exp.	Aprendizaje Id. deLocutores, Sexo, NºMuestras	Test Id. de Locutores, Sexo, NºMuestras	Descripción
L1	1,2,3,4,5,6,7,8,9,10 5F,5M $10*8_{(primeras)}*9=720$	1,2,3,4,5,6,7,8,9,10 5F,5M $10*2_{(últimas)}*9=180$	Multilocutor, vocabulario: "H,I,J,K,L,M,N,O,P,Q" (9 letras)
L2	1,2,3,4,5,6,7,8,9,10 5F,5M $10*8_{(primeras)}*9=720$	1,2,3,4,5,6,7,8,9,10 5F,5M $10*2_{(últimas)}*9=180$	Multilocutor, vocabulario: "F,L,LL,M,N,Ñ,R,RR,S" (9 letras)
L3	1,2,3,4,5,6,7,8,9,10 5F,5M $10*8_{(primeras)}*30=2400$	1,2,3,4,5,6,7,8,9,10 5F,5M $10*2_{(últimas)}*30=600$	Multilocutor, vocabulario: Todas las letras. (30 letras)
L4	1,2,3,4,5,8 3F,3M 6*10*30=1800	6,7,9,10 2F,2M 4*10*30=1200	Independiente del locutor, vocabulario: Todas las letras (30 letras).

Los resultados, obtenidos utilizando el modelo de error completo y el criterio minEN (definida en 6.6), se resumen en la tabla 8.12.

Tabla 8.12 Resultados de los experimentos sencillos de reconocimiento de letras hablados (% de aciertos). Corpus de 3000 letras, distribuídas diferentemente según los experimentos

L1	L2	L3	L4
99	86,7	86	76,5

Se comprueba efectivamente la mucho mayor dificultad del conjunto de las letras y EE-letras, y el empeoramiento de los resultados en el caso independiente del locutor, debido en partes iguales al menor número de muestras de aprendizaje para cada autómata y a la mayor disimilitud de las muestras del conjunto de test.

8.1.2.3.1 Leaving-k-out

Para obtener una estimación del comportamiento de ECGI en un caso aún más difícil, se repitió el experimento L2 (EE-letras), pero en el caso independiente del locutor. Para aumentar la fiabilidad de los resultados se recurrió a la técnica de "leaving-k-out". Se realizó pues 5 veces el experimento, utilizando dos locutores en la fase de test (uno masculino y otro femenino) e intercambiando en cada experimento estos dos locutores con otros dos del conjunto de aprendizaje. Con ello se dispone en cada experimento de 720 muestras de aprendizaje (72 por autómata) y 180 de test (llamamos LLKO a este experimento), dando lugar a un conjunto de test efectivo de 900 muestras (tabla 8.13).

Tabla 8.13 Experimentos "leaving-k-out" (LLKO) para las letras habladas. El número de muestras se da como Locutores*muestras_locutor*clases. F,M son significan "femenino" y "masculino" respectivamente.

Exp.	Aprendizaje			Test		
	Locutores	Sexo	NºMuestras	Locutores	Sexo	NºMuestras
L11	3,4,5,6,7,8,9,10	4F,4M	8*10*9=720	1,2	1F,1M	2*10*9=180
L22	1,2,5,6,7,8,9,10	4F,4M	8*10*9=720	3,4	1F,1M	2*10*9=180
L33	1,2,3,4,6,7,9,10	4F,4M	8*10*9=720	5,8	1F,1M	2*10*9=180
L44	1,2,3,4,5,7,8,10	4F,4M	8*10*9=720	6,9	1F,1M	2*10*9=180
L55	1,2,3,4,5,6,8,9	4F,4M	8*10*9=720	7,10	1F,1M	2*10*9=180

El experimento se realizó utilizando el criterio minEN (definida en 6.6), con el modelo de error completo, ignorando las frecuencias de los errores, ignorando las frecuencias de las reglas y sin utilizar probabilidades. Las tasas de reconocimiento obtenidas se dan en la tabla 8.14.

Tabla 8.14 Tasas de reconocimiento (% aciertos) en el experimento "leaving-k-out" llevado a cabo con letras habladas (EE-letras). Sin información estocástica, ignorando las frecuencias de las reglas, de los errores y con el modelo de error completo.

Algoritmo de Reconocimiento	L11	L22	L33	L44	L55	Total
No estocástico						
Modelo de error completo	76,1	73,8	71,1	68,3	60	69,9
Estocástico						
Completo	77,7	76,1	76,6	72,2	66,6	73,8
Ignorando frecuencia de reglas	72,2	73,9	72,2	67,8	62,2	69,8
ignorando frecuencia de errores	76,7	77,2	70	70	62,8	71,3
Número medio de estados	831	784	797	763	749	785

Las tasas de reconocimiento mostradas en la tabla 8.14 evidencian una vez más la importancia que tiene para el buen funcionamiento de ECGI el utilizar la información estadística. Estas tasas, sin ser las que obtiene Bahl [Bahl,87] aplicando HMM al problema (similar) del E-set inglés (92%), son muy aceptables si se tiene en cuenta que no se han utilizados "símbolos continuos" (vectores de parámetros sin cuantizar vectorialmente) (Bahl –y Jelinek– obtienen un 79% con 200 símbolos discretos submuestreando a 100Hz). Utilizando el mismo corpus de EE-letras y HMM de 20 estados [Casacuberta,91] llega a obtener un 75%.

En la matriz de confusión media de los 5 experimentos se puede comprobar que la letra que da más problemas para su reconocimiento es la N (que se confunde con la M y Ñ), seguida de la L (que se confunde con la LL y R), lo cual es completamente consistente con nuestra apreciación perceptiva de la dificultad de la tarea (tabla 8.15).

Tabla 8.15 Matriz de confusión para el experimento LIO de reconocimiento de letras habladas (EE-letras). Los valores mostrados son acumulados para los 5 reconocimientos.

Letra	F	L	LL	M	N	Ñ	R	RR	S	%Clase
F	82	1	0	0	0	0	0	2	15	82%
L	0	55	16	5	1	2	14	7	0	55%
LL	0	15	78	0	0	5	1	0	1	78%
M	1	3	0	70	6	13	1	6	0	70%
N	1	5	0	24	43	21	6	0	0	43%
Ñ	2	2	5	3	1	87	0	0	0	87%
R	0	23	1	4	2	1	69	0	0	69%
RR	3	6	0	2	0	0	0	89	0	89%
S	11	0	0	0	0	0	0	0	89	89%
ToTal										**73,5**

Las cantidades que resumen la estructura media de los 45 autómatas se dan en la tabla 8.16.

Tabla 8.16 Estadísticas (promedios) para los 45 autómatas inferidos en el experimento LLKO. Tamaño del lenguaje, número de estados, factor de ramificación, número de reglas, longitud promedio de las cadenas del conjunto de aprendizaje, longitud mínima, media y máxima de las cadenas aceptadas.

Tam. Len.	NºEstados	F.Ramif.	NºReglas	Long.Cad. aprendizaje	Long.Cad. Aceptadas min.	med.	max.
$1{,}2 \cdot 10^{26}$	785	1,62	1271	56,8	26,8	51,8	130,3

La mayor cantidad media de reglas (3 veces más) de estos autómatas, si se les compara con los de los dígitos hablados, es debida no sólo a que las cadenas son el doble de largas en promedio, sino también a la mucha mayor variabilidad inducida por la existencia del doble número de símbolos.

8.2 Reconocimiento de imágenes planas

Aunque ECGI surgió durante la búsqueda de una solución para un determinado problema de reconocimiento del habla, fué evidente desde un principio que representaba una metodología aplicable a muchos otros campos del reconocimiento de formas.

Los siguientes experimentos se llevaron a cabo para demostrar esta independencia de ECGI de un campo de aplicación concreto, y pusieron en evidencia una vez más su eficacia como método de reconocimiento de formas.

Hay que hacer notar que los resultados obtenidos, y que se muestran a continuación, se han conseguido *sin ningún tipo de adaptación* de ECGI al problema concreto del reconocimiento de imágenes.

8.2.1 Representación simbólica de imágenes planas

La adquisición, parametrización y conversión en una cadena de las imágenes se realiza siguiendo el procedimiento esquematizado en el capítulo 1. Más concretamente, todos los experimentos se han realizado a partir de muestras de dígitos (arábicos) manuscritos o impresos, adquiridas mediante una cámara que proporciona una imagen de 512*512 pixels, con una resolución de 3 pixels/mm.

Una vez digitalizada la imagen a 8 bits/pixel, se aplica un umbral para reducirla a 1 bit/pixel, suprimiendo así los grises. Seguidamente un barrido de izquierda a derecha y de arriba abajo, detecta la parte superior izquierda de cada uno de los dígitos individuales. Este es el primer punto del contorno de dicho dígito. A continuación, se aplica un algoritmo de seguimiento de contornos [Freeman,74], que proporciona la dirección en la que se encuentra el siguiente punto del contorno, de entre las 8 posibles. La secuencia de direcciones, codificadas del 0 al 7 (figura 8.7), proporciona la cadena de símbolos correspondiente a la imagen.

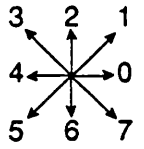


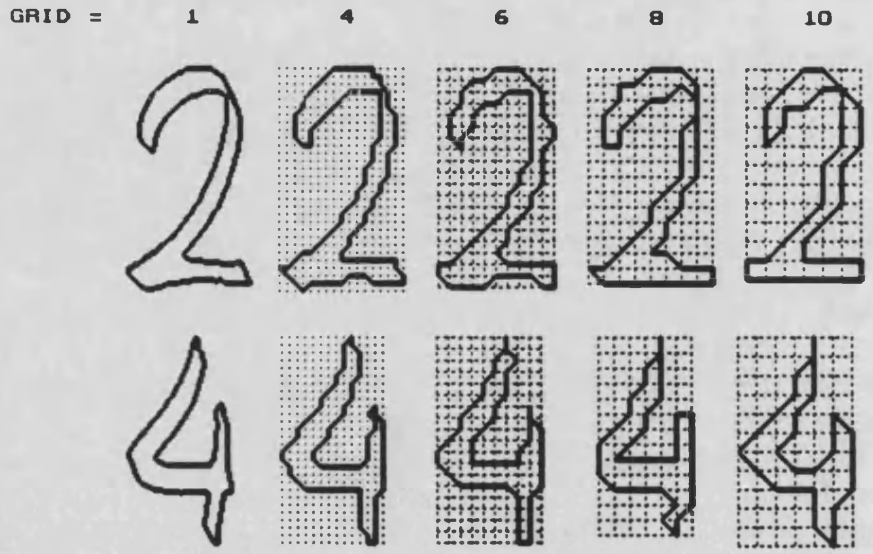
Figura 8.7 Símbolos para las cadenas de los corpus de dígitos manuscritos e impresos.

Los puntos aislados o puntos de "ruido" del contorno pueden suprimirse durante esta etapa. Evidentemente, esta codificación no tiene en cuenta los espacios huecos internos a la imagen del dígito, lo cual tiene fuertes repercusiones cuando no se cierra un trazado (ver ejemplos de dígitos manuscritos en la figura 8.9).

No se ha aplicado ningún procedimiento para asegurar la invarianza de la representación con respecto al giro y a la escala. La invarianza a la posición es intrínseca a la representación. Por lo tanto, se supondrá que ECGI es perfectamete capaz de tener en cuenta, gracias a su capacidad de generalización y corrección de errores, pequeñas variaciaciones de giro, inclinación y tamaño de los caracteres.

La resolución original de la imagen de 3 pixels/mm., proporciona cadenas largas, con gran cantidad de información probablemente redundante. Conocida la capacidad de ECGI de obtener resultados con muy poca información, se decidió disminuir esta resolución progresivamente,

mediante aplicación de 4 rejillas de separación respectivamente de 4, 6, 8 y 10 pixels sobre la imagen original (figura 8.8). Un objetivo añadido de los experimentos consiste pues en determinar la rejilla (resolución, o lo que es igual: longitud de las cadenas) máxima que nos proporcione resultados óptimos.



CHAIN CODES

```
2 (GRID 8) ->  00766666656565070064444444301112112221235455642212O1
2 (GRID 10)->  07766566556000644444420111221222454652211
4 (GRID 8) ->  66665653000022206666665631344432211212l2
4 (GRID 10)->  6665657012276665663224433111212
```

Figura 8.8 Imágenes de los dígitos manuscritos, reconstruídas a partir de las cadenas obtenidas con rejillas de distinta resolución. Algunas de las cadenas correspondientes a estos dígitos.

8.2.3 Dígitos Manuscritos

El corpus de los dígitos manuscritos está formado por 20 repeticiones de cada dígito, escritas por 12 personas diferentes (2400 dígitos en total).

Cada escritor, utilizando un rotulador grueso, escribió los dígitos en hojas de papel blanco, en las que ya figuraban unas líneas guía inferiores, muy ténues. La única condición impuesta a los escritores fue que los dígitos se escribieran separados unos de otros. No se mencionó ninguna restricción de tamaño ni estilo (aunque las líneas guía y el grosor del rotulador impidieran fantasías excesivas), observándose finalmente una variación 1:1/2 en tamaño y variaciones de inclinación del orden de -10 a +20 grados.

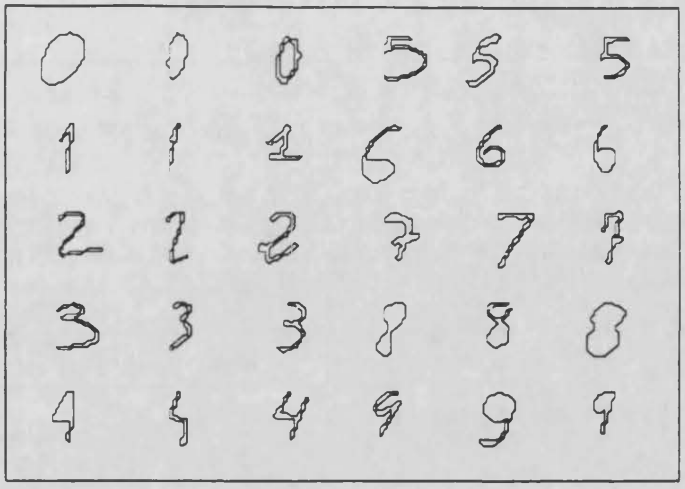


Figura 8.9 Imágenes escogidas para mostrar la variabilidad del corpus de dígitos manuscritos, reconstruídas a partir de las cadenas correspondientes a 6 pixels de resolución. Nótese el efecto de no cerrar algunos trazos (p.e. en los /8/).

8.2.3.1 Los experimentos

Al igual que en el experimento HLKO de los dígitos hablados, se recurrió a la técnica del "leaving-k-out". Se realizaron 12 experimentos, en cada uno de ellos se utilizaron las muestras de 10 de los escritores (2000 cadenas) para el aprendizaje y las de los dos restantes para test (400 cadenas). Por lo tanto, en todos los casos las cadenas de test pertenecían a escritores diferentes que los de las cadenas de aprendizaje. Cada experimento se diferencia del anterior por permutación circular del papel de cada uno de los escritores (tabla 8.17). De esta manera el total de operaciones de reconocimiento se eleva a 4800, habiéndose generado 120 autómatas a partir de 200 cadenas cada uno.

Tabla 8.17 Los 12 experimentos "leaving-k-out" con los dígitos manuscritos. El número de muestras se da como escritores*repeticiones*clases.

Exp.	Aprendizaje		Test	
	Escritores	NºMuestras	Escritores	NºMuestras
M1	3,4,5,6,7,8,9,0,A,B	10*20*10=2000	1,2	2*20*10=400
M2	1,4,5,6,7,8,9,0,A,B	10*20*10=2000	2,3	2*20*10=400
...	...		...	
M12	1,2,3,4,5,6,7,8,9,0	10*20*10=2000	A,B	2*20*10=400

8.2.3.2 Los resultados

Los autómatas se infirieron utilizando el criterio maxA (definido en 6.6). Cada experimento de "leaving-k-out" se repitió variando el algoritmo de reconocimiento; obteniéndose los resultados mostrados en la tabla 8.18.

Tabla 8.18 Tasas de reconocimiento (% aciertos) para el experimento "leaving-k-out" con dígitos manuscritos. Cada cifra coresponde a un total de 24000 muestras efectivas de aprendizaje (12 experimentos, en cada uno de los cuales se han utilizado 200 muestras para cada una de las 10 clases), y 48000 análisis sintácticos (12 experimentos con 10 autómatas a los que se presentaron 400 muestras).

Algoritmo de Reconocimiento	Rej4	Rej6	Rej8	Rej10
No estocástico				
Modelo de error Completo	92,3	92	91	89
Sólo substituciones	92,5	92	91	89,4
Estocástico				
Modelo de error Completo	98,4	98,3	96,9	96,0
Sólo substituciones	98	98,1	96,9	96,3

Vuelve a mostrarse la importancia de la información estadística. Se observa que la diferencia de precisión entre utilizar el modelo de error completo y prohibir inserciones y borrados es muy reducida, e incluso es a favor de utilizar sólo substituciones en el caso no estocástico.

Por otro lado, y como se esperaba, al aumentar la resolución de la rejilla la tasa de error disminuye. Sin embargo, la mejora obtenida al pasar de la rejilla de resolución 8 a la 6 es muy superior a la obtenida al pasar de la 6 a la 4, casi insignificante, por lo que no resulta útil emplear esta última.

En la tabla 8.19, se muestra la matriz de confusión media de los 12 reconocimientos efectuados para la rejilla 6 en el caso estocástico de sólo substitución en reconocimiento. La matriz correspondiente al modelo de error completo es muy similar (véase apéndice B para poder comparar las 4 rejillas).

Tabla 8.19 Matriz de confusión para el experimento con rejilla 6 y sólo sustitución de los dígitos manuscritos. Los valores mostrados son acumulados para los 12 reconocimientos.

Dígito	0	1	2	3	4	5	6	7	8	9	%Clase
0	474	0	0	0	0	0	3	0	3	0	98.7%
1	0	467	8	0	0	0	0	0	0	5	97.3%
2	0	0	480	0	0	0	0	0	0	0	100.0%
3	0	0	0	480	0	0	0	0	0	0	100.0%
4	0	1	0	0	465	0	0	4	0	10	96.9%
5	0	0	0	0	0	480	0	0	0	0	100.0%
6	0	0	0	0	0	0	480	0	0	0	100.0%
7	0	2	0	0	0	0	0	476	0	2	99.2%
8	0	0	12	6	1	0	1	0	453	7	94.4%
9	0	2	0	12	0	6	0	0	4	456	95.0%

Seguidamente se tabulan las tasas de error, para cada uno de los autómatas y cada uno de los 12 reconocimientos (R1 a R12) efectuados con las 4 rejillas, también cuando sólo se permiten substituciones en reconocimiento.

Tabla 8.20 Tasa de aciertos para los 48 reconocimientos de dígitos manuscritos realizados en el experimento estocástico con sólo substitución (12 experimentos leaving-k-out, R1 a R12, con 4 rejillas distintas).

Rec.	Rej4	Rej6	Rej8	Rej10	Promedio
R1	99,7	99,7	98,7	98,7	99,2
R2	98,7	99	98,5	98,2	98,6
R3	99,2	98,7	98,2	97	98,3
R4	100	99,2	99,5	96,5	98,8
R5	96	98	95,7	94,7	96,2
R6	93,5	95	92,7	90,7	93,7
R7	97,2	98	96,2	94,7	96,5
R8	98,2	97,7	97,5	98,7	98,3
R9	97	96,5	96,5	96,5	96,6
R10	98,7	98,2	97	97	97,7
R11	99,7	99	96,2	97,5	98,1
R12	98,7	98,5	96	95,2	98,1
Media	98	98,1	96,9	96,3	

Una idea de la estructura y dimensiones espaciales de los autómatas se puede extraer de la tabla 8.21.

Tabla 8.21 Dimensiones espaciales de los autómatas inferidos en los experimentos de dígitos manuscritos.

	Rej4	Rej6	Rej8	Rej10
Tamaño del Lenguaje	$5{,}6 \cdot 10^{60}$	$1{,}8 \cdot 10^{40}$	$1{,}0 \cdot 10^{31}$	$8{,}5 \cdot 10^{23}$
Número medio de estados	309	223	175	144
Factor de Ramificación	3,78	3,71	3,70	3,60
Nº de Reglas	1170	826	649	519
Longitud Cadenas Corpus	73,1	48,1	35,7	28,3
Longitud de cadenas Aceptadas				
mínima	22	16	13	11
máxima	182	120	91	71
media	80	52	39	30

8.2.4 Dígitos Impresos

El corpus de los dígitos impresos está formado por caracteres impresos con 8 diferentes tipos de letra. Cada tipo se ha impreso con 4 tamaños diferentes en el rango 1:1/2. Para cada tamaño de cada tipo se imprimió cada dígito 10 veces, 5 en negrita y 5 en grosor normal. Para cada tipo de letra hay por lo tanto 4*(5+5)*10=400 muestras, estando el corpus completo constituído por 3200 imágenes. En la figura 8.10 se muestra un ejemplo representativo de las mismas.

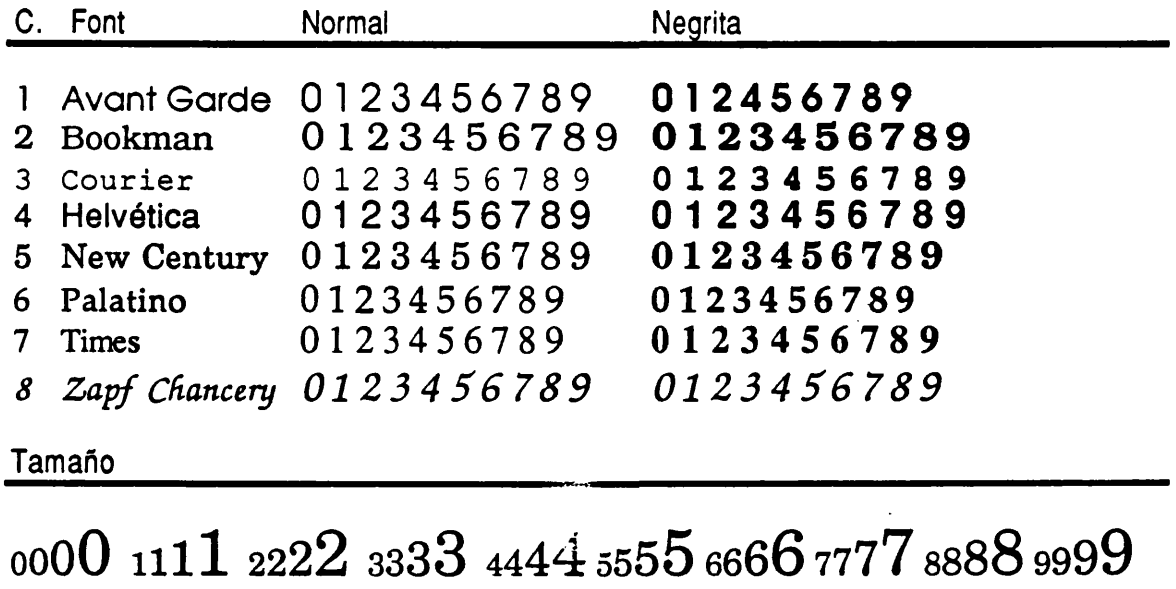


Figura 8.10 Tipos de letra (clases) utilizados en los experimentos de dígitos impresos. Los tamaños no son absolutos.

Por otra parte, debido a una presentación poco cuidadosa de las hojas de papel al sistema de adquisición, se observó una inclinación de los tipos (añadida a la natural de los propios tipos de letra) que variaba en un rango de ±10 grados (figura 8.11).

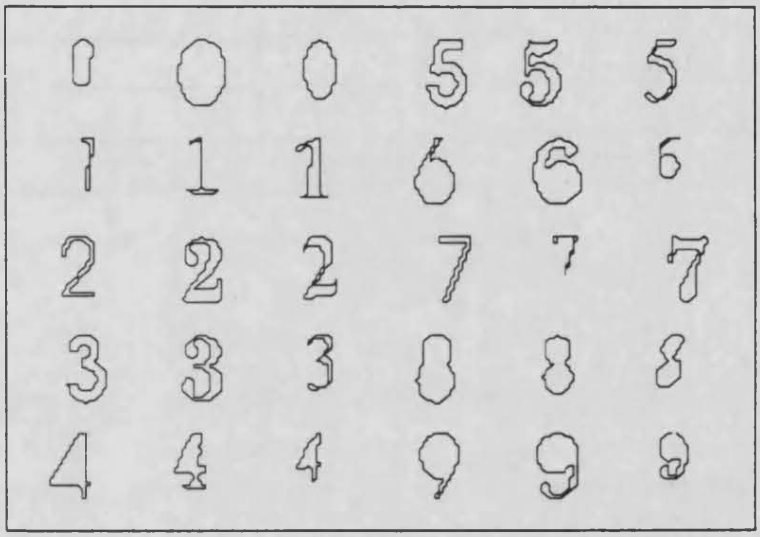


Figura 8.11 Imágenes escogidas para mostrar la variabilidad del corpus de dígitos impresos una vez adquirido y parametrizado. Dígitos reconstruídos a partir de las cadenas correspondientes a 6 pixels de resolución.

8.2.4.1 Los experimentos

Utlizando la técnica del "leaving-k-out" se realizaron 8 experimentos, en cada uno de ellos se utilizaron las muestras correspondientes a 7 tipos de letra para el aprendizaje (2800 cadenas), y la restante para el reconocimiento (400 cadenas). Por lo tanto, las cadenas de test pertenecían, en todos los casos, a un tipo de letra distinto que el de las utilizadas en aprendizaje. Cada experimento se diferencia del anterior por permutación circular del papel de cada uno de los tipos. De esta manera el total de operaciones de reconocimiento se eleva a 3200, habiéndose generado 80 autómatas a partir de 280 cadenas cada uno (tabla 8.22).

Tabla 8.22 Los 8 experimentos "leaving-k-out" con los dígitos impresos. El número de muestras se da como tipos*repeticiones*clases

Exp.	Aprendizaje		Test	
	Tipos	NºMuestras	Tipos	NºMuestras
I1	2,3,4,5,6,7,8	7*40*10=2800	1	1*40*10=400
I2	1,3,4,5,6,7,8	7*40*10=2800	2	1*40*10=400
...	...		...	
I8	1,2,3,4,5,6,7	7*40*10=2800	8	1*40*10=400

8.2.4.2 Los resultados

Los autómatas se infirieron utilizando el criterio maxA (definido en 6.6). Cada experimento de "leaving-k-out" se repitió con el modelo de error completo y prohibiendo inserciones y borrados; obteniéndose los resultados mostrados en la tabla 8.23.

Tabla 8.23 Tasas de reconocimiento para el experimento "leaving-k-out" con dígitos impresos. Cada cifra se ha obtenido con 22400 muestras de aprendizaje (8 experimentos en los que se utilizaron 280 muestras para cada una de las 10 clases) y corresponde a un total 32000 análisis sintácticos (8 experimentos en cada uno de los cuales se presentaron 400 muestras a 10 autómatas).

Algoritmo de Reconocimiento	Rej4	Rej6	Rej8	Rej10
Estocástico				
Completo	99,4	99,8	99,6	98,2
Sólo substituciones	99,9	99,4	98,9	97,7

De nuevo se observa que cuando la información es abundante para generar los autómatas, el prohibir las inserciones y borrados puede incluso llevar a mejorar la tasa de reconocimiento.

Con el modelo de error completo resulta ser ventajoso limitarse a la rejilla de resolución 6. Si sólo se autorizan sustituciones la variación de la rejilla 8 a la 4 es casi lineal, pudiéndose entonces considerar la rejilla 6, al igual que para los dígitos manuscritos, como un buen compromiso en caso de necesidad imperiosa de reducir la complejidad espacial y temporal.

Los resultados, como era de esperar, son mejores que los obtenidos en los experimentos equivalentes de dígitos manuscritos, situándose aproximadamente en un 1,5% en el caso de sólo substitución. Con el modelo de error completo, en las bajas resoluciones de la rejilla (8 y 10) se llega a más de un 2% de diferencia.

A continuación (tabla 8.24) se muestra la matriz de confusión media de los 8 reconocimientos efectuados para la rejilla 6 en el caso de sólo substitución en reconocimiento. La matriz correspondiente al modelo de error completo es muy similar (véase apéndice B para poder comparar las 4 rejillas).

Tabla 8.24 Matriz de confusión para el experimento de reconocimiento de dígitos impresos con rejilla 6 y con sólo substitución. Los valores mostrados son acumulados para los 8 reconocimientos.

Dígito	0	1	2	3	4	5	6	7	8	9	%Clase
0	319	0	0	0	0	0	0	0	1	0	99.7%
1	0	309	0	0	0	0	0	11	0	0	96.6%
2	0	0	320	0	0	0	0	0	0	0	100.0%
3	0	0	0	320	0	0	0	0	0	0	100.0%
4	0	1	0	0	320	0	0	0	0	0	100.0%
5	0	0	0	0	0	320	0	0	0	0	100.0%
6	0	0	0	0	0	0	316	0	4	0	98.8%
7	0	0	0	0	0	0	0	320	0	0	100.0%
8	1	0	0	0	0	0	0	0	319	0	99.7%
9	0	0	0	0	0	0	0	0	1	319	99.7%

Seguidamente se tabulan las tasas de error, para cada uno de los autómatas y cada uno de los 8 (R1 a R8) reconocimientos efectuados con las 4 rejillas, también cuando sólo se permiten substituciones en reconocimiento (tabla 8.25).

Tabla 8.25 Tasa de aciertos para los 8 experimentos R1 a R8 y las 4 rejillas (un total de 12800 reconocimientos), experimentos de sólo substitución con los dígitos impresos.

Rec.	Rej4	Rej6	Rej8	Rej10	Promedio
R1	99,75	99,75	96	94,25	97,43
R2	100	99,75	100	98	99,43
R3	100	100	99,25	99	99,5
R4	100	100	99	99,25	99,56
R5	100	100	100	96,75	99,18
R6	100	100	100	99,25	99,18
R7	100	100	100	99,25	99,8
R8	100	100	97	95,5	98,12
Media	99,97	99,44	98,91	97,75	

Donde se observa claramente que el tipo que más difícil es de reconocer es el que sirve de test en el primero de los 8 reconocimientos: el tipo AvantGarde (0123456789). Este tipo es, con el Helvetica (0123456789) el

único que no tiene *serif*, y tiene los 6 y 9 mucho más abiertos que todos los otros tipos. El siguiente tipo en dificultad es el ZapfChancery (*0123456789*), del cual se esperaba que fuera el más difícil por su estilo mucho más curvilíneo.

Una idea de la estructura y dimensiones espaciales de los autómatas se puede tener examinando la tabla 8.26.

Tabla 8.26 Dimensiones espaciales de los autómatas inferidos en los experimentos de dígitos impresos.

	Rej4	Rej6	Rej8	Rej10
Tamaño del Lenguaje	$3{,}3 \cdot 10^{45}$	$1{,}2 \cdot 10^{36}$	$5{,}4 \cdot 10^{26}$	$6{,}2 \cdot 10^{20}$
Número medio de estados	255	174	134	114
Factor de Ramificación	3,57	3,46	3,34	3,22
Nº de Reglas	911	602	449	369
Longitud Cadenas Corpus	72,8	47,9	35,5	28,1
Longitud de cadenas Aceptadas				
mínima	22	17	14	12
máxima	154	101	75	60
media	66	42	30	24

8.2.4.3 El uno sin base

Con exactamente la misma filosofía descrita en el apartado anterior, se repitieron todos los experimentos en los que sólo se autorizaba substitución y en los que se utilizaba el modelo de error completo, pero quitando del conjunto de muestras, una vez el ZapfChancery, y otra el Helvética. Los resultados se muestran en la tabla 8.27.

Tabla 8.27 Resultados de los experimentos de "leaving-k-out" para los dígitos impresos. Los mismos, pero suprimiendo del conjunto de datos el tipo ZapfChancery y el Helvética.

Algoritmo de Reconocimiento	Rej4	Rej6	Rej8	Rej10
Todos Los Tipos				
Completo	99,4	99,78	99,59	98,22
Sólo substituciones	99,97	99,44	98,91	97,75
Sin ZapfChancery				
Completo	99,93	99,75	99,43	98,50
Sólo substituciones	100	99,29	99,32	97,82
Sin Helvética				
Completo	99,36	99,75	98,82	98,50
Sólo substituciones	99,18	98,75	98,14	97,32

Donde resulta obvio que ZapfChancery es más dificultoso que Helvética, hasta el punto que no incluirlo en el conjunto de datos permite conseguir el 100% de aciertos, notablemente, en el caso de sólo tolerar errores de substitución, aunque, eso, sí con la rejilla de mayor resolución.

Por otra parte, resulta interesante comprobar cómo ECGI modeliza efectivamente las características más importantes de las formas consideradas. Examinando con detalle los resultados del experimento sin tipo Helvética, se comprueba que en *todos los casos* (con rejilla 4 y 6) se obtuvo un 100% de aciertos excepto en el primero de los 8 reconocimientos, el R1, en el que se obtuvieron las tasas mostradas en la tabla 8.28.

Tabla 8.28 Resultados del reconocimiento R1 para dos rejillas. Experimento sin Helvética de reconocimiento de dígitos impresos

	Rej4	Rej6
Completo	95,5	94,25
Sólo substituciones	98,25	91,25

Lo que evidencia que la única dificultad se presenta cuando el tipo AvantGarde esta en el conjunto de test. Del examen de las matrices de confusión (en apéndice B) se desprende que en un 71% de los casos el error es debido a confundir un 1 con un 7. Es decir, en vez de asignar (1) a la clase cuyo modelo se ha aprendido mediante (1 1 1111), se le asigna a la clase correspondiente a (7 7 7777), lo cual evidentemente es debido a dos obvias característica diferenciadora del (1) de AvantGarde: no tiene la "base" debida al serif, y la parte superior del uno es horizontal en vez de dirigirse hacia abajo. Ello es tanto más relevante, en cuanto el único error que se

produce en el experimento con todos los tipos (correspondiente al valor 99,97% de aciertos: 1 error de 3200 muestras, sólo substitución con rejilla 4) es debido a la misma confusión de (1) de AvantGarde con un 7 (véase matrices de confusión en el apéndice B), y ello aunque ECGI haya aprendido, gracias a Helvética que existen unos sin serif: (1) (figura 8.12).

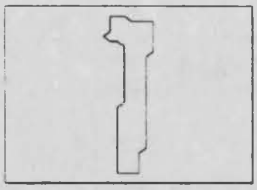


Figura 8.12 La única muestra no reconocida en el experimento de reconocimiento de dígitos impresos (Rejilla 4 y sólo autorizando errores de substirución).

8.2.6 Invarianza a la rotación y otras posibles extensiones

Como ya se expuso, al hablar del método de parametrización y conversión de imágenes planas a cadenas, la representación utilizada en los experimentos no es en absoluto invariante a la rotación ni al cambio de escala. Se ha comprobado, a través de los resultados obtenidos, que ECGI es capaz de adaptarse a *pequeñas* variaciones de escala y ángulo de presentación de los objetos, recurriendo a sus capacidades de generalización. No ocurre lo mismo si las variaciones de tamaño y de inclinación son muy amplias, pues ello llevaría a un crecimiento desmesurado de los modelos inferidos por ECGI, al tener éstos que dar cuenta de todas las posibilidades de presentación. Afortunadamente, una invarianza a la escala es fácilmente obtenible mediante una adaptación automática de la resolución de la rejilla.

Sin embargo, la invarianza a la rotación presenta mayores inconvenientes. Una posibilidad podría ser el utilizar códigos de cadena *relativos* [Bribiesca,80], en los que cada símbolo representa un ángulo *relativo al anterior* y no un ángulo absoluto como aquí se ha definido. Todos los contornos de un mismo objeto en distintas rotaciones, codificado de esta manera, proporcionará la misma *cadena circular*. Desafortunadamente, dependiendo del punto de corte se obtendrán diferentes cadenas lineales.

En casi todos los casos, en aprendizaje sigue siendo posible presentar los objetos patrón en una posición específica, lo cual permite cortar las cadenas circulares en un punto fijo (p.e. la parte superior izquierda) y aplicar continuación ECGI en aprendizaje como es usual. En reconocimiento, sin embargo, será necesario aplicar un proceso de comparación que tenga en cuenta la posible permutación circular de las cadenas.

También es posible el reconocimiento de dígitos trazados, en los que se adquiere directamente una secuencia de direcciones, la cual es mucho más significativa que la proporcionada por los contornos. Si se prescinde del problema de la invarianza a la rotación, ECGI es inmediatamente aplicable a este tipo de adquisición con sólo realizar una adecuada normalización de escala (y quizá algún filtrado).

Por otra parte, no es imprescindible representar las imágenes mediante contornos (pueden utilizarse esqueletos por ejemplo [Davies,81] [Zhang,84]), ni siquiera mediante modelos lineales. Es extremadamente sencillo extender los algoritmos que mediante corrección de errores realizan la comparación de una cadena con una gramática, es decir de una estructura lineal (la cadena) con una arbórea (la gramática), de manera que permitan comparar dos estructuras arbóreas. Así se podría conseguir que encontraran (p.e.) la cadena más similar entre dos gramáticas, o comparar una estructura arbórea (e incluso quizás una red en general) con otra (una estructura arbórea con una gramática de Web o de árbol [Fu,82]). Ello permitiría utilizar ECGI para inferir modelos de estructuras *no lineales*.

8.3 Experimentos de síntesis

Una posible aproximación, para comprobar hasta qué punto los modelos inferidos por ECGI modelizan las formas que se suponen deben representar, consiste en utilizar los autómatas inferidos en la *síntesis* de nuevas muestras (objetos), y compararlos "de visu" con objetos reales.

Ello es especialmente factible en el caso de imágenes, las cuales presentan además la ventaja —que no tendría un experimento de síntesis de la palabra— de proporcionar resultados fácilmente mostrables.

El algoritmo de síntesis consiste simplemente en un recorrido aleatorio de las reglas de la gramática: partiendo del axioma se llega hasta una regla final ($A \rightarrow a$), escogiendo en cada paso al azar la siguiente regla a aplicar. La decisión aleatoria tiene en cuenta las probabilidades de las reglas (pero no las de error: no se pretende añadir a las cadenas sintetizadas errores provenientes del modelo de error). Este proceso se ha utilizado para generar las muestras de las figuras siguientes, que presentan cada una 100 muestras. Las muestras (figura 8.13), en un primer caso (parte superior de la la figura) han sido generadas con los autómatas inferidos a partir del corpus de dígitos manuscritos, y en un segundo caso (parte inferior de la figura) a partir del de dígitos impresos.

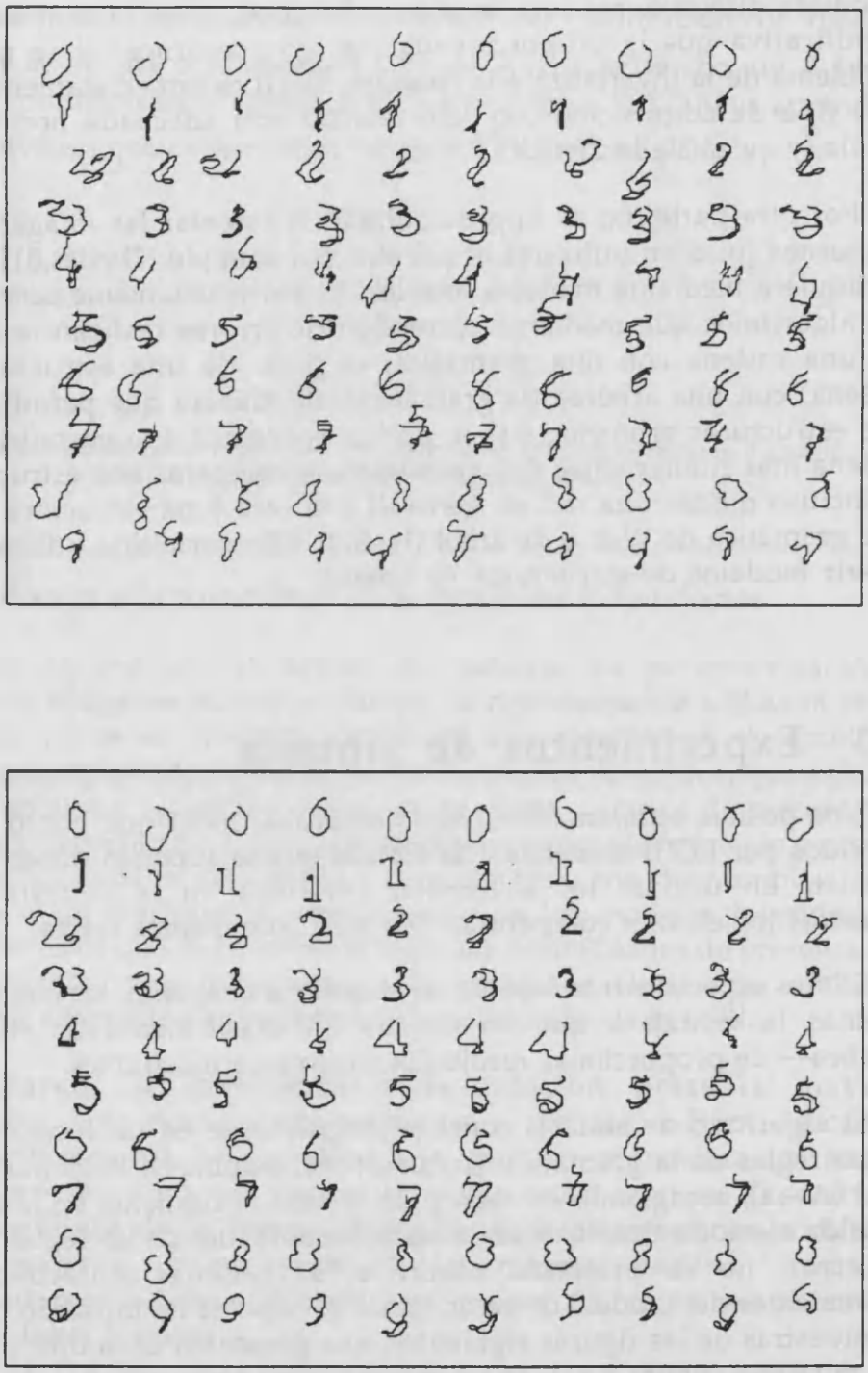


Figura 8.13 100 Muestras de dígitos manuscritos (arriba) e impresos (abajo) sintetizadas a partir de las gramáticas inferidas por ECGI. De arriba abajo, 1 fila de cada dígito, en el orden 0123456789.

Se observa que, en su gran mayoría, los dígitos sintetizados corresponden con la clase a los que se los asignaría intuitivamente. Si exceptúa unos pocos casos irreconocibles[3], la defomidad observada en mchos dígitos se debe en gran medida a que ECGI no es capaz de inferir la condición de cierre del contorno, lo que incluso le lleva a veces a intercambiar los contornos derecho e izquierdo.

8.4 Resumen de resultados

Del conjunto de resultados presentados a lo largo de todo este capítulo, se puede comprobar la gran eficacia que tiene el método ECGI en el reconocimiento de formas, por poco que el problema se adapte a los heurísticos que el método lleva implícitos.

En reconocimiento del habla, los resultados son similares o superiores a los proporcionados por otros métodos de reconocimiento de palabras aisladas, usualmente considerados como los que más eficaces en la práctica (modelos de Markov (HMM), alineamiento temporal (DTW),...) (ver capítulo 12). En tareas de reconocimiento de dígitos hablados (independientes del locutor, en ambiente no ruidoso) se consigue con ECGI tasas de aciertos superiores al 99% en todos los casos, llegando sin especial dificultad al 99,8% e incluso al 100% en algunos casos. Para diccionarios difíciles, como las letras, y especialmente las EE-letras, los resultados son también similares a los que proporcionan otras técnicas: 76,5% de aciertos para las 26 letras castellanas y 73.8% para las 9 EE-letras. Como punto de comparación, se puede mencionar que en [Casacuberta,91] se obtiene un 75% en este último corpus, mientras que en [Bahl,87] se obtiene (con HMM) un 92% reconociendo el E-set inglés, pero utilizando símbolos no cuantizados vectorialmente. Los resultados de [Bahl,87] bajan a un 79% si se procede a la cuantización, aunque se muestree a 100Hz., en vez de los 66.6Hz. de los experimentos aquí presentados. Por otro lado, ECGI consigue muy buenos resultados (99.5%, dependiente del locutor) incluso con cadenas formadas por símbolos muy poco elaborados y con relativamente pocas muestras de aprendizaje (38 por clase).

En reconocimiento de imágenes (en concreto dígitos manuscritos e impresos), los resultados son similarmente satisfactorios, realmente mejores que otros métodos más clásicos (98% frente a 80-90% obtenidos mediante la utilización de momentos geométricos y distancia de Mahalanobis [Vidal,92] [Vidal,92a], ver capítulo 12) y similares a las mejores que se obtienen

[3] Debería decir: ...que muestran inmundos garabatos indignos del escolar más travieso.

actualmente con métodos mucho más adaptados a la tarea específica (98-99% [Kurosawa,86] [Shridar,86] [Baptista,88]).

Por otra parte, los experimentos han mostrado la real y necesaria aportación que lleva a cabo la información estadística al reconocimiento (la inferencia ha sido en todos los casos no estocástica), ganando ECGI gracias a ella un 5% de promedio en la tasa de reconcimiento. También se ha evidenciado que es posible conseguir los mismos resultados (a veces mejores) incluso suprimiendo parte del modelo de error en reconocimiento (permitiendo tan sólo errores de substitución), lo que reduce fuertemente la complejidad temporal del reconocimiento y permite utilizar otros métodos que llevan a reducir dicha complejidad de hasta un 90% [Torró, 90].

La complejidad espacial (que afecta fuertemente a la temporal) de los modelos inferidos nunca llega a ser excesiva: de 400 reglas en tares sencillas (dígitos hablados) a 1300 en tareas difíciles (EE-letras habladas) y puede ser reducida por los métodos de simplificación que se presentarán en el capítulo 10. En la mayoría de los casos ECGI se comporta "razonablemente", pudiéndose comprobar que los errores de reconocimiento son debidos a muestras notablemente distintas de las utilizadas para el aprendizaje (locutor 11 en dígitos hablados, el uno de AvantGarde en dígitos impresos) o a clases que se prestan a confusión (N se confunde con M y Ñ, L se confunde con LL y R, en letras habladas, 1 se confunde con 7 en dígitos impresos).

Finalmente, se ha comprobado de manera muy visual, mediante experimentos de síntesis de imágenes, que los modelos inferidos por ECGI almacenan realmente suficiente información como para poder generar nuevos objetos de su clase, que evidencian todas las características que los diferencian de las otras clases.

9

ECGI: Estudios Empíricos

9.1 Introducción

Se abordan en este capítulo una serie de estudios sobre características de ECGI que se han decidido analizar empíricamente, ya sea porque su estudio teórico era excesivamente complejo, ya sea porque eran cuestiones cuya respuesta sólo tenía sentido en el contexto de una aplicación práctica:

- Variación en el comportamiento de ECGI en función del criterio de disimilitud utilizado.
- Convergencia de ECGI.
- Ambigüedad de las gramáticas generadas por ECGI.
- Necesidad o no de incluir las reglas de inserción y borrado en la fase de reconocimiento de ECGI.
- Importancia relativa de la información aportada por las distintas estadísticas que que utiliza ECGI estocástico (frecuencia de las reglas, de los errores, de ambas).

9.2 ¿Minimizar errores o maximizar aciertos?

Resulta difícil establecer en principio cuál de los criterios definidos en el capítulo 6.6 (minE: minimizar errores, minEL: minizar errores respecto a la longitud de la derivación y maxA: maximizar aciertos) es el más adecuado para ECGI en el caso más general.

Por otra parte, la comprobación experimental demuestra que los resultados de reconocimiento de ECGI no se ven afectados drásticamente por un cambio del criterio utilizado, por lo menos en los problemas aquí tratados.

Cada línea de la tabla que se muestra a continuación (tabla 9.1), corresponde a la utilización de ECGI con uno de los tres criterios estudiados en el capítulo 6. Cada línea corresponde a la aplicación de uno de los criterios al conjunto de experimentos de reconocimiento de dígitos manuscritos con resolución de rejilla 6 descrito en el capítulo 8. Como ya se explicó en dicho capítulo, cada experimento es en realidad un conjunto de 12 experimentos en los que se hallan involucrados un total de 120 autómatas (10 por experimento) y en los que se han llevado a cabo 48000 análisis sintácticos (400 muestras se han presentado a cada autómata). En el reconocimiento se ha utilizado el modelo de error restringido a solo sustituciones (capítulo 6 y 7).

Tabla 9.1 Comparación de los tres criterios estudiados. Se muestra el número medio de estados, el factor de ramificación, la talla y la longitud mínima,máxima y media de las cadenas del lenguaje generado y la tasa media de reconocimiento para cada experimento (dígitos manuscritos, con rejilla 6).

	Estados	**F.Ramif.**	**|L(G)|**	**Long.Cadenas** Min..Max : Med	**Resultados** (%Rec.) Peor..Mejor : Med
maxA	223	3,71	$1{,}8 \cdot 10^{40}$	11..145 : 69	95..99.75 : 98.15
minEL	271	2,37	$1{,}7 \cdot 10^{34}$	16..145 : 68	93..99.75 : 97.71
minE	475	1,94	$2{,}2 \cdot 10^{32}$	20..140 : 70	94..99.75 : 98.15

En la tabla se muestra el promedio del número de estados inferidos, del factor de ramificación (*Factor de Ramificación:* número de reglas con el mismo no terminal a la izquierda) y del número de cadenas de los lenguajes inferidos. Asimismo se muestra la longitud mínima, máxima y promedio de las cadenas de esos mismos lenguajes; finalmente, el % de cadenas correctamente reconocidas en el peor, mejor y en promedio de los 12 experimentos.

Los resultados, tal como sugería la discusión teórica del capítulo 6, se inclinan a favor de maximizar el número de aciertos. Se comprueba que la diferencia, en lo que se refiere a resultados de reconocimiento, es pequeña comparada con minEL (0.44%), pero a favor de maxA. MaxA mejora de un 2% el peor de los casos, no empeorando el mejor. Además, maxA proporciona automatas con un 18% menos de estados que minEL, aunque eso sí, con mayor número de reglas (23% más) debido a un mayor factor de ramificación.

Es muy notable el que la minimización de errores sin normalización proporcione los mismos resultados de reconocimiento que maxA, pero

generando automatas con un 100% más de estados (y un 10% más de reglas). Esta es la razón por la que este criterio no se haya utilizado en la práctica.

No cabe sin embargo afirmar que maxA proporcione siempre resultados mejores, aunque en general los mejora más que los empeora. Otro experimento de comparación de distancias se llevó a cabo repitiendo el reconocimiento en el peor (R6) y el mejor (R1) de los 12 casos (R1 a R12) del experimento anterior, pero esta vez utilizando un modelo de error completo en reconocimiento (tabla 9.2).

Tabla 9.2 Comparación entre maximizacion de aciertos y minimización de errores normalizando por la longitud de la derivación (%aciertos, ECGI estocástico con el modelo de error completo, y sólo permitiendo substituciones)).

	Sólo Substituciones		Modelo de error Completo	
	R6	**R1**	**R6**	**R1**
maxA	95	99,75	95.75	99.00
minEN	93	99,75	94.25	99.75

Aquí maxA proporciona en R1 resultados 0.75% peores, posiblemente debido a que, al producir un lenguaje mucho mayor (6 órdenes de magnitud), generaliza demasiado al superponer a las gramáticas el modelo de error completo, lo cual por otro lado es una ventaja si el aprendizaje es insuficiente (R6).

Para comprobar si la eficacia relativa de los distintos criterios dependía del problema abordado se repitió el experimento HLKO11 (dígitos hablados, ver capítulo 8), utilizando las tres distancias, también con el algoritmo estocástico de reconocimiento, pero una vez con modelo de error completo y otra prohibiendo inserciones y borrados. Aquí cada entrada de la tabla corresponde a 10000 análisis sintácticos (5 experimentos con 10 autómatas, 200 muestras presentadas a cada autómata) (tabla 9.3).

En el caso de los dígitos hablados, la tasa de reconocimiento se inclina ligeramente a favor de minEL, probablemente porque se requiere mayor generalización al haber mayor variabilidad en los datos.

Por otra parte, la consideración de la ambigüedad de las gramáticas generadas, también puede hacer dudar a la hora de elegir el criterio más conveniente (ver más adelante).

Tabla 9.3 Comparación de los tres criterios entre cadenas estudiadas. Se muestra el número medio de estados, el factor de ramificación, la talla del lenguaje generado y la tasa media de reconocimiento para cada experimento, con modelo de error completo y con sólo substituciones (dígitos hablados).

	Estados	Branch.	|L(G)|	Resultados (%Rec.) Peor..Mejo : Med
maxA	171	2,53	$1{,}3 \cdot 10^{18}$	
Reconocimiento con Modelo de Error Completo				99..100 : 99,6
Reconocimiento con Sólo Substituciones.				99..100 : 99,7
minEN	197	1,97	$6{,}2 \cdot 10^{14}$	
Reconocimiento con Modelo de Error Comp.				99,5..100 : 99,8
Reconocimiento con Sólo Substituciones.				99,5..100 : 99,8
minE	246	1,79	$3{,}3 \cdot 10^{13}$	
Reconocimiento con Modelo de Error Completo				98,5..100 : 99,6
Reconocimiento con Sólo Substituciones.				98,5..100 : 99,2

9.3 Convergencia del aprendizaje

De la propia definición de ECGI se observa que sólo se añaden reglas (y no terminales) cuando se produce la aparición de una (sub)cadena aún no modelizada (en esa posición) por la gramática inferida. De ello se deduce inmediatamente que **el tamaño de la gramática inferida por ECGI *no está limitado* a priori,** pues siempre es posible que aparezca una (sub)cadena no modelizada aún por la gramática.

Sin embargo, podemos asumir que todo conjunto de muestras real tiene una *variabilidad* limitada, puesto que, por hipótesis, pertenece al lenguaje de una gramática inferible. Es por lo tanto lícito suponer que, si ECGI es un método adecuado de inferencia para esa gramática, llegará un momento, en que ECGI no tenga que añadir prácticamente ningún estado más, al contener la gramática construída a la buscada. A ello puede contribuir además, el hecho empírico de que el tamaño del lenguaje inferido crece exponencialmente con el número de reglas añadidas

El problema de la convergencia del aprendizaje realizado por ECGI se presenta entonces más bien como el problema de determinar lo apropiados que son los heurísticos de ECGI para una tarea concreta.

El límite máximo de crecimiento de una gramática generada por ECGI viene dado por la inferencia de la gramática canónica de R_+, que ECGI generará en el caso (casi imposible) de que todas las (sub)cadenas muestra fueran totalmente dispares en posición y semejanza. En el caso más real, de

que la muestra sea muy inadecuada –p.e: proveniente de una gramática con muchos circuitos– ECGI puede no converger y hacer crecer la gramática con un incremento constante determinado por la variabilidad de R_+. Recuérdese además que, debido a que no genera circuitos, *ECGI no puede inferir gramáticas con cadenas de longitud arbitrariamente largas*, y que en todos los casos el número mínimo de estados que genera es igual al de la cadena más larga de R_+.

En un caso de aplicación práctica, adecuada a ECGI, se espera que la distancia de las nuevas muestras a la gramática inferida (y por lo tanto el crecimiento de ésta) se irá reduciendo con el número de muestras presentadas, tendiendo asintóticamente hacia un tamaño máximo. Esto se ha comprobado empíricamente en múltiples experimentos (todos los realizados hasta el momento con ECGI), y queda evidenciado en las figuras adjuntas (figuras 9.1 y 9.2), en las que se muestra la distancia de cada nueva muestra a la gramática en ese momento, junto con el número de reglas de la gramática. Obsérvese cómo en algunos casos aprarecen "picos" de variabilidad en el conjunto de muestras (especialmente claros en los dígitos impresos, en las que se cambia de tipo de letra cada 40 muestras), y como en otros se ha detenido la inferencia relativamente lejos de la asíntota (las letras habladas, dada su gran variabilidad).

Asimismo, en la parte inferior de las figuras se muestra 1) el número total de símbolos presentados a ECGI, o número de caracteres total de R_+ (que es equivalente al número de reglas de la gramática canónica; comparar con el número de reglas en la parte superior); y 2) la evolución del tamaño del lenguaje de la gramática (con el fin de cuantificar la amplitud de la generalización efectuada por el ECGI), ambas cantidades frente al número de muestras de aprendizaje.

La convergencia de ECGI puede llegar a ser extremadamente rápida, principalmente debido a que el lenguaje de la gramática inferida tiende a crecer exponencialmente (en paralelo con el número de posibles combinaciones de un número cada vez mayor de subcadenas incluídas). Ello, en un caso favorable le hace superar enseguida la posible variabilidad de las muestras. Ello permite a ECGI construir gramáticas muy compactas (un número de reglas un orden de magnitud inferior a la canónica, con 200 muestras) y de hecho, aún reducibles de hasta un 30% sin merma apreciable de resultados (ver la simplificación de autómatas, en capítulo 10).

Como demostración empírica de la compacidad de las gramáticas generadas por ECGI, se adjunta una tabla, correspondiente a la mayoría de los experimentos presentados con detalle en el capítulo 8 (tabla 9.3). Nótese que esta tabla es tan sólo un resumen de resultados ya presentados en dicho capítulo.

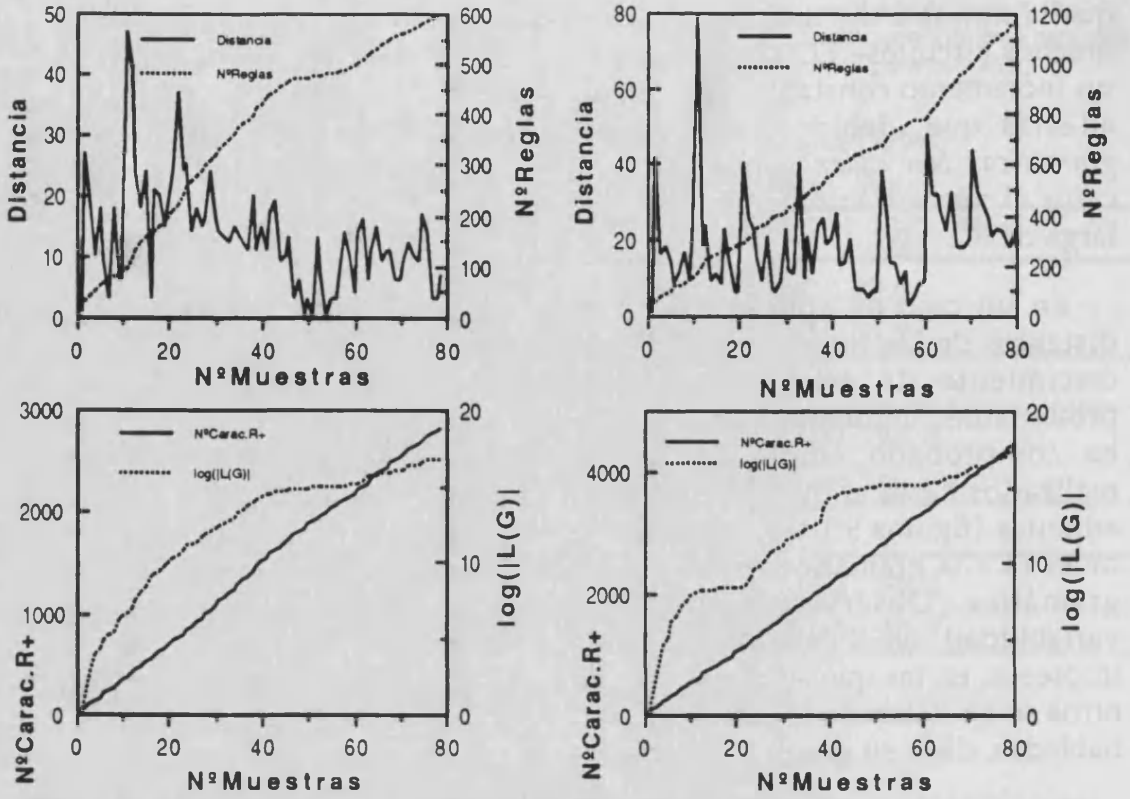


Figura 9.1 Convergencia y generalización de ECGI. Experimentos con Palabras Habladas, Dígitos (Izquierda) y Letras (Derecha) (**NºCarac.R$_+$** es el número *símbolos* presentados a ECGI).

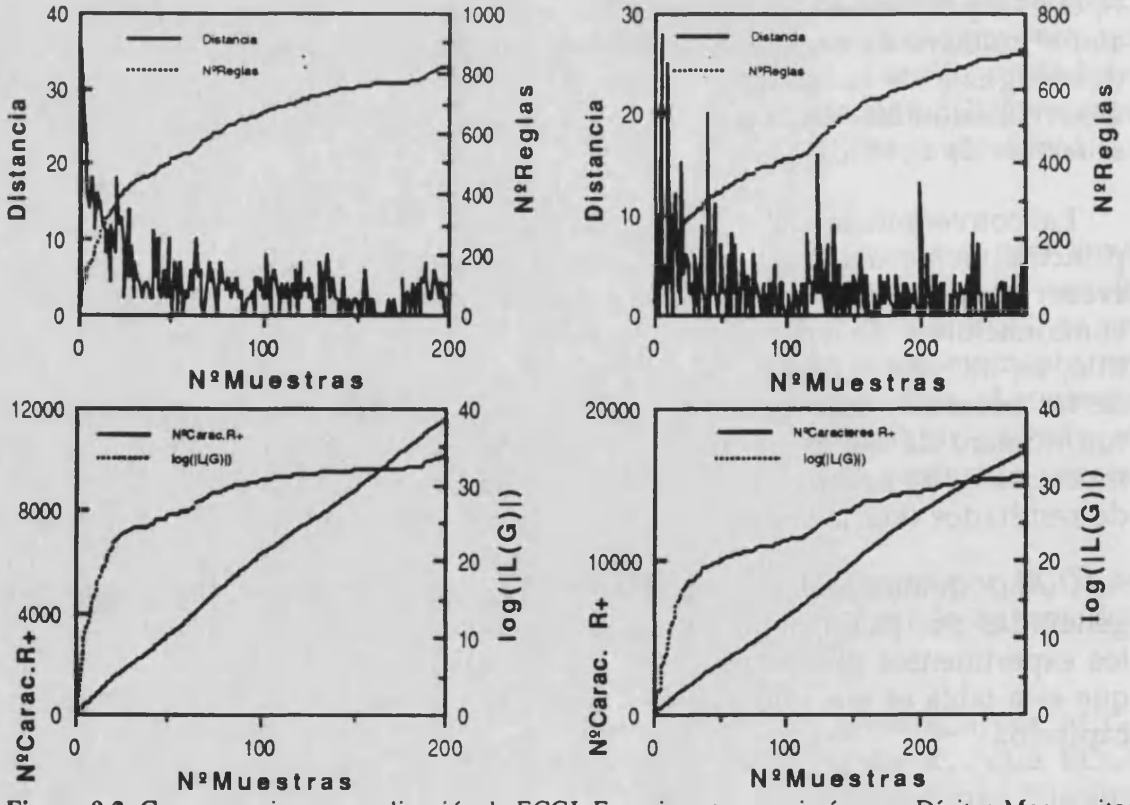


Figura 9.2 Convergencia y generalización de ECGI. Experimentos con imágenes, Dígitos Manuscritos (Izquierda) e impresos (Derecha) (**NºCarac.R$_+$** es el número de *símbolos* presentados a ECGI).

Tabla 9.3 Compacidad de las gramáticas inferidas por ECGI. Para cada experimento se dan la longitud media de las cadenas de aprendizaje, el número medio de estados y el factor de ramificación de los autómatas inferidos, la relación de éstos con la gramática canónica (número de reglas), y número de cadenas del lenguaje inferido.

Experimento	Longitud M. Cadenas	Estados / Autómata	Branching Factor	Relación con G. Canónica	|L(G)|
Dígitos Hablados (Muestras/autómata 80)					
HLKO11	28,3	196	1,97	1/5,8	$6{,}25 \cdot 10^{14}$
Letras Habladas (Muestras/autómata: 80)					
LLKO	56,8	785	1,62	1/4,5	$1{,}2 \cdot 10^{26}$
Dígitos Manuscritos (Muestras/autómata: 200)					
Rej4	73,1	310	3,78	1/12.,5	$5{,}6 \cdot 10^{70}$
Rej6	48,1	223	3,71	1/11,6	$1{,}8 \cdot 10^{40}$
Rej8	35,7	175	3,70	1/11	$1{,}1 \cdot 10^{31}$
Rej10	28,3	144	3,60	1/10,8	$8{,}5 \cdot 10^{23}$
Dígitos Impresos (Muestras/autómata: 140)					
Rej4	72,82	255	3,57	1/11,2	$3{,}3 \cdot 10^{45}$
Rej6	47,95	174	3,46	1/11,1	$1{,}2 \cdot 10^{36}$
Rej8	35,52	134	3,34	1/11	$5{,}4 \cdot 10^{26}$
Rej10	28,13	114	3,22	1/10,6	$6{,}2 \cdot 10^{20}$

Obsérvese como, cuando hay un número mayor de cadenas de aprendizaje, la diferencia con la gramática canónica aumenta (de 1/5 con 80 muestras, pasa a 1/11 con 200), muestra evidente de la convergencia de ECGI.

9.4 Ambigüedad de las gramáticas

Las gramáticas inferidas por ECGI son en general ambiguas, lo cual es poco deseable y, como se vió en el capítulo 7, presenta (entre otros) inconvenientes mayores para una estimación correcta de probabilidades en la extensión estocástica de ECGI.

Para comprobar hasta qué punto es importante la ambigüedad de las gramáticas que genera ECGI, se ha llevado a cabo una comprobación sistemática de (la mayoría) de los autómatas que se generaron en los distintos experimentos.

Dada la no existencia en la literatura de un algoritmo de complejidad aceptable para determinar con estrictamente la no ambigüedad de una gramática no determinista, se procedió a utilizar un método que permite

asegurar que la gramática es ambigua, e incluso si es "poco" o "muy" ambigua (es ambigua para relativamente pocas o muchas cadenas).

El método se basa en la comprobación de la existencia de más de un camino de derivación para cada una de las cadenas de un conjunto de muestras dado. Para ello se realiza un análisis sintáctico no determinista (Viterbi) y se detecta cuando dos derivaciones de la misma cadena llegan al mismo no terminal (vértice del trellis). Denominaremos *punto de ambigüedad* a aquellos vértices del trellis en los que se produzca esta situación.

La siguiente tabla da cuenta de los resultados obtenidos, utilizando como muestras las mismas que sirvieron de aprendizaje a cada autómata (tabla 9.4).

Tabla 9.3 Ambigëdad de las gramáticas generadas por ECGI. El número de puntos de ambigüedad es el número medio de veces en que dos derivaciones distintas de una misma cadena se han encontrado en el análisis sintáctico mediante el algoritmo de Viterbi.

Experimento	Puntos Ambigüedad	NºReglas	|L(G)|
Dígitos Hablados (Muestras/autómata 80)			
HLKO11	6,39	387	$6{,}25 \cdot 10^{14}$
Letras Habladas (Muestras/autómata: 80)			
LLKO	7,55	1272	$1{,}2 \cdot 10^{26}$
Dígitos Manuscritos (Muestras/autómata: 200)			
Rej4	353.9	1170	$5{,}6 \cdot 10^{70}$
Rej6	164	826	$1{,}8 \cdot 10^{40}$
Rej8	96,3	649	$1{,}1 \cdot 10^{31}$
Rej10	58,34	519	$8{,}5 \cdot 10^{23}$
Dígitos Impresos (Muestras/autómata: 140)			
Rej4	331,13	910	$3{,}3 \cdot 10^{45}$
Rej6	136,8	602	$1{,}2 \cdot 10^{36}$
Rej8	70,10	449	$5{,}4 \cdot 10^{26}$
Rej10	43,31	369	$6{,}2 \cdot 10^{20}$

No se ha encontrado *ningún* autómata que, habiendo sido generado por ECGI, no sea ambiguo.

El número de puntos de ambigüedad crece con la complejidad de la gramática, lo cual no es del todo inesperado. Nótese que el número de puntos de ambigüedad *no* es el número de derivaciones posibles de una misma cadena; estas están relacionadas con el número de puntos de

ambigüedad de la misma manera que el número de cadenas del lenguaje con el de estados; es decir, de una forma que puede llegar a ser exponencial.

Obsérvese también que la relación del número de reglas con el número de puntos de ambigüedad es muy superior en el caso de reconocimiento de palabras (60 en vez de unos 5 –varía de 3 a 9–). La razón de esto se halla en el criterio de (di)similitud utilizado. En efecto, se comprobó la ambigüedad en un experimento realizado con los distintos criterios (Dígitos manuscritos, rejilla 6) (tabla 9.4).

Tabla 9.4 Ambigüedad de los autómatas generados por ECGI con distintos criterios.

Distancia	Puntos Ambigüedad	NºEstados	Factor de Ramificación	NºReglas	|L(G)|
Dígitos Manuscritos, rejilla 6. (Muestras/autómata: 200)					
maxA	164	223	3,71	826	$1{,}8 \cdot 10^{40}$
minEL	31	270	2,37	642	$1{,}7 \cdot 10^{34}$
minE	20	476	1,9	921	$2{,}2 \cdot 10^{32}$

Sorprendentemente, la distancia maxA proporciona autómatas enormemente más ambiguos, posiblemente debido a que para generar un lenguaje de igual tamaño emplea la mitad de estados, y por consiguiente un factor de ramificación doble. Desde el punto de vista de la ambigüedad, el mejor compromiso es entonces minEL, aunque ya hemos visto que a veces proporciona resultados de reconocimiento inferiores.

Por otra parte, hay que notar que cuando se compararon los criterios maxA y minE, la efectividad de reconocimiento de ambos resultó equivalente, a pesar del hecho de que se estaba utilizando la extensión estocástica, y de que la precisión de la estimación de probabilidades de minE debió ser mucho mayor al ser la gramática mucho menos ambigüa. No parece por lo tanto que, en la práctica, la imprecisa estimación de las reglas afecte mucho a la eficacia de reconocimiento de ECGI. Esto se ve confirmado por los experimentos dedicados a generar autómatas deterministas (capítulo 11) y por otros trabajos que intentaron, mediante reestimación de probabilidades (reconsiderando sucesivas veces el conjunto de muestras), mejorar la eficacia de ECGI; objetivo que no fue alcanzado plenamente [Castaño,90].

9.5 Sólo Substituciones

En el capítulo 6 se introdujo la posibilidad de reducir la complejidad temporal del análisis sintáctico realizado por ECGI a cada muestra, mediante simplificación del modelo de corrección de errores utilizado. Como se

comentó entonces, dicha simplificación consite en prohibir los errores de inserción y borrado, permitiendo tan sólo los de sustitución. Ello no sólo reduce el número de reglas a considerar durante el análisis, sino que permite aplicar diversas técnicas que limitando el número de vértices del trellis visitados, permiten reducir hasta en un 90% la complejidad temporal [Torró,90].

Para comprobar hasta qué punto esta simplificación afecta a los resultados de reconocimiento, no queda más remedio que recurrir a la comprobación empírica. Esta se ha llevado sistemáticamente a cabo en todos los experimentos realizados y descritos previamente en el capítulo 8.

Tanto cuando se utiliza un modelo estocástico (ver capítulo 7) como cuando no, es posible observar que se produce un (pequeño) deterioro de los resultados al prohibir inserciones y borrados. Aunque, en algunos casos, se llega a perder hasta un 3,6%, normalmente se registra, o ninguna pérdida en la tasa de reconocimiento, o bien pérdidas y mejoras del orden de 0.3%, con cierto predominio de las pérdidas (aunque en un caso se ha obtenido hasta un 1% de mejora). Todo esto se resume en la tabla comparativa que se presenta a continuación (tabla 9.5) (solo se presentan los experimentos más significativos, ya que esta tabla es un resumen de resultados ya presentados en el capítulo 8).

Tabla 9.5 Comparación de tasas de reconocimiento (%) con sólo sustituciones y con el modelo de error completo, para algunos de los experimentos realizados.

Experimento	Solo Sustituciones	Completo	Relación
No Estocástico			
Piloto	98.0	97.0	-1.0
H5	98.5	99.5	+1.0
Estocástico			
H1	95.6	99.2	-3.6
HLKO11	99.8	99.8	0.0
Dígitos Manuscritos			
Rej4	8.1	98.4	-0.3
Rej6	8.1	98.3	-0.2
Rej8	6.9	96.9	0.0
Rej10	6.3	96.1	+0.2
Dígitos Impresos			
Rej4	100	99.9	+0.1
Rej6	99.3	99.7	-0.4
Rej8	99.3	99.4	-0.1
Rej10	97.8	98.5	+0.3

En los casos en los que se midió el tiempo (todos los experimentos de dígitos manuscritos e impresos) se pudo observar, como previsto (al reducirse de un tercio el número de reglas a examinar: de $3 \cdot |V|+1$ a $2 \cdot |V|+1$), una reducción sistemática del orden de un 30% en el tiempo de reconocimiento (tabla 9.6).

Tabla 9.6 Tiempos total de aprendizaje, y tiempo total de reconocimiento con sólo substituciones y con el modelo de error completo (horas CPU). Se muestran todos los experimento leaving-k-out de dígitos manuscritos e impresos. Los tiempos de reconocimiento corresponden a 48000 (dígitos manuscritos) o a 32000 (dígitos impresos) análisis sintácticos, correspondiendo los de aprendizaje a 24000 y 22400 respectivamente. Nótese que los dos grupos de experimentos se han realizado en máquinas diferentes.

Experimento	Aprendizaje	Solo Sustituciones	Completo
Dígitos Manuscritos (HP 9380)			
Rej4	17	21	36
Rej6	8	10	17
Rej8	5	6	10
Rej10	3	4	6
Dígitos Impresos (RS-6530H)			
Rej4	6.22	5.56	8.62
Rej6	2.90	2.60	4.03
Rej8	1.72	1.53	2.33
Rej10	1.17	1.05	1.58

Resumiendo la discusión, se puede afirmar que, en los casos de que sea importante el tiempo de reconocimiento (p.e. tiempo real), se puede recurrir al modelo de error con sólo sustituciones, con una pérdida mínima de eficacia.

9.6 ECGI estocástico y no estocástico

En los resultados presentados en el capítulo 8 (algunos de los cuales se resumen en la tabla 9.7) se evidenció la gran importancia que tiene, para las prestaciones en reconocimiento de ECGI, la información estadística incorporada en la extensión estocástica (capítulo 7). Concretamente, y como se puede comprobar en la tabla 9.7, añadir la información probabilística a las reglas mejora a veces de hasta un 7% la tasa de reconocimiento.

Tabla 9.7 Comparación de resultados de reconocimiento de ECGI estocástico con la versión no estocástica (en cada caso con modelo de error completo o prohibiendo inserciones y borrados).

	Modelo de Error Completo		Sólo Substituciones	
	No Estocástico	Estocástico	No Estocástico	Estocástico
Dígitos Hablados				
Piloto	97,5	99,5	96,5	99,5
H5	99,5	100	98,5	99,75
H6	99,5	100	99	99,8
HLKO11	99,5	99,8	99,5	99,8
Dígitos Manuscritos				
Rej4	92,3	98,4	92,5	98,0
Rej6	92	98,3	92	98,1
Rej8	91	96,9	91	96,9
Rej10	89	96	89,4	96,3

La extensión estocástica de ECGI no supone un aumento notable en la complejidad del reconocimiento. El trellis a calcular es idéntico, añadiéndose únicamente una suma y un acceso a tabla en cada punto del mismo (de hecho ello es menos costoso que una normalización subóptima por la longitud del camino). Sin embargo, en vistas a una posible utilización a tiempo real y con en el fin de tener una mejor comprensión del algoritmo, resulta interesante localizar la contribución aportada por cada elemento de la extensión estocástica, y estudiar hasta que punto afectaría a los resultados si se la ignorara. Dos son las simplifaciones que se consideran en los siguientes apartados:

- Ignorar las frecuencias de las reglas de la gramática no expandida.
- Ignorar las frecuencias de los errores en el modelo de error.

9.6.1 Ignorar la frecuencia de las reglas

La cantidad de información aportada por las probabilidades P(r) de las reglas (transiciones) se puede estudiar con sólo ignorarlas en el reconocimiento. Para ello, basta obtener las probabilidades de los errores como es normal para ECGI, y generar unas probabilidades ficticias para las reglas que den igual importancia relativa a todas las reglas asociadas a un no terminal.

Sea $r_k = A \rightarrow x_k B_k$ $k=1..m$ las m reglas asociadas a $A \in N$. Un conjunto de probabilidades para dichas reglas $p(r_k)$ que da igual importancia a cada una

de ellas y asegura la consistencia de la gramática carácterística viene dado por:

$$p(r_k)=\frac{1}{m}; \qquad L(r_k)= -\log(m); \qquad \text{y evidentemente: } \sum_{k=1}^{m} P(r_k) =1$$

Aplicando este artificio en una serie de experimentos con el corpus de las letras habladas (ver capítulo 8, experimento LLKO) se consiguieron los resultados mostrados en la tabla 9.8. Estos resultados se comparan (en la misma tabla) con los obtenidos aplicando el modelo estocástico completo y el modelo no estocástico, así como con la posibilidad de ignorar las frecuencias de los errores, descrita en el apartado siguiente (cada cifra de la tabla es un experimento en el que se han utilizado 9 clases - autómatas- y 180 muestras de test). Esta tabla, mostrada ya en el capítulo 8, permite comprobar que el no considerar la información aportada por la frecuencia de utilización de las reglas de no error de la gramática inferida, empeora la tasa de reconocimiento de hasta un 5%.

Tabla 9.8 Tasas de reconocimiento (% aciertos) en el experimento "leaving-k-out" llevado a cabo con letras habladas (EE-letras). Sin información estocástica, ignorando las frecuencias de las reglas, ignorando la frecuencia de los errores y con el modelo de error completo.

	No Estocástico	Estocástico Ignorando frecuencia		Estocástico
		de las reglas	de los errores	
L11	76,1	72,2	76,7	77,7
L22	73,8	73,9	77,2	76,1
L33	71,1	72,2	70	76,6
L44	68,3	67,8	70	72,2
L55	60	62,2	62,8	66,6
Media	69,9	69,8	71,3	73,8

9.6.2 Ignorar las frecuencias de los errores

Aplicando la misma filosofía que en el apartado anterior, es posible anular la información estadística aportada por el modelo de error y observar las consecuencias. Para ello, una vez estimadas las probabilidades de la gramática característica de la forma usual, es necesario sintetizar una tabla de substitución T_{ls} (ver capítulo 7) que dé una misma importancia a todos los errores de sustitución, a todos los de borrado y a todos los de inserción, determinando de alguna manera la importancia relativa no error/ substitución/ inserción/ borrado.

El heurístico utilizado para obtener los resultados que se muestran en la tabla 9.8, consistió en definir arbitrariamente, para cada símbolo $a \in V$, la probabilidad de no error $p_s(a \mid a)$ como 0.5 y repartir el resto de la probabilidad entre la sustitución $p_s(b \mid a)$ (por otro símbolo $b \in V$) y el borrado $p_b(a)$. A la inserción $p_i(b)$ se le da la misma probabilidad que a los otros errores.

$$n = |V|; \qquad p_s(a \mid a) = 0.5; \qquad p_s(b \mid a) = p_b(a) = p_i(b) = \frac{1-0.5}{n-1};$$

con lo que la consistencia en susbstitución y borrado se conserva:

$$\sum_{1}^{n-1} \frac{1-0.5}{n-1} + 0.5 = 1;$$

y se puede utilizar el procedimiento acostumbrado para obtener la probabilidad de inserción de cualquier símbolo (P_i).

También para este caso los experimentos muestran que la tasa de reconocimiento empeora de hasta un 6% al ignorar la información sobre las frecuencias de los errores.

10

Caminos y Simplificación de Autómatas

10.1 Introducción

En los capítulos anteriores se ha comprobado que la complejidad espacial de los modelos que infiere ECGI, sin ser excesiva, es suficientemente grande para que en algunos casos represente una penalización. En este capítulo se presenta una metodología de simplificación de autómatas, que autoriza reducir la complejidad espacial (y por lo tanto la complejidad temporal en reconocimiento) de los modelos inferidos en más de un 40%, sin merma apreciable en la eficacia del reconocimiento efectuado con dichos modelos.

El método de simplificación se basa tanto en la información estadística aportada por la extensión estocástica de ECGI, como en ciertas propiedades estructurales de los autómatas (gramáticas) inferidos: el *tráfico por estado* y/o el *tráfico por transición*. La definición de estos tráficos se basa directamente en en el concepto de *caminos en un grafo sin circuitos*, al cual se dedican los primeros apartados de ese capítulo.

10.2 Caminos en un grafo sin circuitos

En los siguientes apartados se presentan una serie de algoritmos que permiten un análisis de las propiedades estructurales de un grafo dirigido y sin circuitos (DAG: "directed acyclic graph" [Aho,73]), concentrándose principalmente de aquellas relacionadas con los caminos que van de un vértice a otro a través de dicho grafo. En todo lo que sigue se tratará con un grafo dirigido $G=(\mathcal{V},A)$; $A=\{(u,v): u\in \mathcal{V}, v\in \mathcal{V}\}\subseteq \mathcal{V}^2$.

10.2.1 Numero de caminos por vértice

Se define el **número de caminos** en G, que van desde el vértice p (*principio del camino*) a un vértice u; p,u∈ $\mathcal{V}$ como:

$$N_c(p,u) = \begin{cases} 1 & \text{si } (p,u)\in A \\ \sum_{\forall x\in \mathcal{V}:(x,u)\in A} N_c(p,x) & \text{si } (p,u)\notin A \end{cases}$$

Esta cantidad tiene exactamente el significado intuitivo correspondiente y permite, por ejemplo, calcular el número de cadenas *distintas* (si no hay ambigüedad) del lenguaje generado por una gramática libre de circuitos.

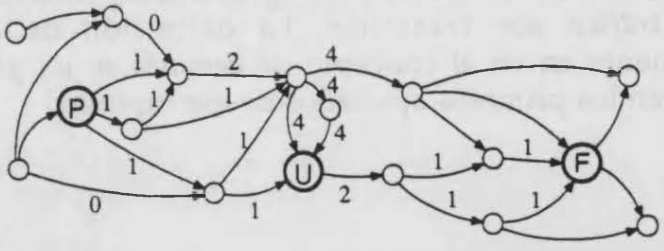
El **número de caminos inversos** en G, que llegan a p desde u (que van de u a p), se obtiene simplemente invirtiendo el sentido de las aristas y efectuando el cálculo al revés:

$$N_{ci}(u,p) = \begin{cases} 1 & \text{si } (p,u)\in A \\ \sum_{\forall x\in \mathcal{V}:(p,x)\in A} N_{ci}(u,x) & \text{si } (p,u)\notin A \end{cases}$$

Y obviamente: $N_c(p,u)=N_{ci}(u,p)$

El **Tráfico** entre p y f (*final del camino*) que pasa u; p,f,u∈ $\mathcal{V}$, es decir, el número de caminos en G que, yendo de p a f, pasan por u, se define simplemente como (figura 10.1):

$$T(p,u,f)=N_c(p,u)\cdot N_{ci}(u,f)$$



$N_c(P,U) = 9$

$N_{ci}(U,F) = 2$

$N_c(P,U) \cdot N_{ci}(U,F) = T(P,U,F) = 18$

Figura 10.1 Número de caminos de p a u, número de caminos inversos de f a u, tráfico que pasa por u y que va de p a f.

10.2.2 Número de caminos por arista

A partir de la definición del número de caminos que llegan a un vértice u de G provenientes de otro p, $N_c(p,u)$, se obtiene inmediatamente el **número de caminos que pasan por la arista** $t=(u,x)\in A$, viniendo del vértice $p\in \mathcal{V}$:

$$N_{ct}(p,t) = N_c(p,u)$$

En efecto: si hasta el vértice u se puede llegar por $N_c(p,u)$ caminos, todos ellos pueden a continuación pasar por cualquiera de las aristas originadas en u.

Similarmente, el **número de caminos inversos que pasan por la arista** t=(u,v), viniendo del vértice p será:

$$N_{cti}(t,p) = N_{ci}(v,p)$$

Nótese que **no** es cierto que si por la arista t=(u,v) pasan $N_{ct}(p,t)$ caminos provenientes de p, entonces pasan el mismo número de caminos inversos $N_{cti}(t,f)$ provenientes de f (véase ejemplo en figura 10.2):

$$N_{ct}(p,t) \neq N_{cti}(t,f)$$

El **tráfico por una arista** $t\in A$ que viene de los vértices p y va a f; $p,f\in \mathcal{V}$ (número total de caminos que van de p a f y pasan por t) se obtiene como:

$$T_t(p,t,f) = N_{ct}(p,t)\cdot N_{cti}(t,f)$$

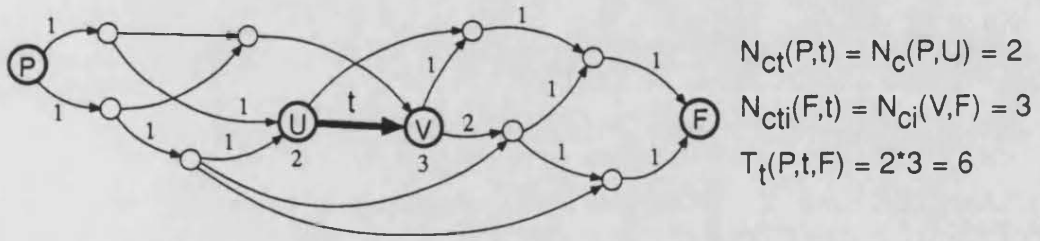


Figura 10.2 Tráfico por una arista t=(u,v).

Si $t_i=(u,x_i)\in A$, i=1..m son las m aristas que tienen por origen el vértice u, la relación entre el tráfico por las aristas y el nodo viene dada por:

$$\sum_{i=1}^{m} T_t(p,t_i,f) = T(p,u,f)$$

puesto que todos los caminos que llegan a u salen por alguna de las t_i.

10.2.3 Camino más largo y más corto

Para obtener el **camino más largo** en G que va de p a f, es decir, el que tiene el mayor número de vértices, se puede utilizar la recurrencia:

$$C_l(p,f) = \begin{cases} 1 & \text{si } (p,f)\in A \\ 1 + \max\limits_{\forall x\in \mathcal{V}:(x,f)\in A} C_l(p,x) & \text{si } (p,f)\notin A \end{cases}$$

y anotar en cada paso la decisión tomada (el vértice del que se viene). $C_l(p,f)$ es la *longitud del camino más largo.*

El camino en G, que va de p a f y que tiene el mayor número de *transiciones* es el mismo que $C_l(p,f)$, ya que el número de transiciones de un camino siempre es uno menos que el de vértices.

El **camino más corto** en G que va de p a f, es decir, el que tiene menor número de vértices, se obtiene con la misma recurrencia, sólo que minimizando en vez de maximizar. Escribiremos C_c la *longitud del camino más corto.*

10.2.4 Algoritmos no recursivos

Aplicando una típica transformación recursivo-iterativa es posible calcular eficientemente las cantidades definidas en los apartados anteriores, evitando recalcular cantidades ya obtenidas:

```
Algoritmo Caminos
Datos     G=(V,A)      /* Grafo */
          p,f∈V        /* vértices inicial y final */
Resultado      Total:ℛ
Auxiliar       sig:V→V   /* orden en V*/
Variables
   x,u∈V
   Acum:C_{V×ℛ}         /* vector indexado por V */
Inicialización
   u:=p; Acum[p]:=1
Método
   mientras u≠f hacer
      u:=sig(u); Acum[u]:=0
      ∀ x:(x,u)∈A hacer     /* predecesores de u */
         Acum[u]:= Acum[u] + Acum[x]
      fin ∀
   fin mientras
   Total:=Acum[f]
fin Caminos
```

Este algoritmo proporciona el número de caminos que llegan desde p a **todos** los vértices a los que llega un camino desde p.

Obsérvese que se ha utilizado el orden topológico del conjunto de vértices, que asegura que todos los predecesores de un estado son anteriores a él, lo cual siempre es posible si el grafo no tiene circuitos (ver capítulo 5: corrección de errores). Gracias a este orden se ha podido obtener el resultado con un único recorrido de los vértices.

El algoritmo presentado obtiene el número de caminos $N_c(p,f)$. El número de caminos inversos $N_{ci}(f,p)$ se calcula con el mismo algoritmo, pero sumando sobre los sucesores y empezando por f. Esto obligará en general a invertir la representación del grafo, puesto que usualmente se representan las aristas como una lista de predecesores (sucesores).

La obtención de C_l y C_c recurre al mismo algoritmo, solo que maximizando o minimizando en vez de sumar.

Obsérvese que el método utilizado para calcular el tráfico, que requiere un barrido de programación dinámica hacia delante ($N_c(p,u)$) y otro hacia detrás ($N_{ci}(u,f)$) de un grafo dirigido sin circuitos, está basado en la misma idea que el algoritmo Forward-Backward utilizado habitualmente para la estimación de probabilidades en modelos de Markov.

10.2.5 Relación Tráfico/Probabilidad

A partir del tráfico por vértice (arista), asumiendo la existencia en el grafo G de dos vértices privilegiados[1] $p,f \in \mathcal{V}$ y siendo $t_i=(u,x_i) \in A$, i=1..m las aristas que tienen por origen el vértice u, es posible definir la **probabilidad estructural de una arista** t_i:

$$P_E(t_i) = \frac{T_t(p,t_i,f)}{T(p,u,f)};$$

de esta definición, dada la relación entre $T_t(p,t_i,f)$ y $T(p,u,f)$, es obvio que se cumple:

$$\sum_{i=1}^{m} P_E(t_i) = 1$$

es decir, estas probabilidades son consistentes.

---

[1] Toda la discusión que sigue considera las propiedades del grafo únicamente en relación con los caminos que van de p a f.

Proposición: Si $\mathcal{C}_{pf}$ es el conjunto de caminos de p a f en G, definimos la **probabilidad estructural de un camino** $C \in \mathcal{C}_{pf}$, formado por la secuencia de k aristas $t_1,t_2..t_k$; como:

$$P_E(C)=\prod_{j=1}^{k} P_E(t_j)\,; \qquad \text{entonces} \qquad \sum_{\forall C \in \mathcal{C}_{pf}} P_E(C) = 1$$

En efecto, si el camino pasa por los nodos $p,u_1,u_2,\ldots,u_{k-1},f \in \mathcal{V}$, su probalidad estructural será :

$$P_E(C)=\frac{T_t(p,t_1,f)}{T(p,p,f)} \cdot \frac{T_t(p,t_2,f)}{T(p,u_1,f)} \cdot \ldots \cdot \frac{T_t(p,t_k,f)}{T(p,u_{k-1},f)} =$$

$$\frac{N_c(p,p)\cdot N_{ci}(u_1,f)}{T(p,p,f)} \cdot \frac{N_c(p,u_1)\cdot N_{ci}(u_2,f)}{T(p,u_1,f)} \cdot \ldots \cdot \frac{N_c(p,u_{k-1})\cdot N_{ci}(f,f)}{T(p,u_{k-1},f)} =$$

$$\frac{1\cdot T(p,u_1,f)\cdot \ldots \cdot T(p,u_{k-1},f)\cdot 1}{T(p,p,f)\cdot T(p,u_1,f)\cdot \ldots \cdot T(p,u_{k-1},f)} = \frac{1}{N_c(p,f)} \qquad \textbf{c.q.d.}$$

Es decir, la probabilidad estructural de cualquier camino de p a f es:

$$P_E(C) = \frac{1}{N_c(p,f)}\,; \quad \forall C \in \mathcal{C}_{pf}$$

Esto implica que una estructura de este tipo induce una distribución de probabilidades entre las aristas y a los nodos, incluso aunque no se asignen explícitamente dichas probabilidades. La probabilidad estructural proporciona la importancia relativa de cada una de las aristas que tienen origen en un nodo determinado en la generación de los caminos que van de un punto a otro de la estructura.

De la anterior proposición también es posible deducir que si, en un autómata estocástico sin circuitos, todas las probabilidades de las transiciones son iguales a las probabilidades estructurales de las aristas del grafo que representa el autómata, entonces todas las cadenas del lenguaje del autómata tienen la misma probabilidad. Eso sí, si el autómata es ambiguo una cadena podrá ser tanto más probable cuanto más caminos en el autómata –en su grafo– reconozcan dicha cadena.

Una medida de la importancia de una arista t y de un vértice u, relativa al número de caminos puede definirse de la forma:

$$I_E(t)=\frac{T_t(p,t,f)}{N_c(p,f)}; \qquad I_E(u)=\frac{T(p,u,f)}{N_c(p,f)}\,;$$

Que llamaremos **importancia estructural** de t y u, y que indican la importancia que tienen dicha arista y vértice en la generación de los $N_c(p,f)$ caminos en G (figura 10.3) . Nótese que:

$$P_E(t) = \frac{I_E(t)}{I_E(u)} \qquad \text{si } \exists\, x \in \mathcal{V}: t=(u,x)$$

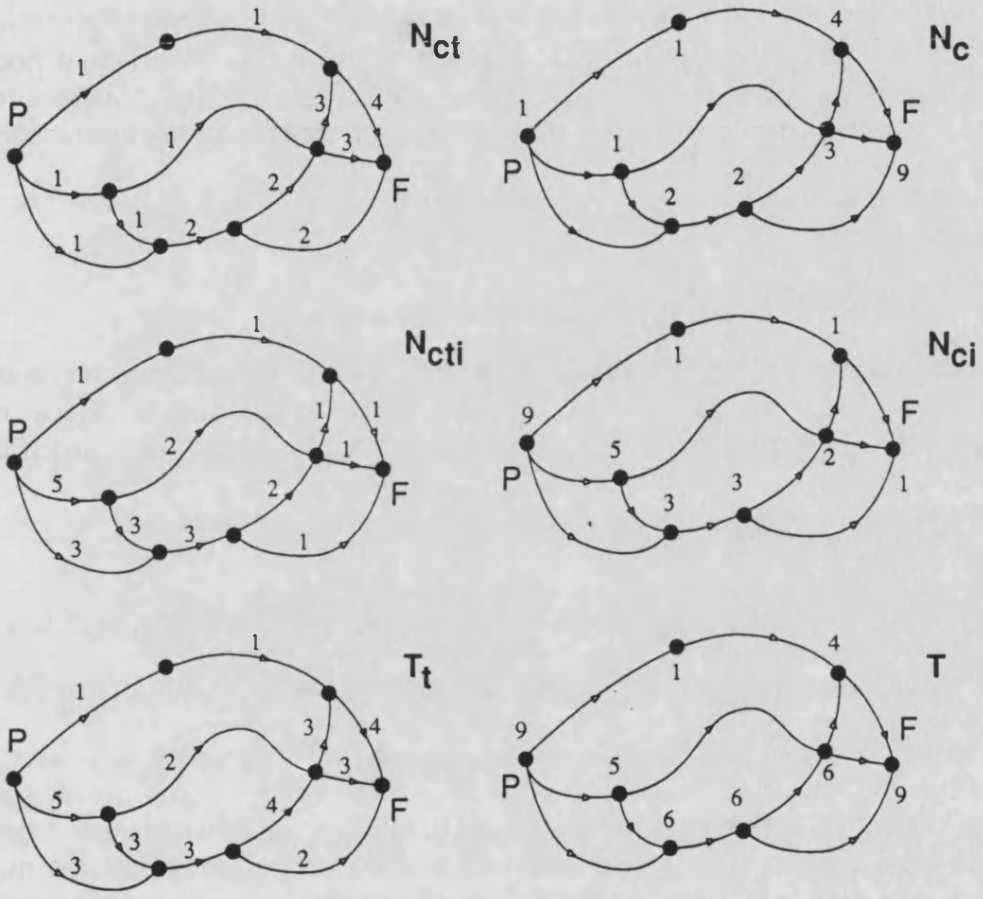


Figura 10.3 De arriba abajo: número de caminos, número de caminos inversos y tráfico por arista (a la izquierda) y vértice (a la derecha) en un grafo con vértices principio (p) y final (f). La importancia estructural por arista y vértice son proporcionales a los tráficos.

10.3 Simplificación de autómatas

Tal como se expuso, en el apartado dedicado a su convergencia, el método ECGI infiere en general autómatas muy compactos (un orden de magnitud más reducidos que el autómata canónico). Sin embargo, ello sigue implicando una gran cantidad de información (a veces hasta 300 estados por autómata), costosa de utilizar en los análisis sintácticos requeridos en la fase de reconocimiento.

Por otra parte, durante el reconocimiento, ECGI nuevamente superpone a la gramática inferida un modelo de error, siendo más que probable que parte de éste duplique parte del que se ha incluído durante la inferencia de la misma estructura. Además, como se ha comprobado durante la exposición de los heurísticos en los que se fundamenta ECGI, ocurre a veces que se incluyan en la estructura estados y transiciones redundantes.

Todo ello induce a pensar que es posible suprimir cierta parte de los estados y transiciones del modelo inferido, sin por ello mermar su poder de reconocimiento. En esta línea, exponemos a continuación una serie de técnicas que se pueden utilizar para este fin y mostraremos cómo con ellas se ha conseguido reducir los autómatas hasta un 30%.

10.3.1 Método

El algoritmo utilizado para la *simplificación* de los autómatas generados por ECGI puede resumirse en una frase: "quitar cada vez el estado menos significativo y repetirlo hasta que se haya conseguido la simplificación requerida".

Evidentemente, ello no es tan simple como parece:

- Hay que definir lo que se entiende por "estado menos significativo"
- Existen varios criterios posibles para decidir cuando la simplificación es suficiente.
- Al suprimir un estado es necesario asegurarse de que no se modifica la calidad de "estado menos significativo" o reestimarla. Asimismo hay que realizar la supresión en cadena de aquellos estados que han quedado desconectados (sin predecesores y/o sucesores) al quitar otro.

Una herramienta decisiva para todo ello es el *tráfico por nodo*, definido anteriormente.

10.3.1.1 Estado menos significativo

Para defnir "el estado menos significativo" q_x se han probado tres posibilidades (q_p y q_f son el estado final e inicial respectivamente):

- El estado de menos tráfico $T(q_p,q_x,q_f)$.
- El estado menos probable, es decir, el menos frecuentemente utilizado por R^+ (menor $f(q_x)$).

- La combinación de ambos criterios anteriores: el estado con menor producto $T(q_p,q_x,q_f)\cdot f(q_x)$.

Como se ha mostrado anteriormente, $T(q_p,q_x,q_f)$ está relacionado con la importacia estructural de q_x, es decir, la importancia de q_x para la variedad y tamaño del lenguaje *inferido*. Por su parte, $f(q_x)$ está directamente relacionado con la importancia de q_x para R_+, y por lo tanto para las cadenas de lenguaje *real*. Ambos criterios son complementarios y es difícil que se pueda decidir cuál es el mejor, lo que justifica la tercera definición.

10.3.1.2 Criterios de detención

Para detener la simplificación se han adoptado dos puntos de vista distintos, el algoritmo se detendrá cuando:

- Se haya suprimido un determinado porcentaje (de cadenas distintas) del lenguaje original.
- Se hayan borrado un determinado porcentaje de los estados del autómata original.

El primer punto es un límite abstracto, directamente relacionado con la generalización que se le permite al modelo. El segundo es un límite concreto, que limita la complejidad espacial (temporal) de la representación. Ambos están muy relacionados, aunque la simplificación por estados es más fácil de controlar (téngase en cuenta que suprimir el 99% del lenguaje aún nos deja con aproximaamente el 50% de los estados, dada la relación exponencial entre tamaño de lenguaje y número de estados).

En la práctica, los porcentajes de estados o cadenas del lenguaje a mantener se deberán estimar empíricamente y dependerán muy fuertemente de la tarea concreta de reconocimiento a la que estén abocados los modelos finales.

10.3.1.3 El algoritmo

El algoritmo concreto de simplificación es el siguiente (transcripción directa del implementado):

```
Algoritmo Simplifica
Datos     A=(V,Q,δ,qp,qf)              /* Autómata a simplificar */
          umbral:Z
Resultado As=(V,Qs,δs,qp,qf)           /* el mismo, simplificado */
Auxiliar                               /* τ es el tipo "estado" */
   criterio:τ→R { T(q)
                { f(q)                  /* según simplificación */
                { T(q)·f(q)

Variables qx :τ
Método
   /* borrado de estados sobrantes */
   repetir
      ObtenerTráfico(A);  Nc(qp,qf):=T(qp)
      si Nc(qp,qf) > umbral entonces
         qx :=        argmin        (criterio(q))
              ∀q∈Q,T(qx)<Nc(qp,qf)
      borrar(qx)
   hasta Nc(qp,qf)<=umbral;

   /* borrado de desconectados */
   ObtenerTráfico(A)
   ∀ q∈Q si T(q)=0 entonces borrar(q)
fin simplifica
```

El algoritmo mostrado utiliza como criterio de detención el tamaño del lenguaje. Es obvia la modificación que sería necesaria para detenerlo por número de estados. En cada caso habrá que calcular el umbral correspondiente a partir de $|Q|$ o $N_c(q_p,q_f)$ iniciales.

Se ha utilizado la particularidad evidente de que todos los caminos que van de q_p a q_f pasan por q_p, es decir, se ha obtenido $N_c(q_p,q_f)$ como $T(q_p)$. La reobtención del tráfico a cada iteración es imprescindible, incluso aunque no se le utilize en ningún criterio, para poder asegurarse de no borrar un estado por el que pasan todos los caminos (lo que puede ocurrir si se lleva la simplificación demasiado lejos). Ello se comprueba en la búsqueda del estado que cumple el criterio de simplificación, asegurándose de no escoger un estado que no se puede borrar.

Nótese que la segunda etapa del algoritmo es la que borra todos los estados que han quedado desconectados, es decir cuyo tráfico es nulo.

Obviamente, borrar(q) también borra las transiciones que llegan (van) a q. Ello altera la cuenta de los caminos que salen de q, por lo que la normalización efectuada en la extensión estocástica deja de ser consistente. Para recuperar la consistencia sería necesario un barrido del autómata final, en el que se igualan de nuevo las frecuencias de los nodos a la suma de las transiciones que les llegan y se vuelve a normalizar.

El algoritmo recurre repetidamente a la evaluación del tráfico. Es por lo tanto imprescindible disponer en todo momento del autómata y de su inverso (con las transiciones invertidas) con el fin de poder efectuar los barridos hacia delante y hacia detrás.

10.3.2 Aplicación práctica y resultados

Aunque se ha implementado ambas técnicas de detener la simplificación, todos los experimentos se han llevado a cabo únicamente simplificando por tamaño de lenguaje, ya que esta es una medida más relacionada con la semántica del modelo a simplificar. En todos los casos, las tasas de reducción por estados que se proporcionan se han medido *a posteriori*.

En 3 de los experimentos realizados con los dígitos hablados H1, H5 y H6 (ver detalles en capítulo 8) se han simplificado los 10 autómatas representativos de cada dígito y se ha repetido el reconocimiento. Recuérdese que todos los experimentos son multilocutor y la única diferencia entre H5 y H6 es un locutor en aprendizaje, notablemente diferente a los demás, en H5. H1 utiliza 5 locutores (10 muestras de cada locutor por dígito) en aprendizaje y 5 en reconocimiento. H5 y H6 utilizan 6 y 4 respectivamente. Para H6 son 6 locutores. En todos los casos se utilizó el modelo de error completo en reconocimiento. Como criterio de estado menos significavo se ha utilizado la combinación $T(q_p,q_x,q_f)\cdot f(q_x)$.

La simplificación se ha efectuado en tres etapas, cada una de las cuales significaba cada vez reducir de un 99% el lenguaje del autómata de la anterior.

Tabla 10.1 Resultados de reconocimiento tras simplificación de autómatas en tres experimentos de dígitos hablados. El porcentaje es respecto al número de estados del autómata original.

Experimento	|Q|	Sin Simplificar	40%	30%	20%
H1	1533	99,2	98	96	94,4
H5	1712	100	99,75	99,75	94,5
H6	1587	100	100,0	99,3	97,5

En la tabla es posible comprobar cómo en la mayoría de los casos los resultados empeoran sólo ligeramente o nada en absoluto cuando se suprime el 99% del lenguaje (el autómata queda con un 40% de estados), lo cual implica que realmente existe mucha información redundante en los autómatas, sobre todo cuando el aprendizaje ha sido suficiente como en el caso de H6.

Para poder estimar cómo empeora la tasa de reconocimiento en función de la simplificación aplicada, se completó el experimento H6 reduciendo

progresivamente el tamaño del lenguaje, midiendo cada vez la tasa de reconocimiento y el porcentaje de estados restantes respecto al autómata original. Representando los valores obtenidos se consiguió la curva de la figura 10.4.

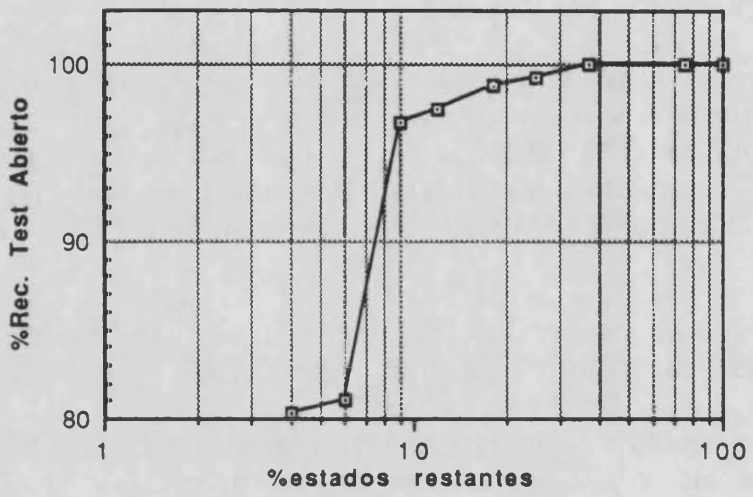


Figura 10.4 Tasa de reconocimiento en función del porcentaje estados restantes en un experimento de reconocimiento de dígitos hablados (H6). Las abcisas de los punto representados son 72%, 46%, 42%, 31%, 29%, 22%, 21%, 16% y 15%.

Entre cada dos puntos se ha producido una reducción de un 99% del lenguaje, cada punto intermedio implica una reducción del 50% del anterior.

Los errores en este caso empiezan entre un 37% y 25% de estados restantes (67 a 49 estados de media), y la tasa de reconocimiento nunca desciende por debajo del 80%, incluso cuando sólo quedan un 15% de los estados de los autómatas (24 estados de media). El emporamiento de la tasa de reconocimiento es muy lento, hasta un cierto punto (alrededor de un 20% de estados) en el cae bruscamente, probablemente debido a que en ese punto se suprimen partes de los lenguajes que diferencian las clases, y no sólo pequeñas variaciones de las que puede dar cuenta el modelo de error empleado en reconocimiento.

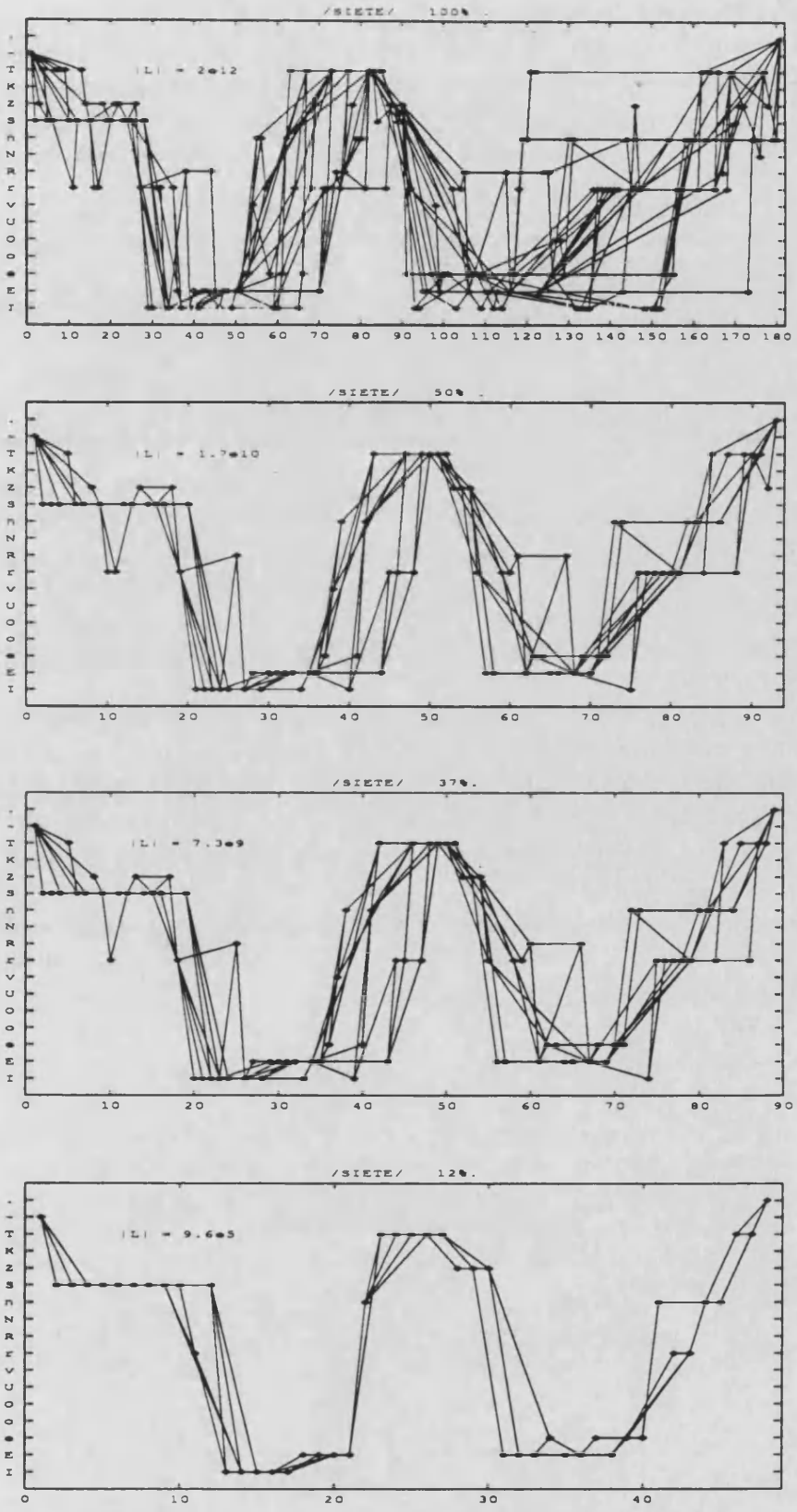


Figura 10.5 Diversas etapas de la simplificación de un autómata inferido por ECGI. El autómata con 37% de estados aún proporciona un 100% de aciertos en reconocimiento. El de 12% aún un 80%.

En la tabla 10.2 se presenta una comparación de las diferentes tasas de reconocimiento que se obtienen variando el criterio de estado menos significativo, escogiendo dicho criterio entre los tres descritos. En la tabla se encuentran los resultados obtenidos repitiendo E6 para cada uno de los criterios y con los niveles de simplificación 40%, 30% y 20% de los estados:

Tabla 10.2 Comparación de los tres criterios de simplificación: f(q)=frecuencia del estado en R_+. T(q)=tráfico del estado. Tasa de reconocimiento en función del número de estados restante.

Criterio	40%	30%	20%
T(q)	97,5	97	93,75
f(q)	99,5	99,5	94,25
T(q)·f(q)	100,0	99,3	97,5

Los resultados presentados demuestran que es muy factible simplificar los autómatas inferidos por ECGI sin afectar significativamente la tasa de reconocimiento. El porcentaje de simplificación admisible depende en gran manera de la tarea concreta a la que se aplicarán los autómatas resultantes, pero se ha comprobado que si éstos están suficientemente bien aprendidos se puede llegar a suprimir más de un 50% de estados, pudiéndose incluso llegar a un 75%.

Nótese que una posible utilización, muy sugestiva, de la simplificación de autómatas, podría consistir en proporcionar la capacidad de "olvido" a un sistema que integrara la inferencia y reconocimiento: un aprendizaje indefinido lleva a la modelización de muchísimos errores por las mismas gramáticas, errores que se producen con muy poca frecuencia y que es superfluo tener "anotados". Una ligera simplificación, realizada cada cierto número de muestras durante el aprendizaje, permitiría suprimir de las gramáticas estos espúreos.

11

ECGI Determinista y Cadena Mediana

11.1 Introducción

Este capítulo se divide en dos partes bien diferenciadas. En la primera de ellas se realiza una modificación (subóptima y heurística) al método de construcción de ECGI con el fin de conseguir que éste genere *gramáticas deterministas* (y por lo tanto no ambiguas). En la segunda parte se propone un nuevo método, basado en autómatas inferidos por ECGI, para obtener la *cadena mediana* de un conjunto de cadenas.

11.2 Gramáticas Deterministas

La posible ambigüedad de las gramáticas generadas por ECGI imposibilita, no sólo la estimación correcta (mediante métodos de reducida complejidad) de las probabilidades de las reglas utilizadas en su extensión estocástica, sino también la estimación exacta del número de cadenas distintas del lenguaje de la gramática. Esta mayor complejidad de tratamiento, debida a la ambigüedad, se extenderá a muchos otros posibles tratamientos y análisis a los que se desee someter las gramáticas generadas por ECGI.

Una manera sencilla e inmediata de hacer desaparecer la ambigüedad consiste en forzar a ECGI a generar gramáticas *deterministas*. El determinismo de las gramáticas es además una propiedad interesante por sí misma (p.e.: si no se utiliza corrección de errores permite realizar el análisis sintáctico sin tener que recurrir a Viterbi).

En los siguientes apartados se introduce una modificación de la etapa de comparación de ECGI, que permite obligar a éste a generar gramáticas deterministas.

11.2.1 Método

No es conveniente intentar introducir el determinismo en la etapa de construcción de ECGI. Ello supondría alterar la asignación de reglas efectuada por la optimización, sin ninguna garantía de mantener una mínima optimalidad. En efecto, si una nueva regla fuera imposible por ser no determinista, sería necesario volver a buscar por toda la gramática dónde y cómo intercalar óptimamente la subcadena que ha quedado desconectada, y ello desde luego, manteniendo la coherencia con los heurísticos de ECGI (ausencia de circuitos, de transiciones nil, etc...).

Es pues en la búsqueda de la derivación óptima donde se introduce el conocimiento de cómo opera la etapa de construcción, para así prevenir la generación de derivaciones que conduzcan a construcciones no deterministas.

Examinando la etapa de construcción de ECGI, se comprueba que toda nueva transición que es añadida a un estado existente, se corresponde con un punto de la derivación óptima donde, a continuación de una regla de no error, aparece una regla de error (es un punto donde la cadena se separa de la cadena más próxima de la gramática). Para asegurar el determinismo basta entonces rechazar, durante la búsqueda de la derivación óptima, toda derivación que demuestre tener una regla de no error seguida de una de error $(A\rightarrow bB)(B\rightarrow cX)$; y ello, siempre que el símbolo asociado a la regla de error aparezca en una regla ya existente con el mismo no terminal a la izquierda, p.e.: $(B\rightarrow cC)$.

Formalmente, ello equivale a decir: si $G_i=(N_i,V,P_i,S)$ es la gramática actual del paso i-ésimo de una inferencia por ECGI, y $D(x_i,G_i)=r_1,r_2,..,r_s,r_{s+1},..,r_k$; $r_j\in P^e_i$; $j=1..k$; $r_s=(A\rightarrow bB)\in P_i$; $r_{s+1}=(B\rightarrow cX)\in P^e_i$; es una derivación de $x_i\in V^*$ en G_i extendida a errores $G^e_i=(N_i,V,P^e_i,S)$, y $\exists(B\rightarrow cC)\in P_i$; entonces $D(x_i,G_i)$ **no es** una derivación admisible para el método ECGI determinista.

Esta condición es imposible de imponer conservando la complejidad del algoritmo `ViterbiCorrector`, pues incumple el principio de optimalidad de Bellman: no se puede determinar si desde un principio se está escogiendo una derivación que en su mitad va a ser prohibida, por lo que no se puede garantizar que una decisión local sea óptima. El método propuesto asume esta no optimalidad para poder generar gramáticas deterministas, y por lo tanto, **en el método ECGI determinista, el proceso de búsqueda de la**

derivación que minimiza la disimilitud es subóptimo (o el que maximiza la similitud).

En la práctica, pues, se mantiene el algoritmo `ViterbiCorrector`, sólo que en el proceso de optimización se rechaza, en un punto dado del trellis, aquella decisión que, en la posterior construcción, llevaría al no determinismo. Con esta condición, el cálculo de ViterbiCorrector deberá obligatoriamente progresar *de atrás hacia delante*, puesto que es necesario, para decidir si se acepta una regla como de no error, tener la regla siguiente (anterior en el cálculo) para comprobar si cumple la condición.

Aparece entonces otro problema, también debido a la no optimalidad del procedimiento empleado, y que reside en el principio de la cadena: cuando llega el momento de la última decisión, ésta puede ser imposible. En efecto, si la segunda regla es de error, la primera deberá cumplir la condición de determinismo. Esto puede forzar a descartar todas las derivaciones que partan sin error del axioma, si con todas ellas ocurre lo mismo (téngase en cuenta que la decisión es local, no se puede volver atrás). El siguiente es un ejemplo de esta situación (en negrita las reglas de error, ~ representa la axioma, e,d∈ V; E,D∈ N;):

(~→eE)(E→**e**E)(E→dD)... /* Derivación no determinista */

con lo que se puede terminar escogiendo una derivación en la que la primera regla es de error:

(~→e~)(~→eE)(E→dD)

Sin embargo, sea o no la primera regla de error, ECGI está forzado a conectar, ocurra lo que ocurra, la (sub)cadena al primer estado (~) y ello, aunque ya exista una regla que empieze con el mismo símbolo y que provocará la aparición del no determinismo.

La solución adoptada no puede ser más pragmática: toda cadena cuya derivación "óptima" determinista empieze con una regla de error es *descartada*. Es decir, **en la inferencia de gramáticas deterministas, ECGI descarta ciertas cadenas patrón por ser incompatibles con la gramática determinista que construye.**

Se supone que estas cadenas son resultado de algún error poco usual, y que la información útil que transportan será aportada por otra cadena patrón más adelante. Ello en el caso general será cierto si el número de muestras de aprendizaje es suficiente. En el peor de los casos habrá que aumentar ligeramente este número pues, tal como se comprueba de forma empírica, suelen descartarse menos de un 20% de cadenas. Alternativamente sería posible, una vez consideradas todas las muestras, reconsiderar aquellas que han sido descartadas, en la esperanza (bastante plausible, al haber crecido notablemente la gramática) de que esta vez serán aceptadas.

Obsérvese que, aunque la gramática así obtenida es determinista, ello **no implica** que su inversa lo sea. Al invertir las transiciones puede perfectamente ocurrir que un estado sea origen de más de una transición con el mismo simbolo. Lo que sí es invariable a una inversión es la ambigüedad: tanto la gramática como su inversa serán no ambiguas; si no hay ningún camino igual a otro en uno de los sentidos, tampoco lo habrá en el contrario.

Por otra parte, la aplicación de un algoritmo determinista imposibilita prácticamente, si se opta por un proceso de aprendizaje continuo, la integración inferencia-reconocimiento del algoritmo de optimización, a menos se se tolere la intrínseca suboptimalidad de este último durante el reconocimiento.

11.2.2 Implementación

Como se mencionó cuando se explicó el algoritmo de construcción para un LAS (autómata de estados etiquetados: ver capítulos 2 y 6), el método ECGI determinista emplea un método de construcción ligeramente distinto a método estándar. Ello es debido a la necesidad que tiene de conocer puntualmente, en un punto dado del trellis, si la regla es o no de error.

En la implementación aquí realizada, y como ya se mencionó en el capítulo 6, ECGI representa los autómatas a la inversa, y que hace el barrido del trellis de adelante hacia atrás. Ello no modifica en absoluto las ideas expuestas en el apartado anterior: únicamente implica invertir las definiciones y explicaciones. Es decir, y resumidamente: en vez de buscar dónde se separan la muestra y la cadena más próxima se buscará donde se *juntan*. La condición de determinismo implica un par $(X \rightarrow aA)(A \rightarrow bB)$ de regla de error seguida de no error (al contrario de lo antes definido), y la regla de error no debe tener el mismo símbolo que cualquier posible regla ya existente $(C \rightarrow aA)$. El problema de bordes aparecerá al *final* de la cadena.

Nótese que la inversión es debida al sentido del barrido, no de la representación. Aunque se calcule con las transiciones hacia delante, hay que usar esta última definición. Sin embargo, la inversión implica que el autómata obtenido es determinista de*atrás hacia delante*, es decir, el autómata que es determinista es el autómata *inverso* al obtenido.

Se describe a continuación el algoritmo que permite asegurarse, en el caso de un LAS, si la derivación es o no determinista en cada punto del trellis. El algoritmo se aplica en el momento de analizar una arista de substitución (pa,p):

```
Algoritmo Substitución Determinista
Auxiliar   /* Las 2 primeras son funciones sobre un vértice del trellis */
      q:Q×V→Q          /* estado asociado */
      v:Q×V→V          /* terminal asociado */
      eti:Q→V          /* etiqueta de un estado (LAS) */
Datos
      p∈Q×V      /* vértice examinado en el trellis */
      pa∈Q×V     /* vértice del trellis anterior según */
                 /* la regla de susbtitución examinada */
Método
  si v(p)=eti(q(p))      /* puede ser regla de no error */
  y ∃ qx:eti(q(pa))=eti(qx) y δ-1(qx,eti(q((p)),q(p))
                         /* existe ya una transición con */
                         /* igual terminal */
  entonces
     si q(pa)=qx         /* es la que se esta examinando */
        y v(pa)=eti(q((pa)) y
          TipoRegla(pt,pa)=Substitución
                         /* la regla anterior no es de error*/
     entonces devolver coste (pa,p)
                         /* esta regla es de no error */
     sino prohibir (pa,p)
                         /* es regla no determinista */
  sino devolver coste (pa,p)
                         /* es regla de error o regla de no */
                         /* error aún no existente */
fin Substitución Determinista
```

Obsérvese que las condiciones de regla de no error son idénticas a las ya explicadas en el algoritmo ECGI-LAS (costrucción), para el caso no determinista, y que una de estas condiciones se aplica a una arista que parece indefinida (pt,pa). En la práctica esta arista es la que obliga el sentido de recorrido y está apuntada en la *matriz de producciones* (matriz que cada punto del trellis indica la arista de donde viene el camino óptimo hasta ese punto).

La que sí queda explícita es la condición de que, si la regla que se está examinando existe ya (es de no error), y la regla anterior es de error, se prohíbe la que se está examinando.

La aplicación de este algoritmo implica el coste añadido, sólo en el caso de aristas de sustitución, de 1 a 4 comparaciones, más un barrido de los predecesores del estado asociado al punto del trellis examinado, para comprobar si tienen el símbolo en cuestión.

La comprobación del efecto de borde se hace justo en el momento de iniciar la etapa de construcción, descartando la cadena si la última regla de la derivación es una inserción o un borrado (debido a la definición de LAS, si es una substitución, no es error; no se puede substituir el símbolo final).

11.2.3 Resultados Experimentales

Para comprobar la efectividad del método de obtención de gramáticas deterministas, y para asegurarse de que el determinismo no implica pérdida de capacidad de reconocimiento (máxime teniendo en cuenta que se descartan cadenas de aprendizaje), se repitieron con la versión determinista de ECGI casi todos los experimentos de reconocimiento realizados con dígitos hablados (ver detalles en capítulo 8) (tabla 11.1). Al final de la inferencia de cada autómata, el método ECGI determinista comprueba (mediante el algoritmo trivial) que efectivamente el (inverso del) autómata generado es determinista.

El método ECGI determinista utiliza la maximización de aciertos como criterio de disimilitud durante la inferencia. Todos los experimentos mostrados se han realizado con el modelo de error completo en reconocimiento. Piloto NE se refiere a un experimento con el corpus piloto en el que no se utiliza la extensión estocástica.

Tabla 11.1 Comparación de la tasa de reconocimiento en distintos experimentos según los autómatas sean deterministas o no. Se muestra también en cada caso la talla media de los autómatas y el porcentaje promedio de patrones descartados en la inferencia determinista

Experimento	No Deter.	|Q|	Deter.	|Q|	Descartadas
Piloto	99,5	1063	98	1059	9%
Piloto NE	97,5	1063	99	1059	9%
H1	99,2	1533	99,2	1379	12,4%
HLKO11	98,2	1965	98,2	1699	17,6%

Ninguno de los autómatas inferidos durante los experimentos cuyos resultados se muetran en la tabla 11.1 ha resultado ser no determinista, lo que confirma el buen comportamiento del algoritmo propuesto. Lo más notable que se aprecia en la tabla es la nula pérdida (o ganancia) de capacidad de reconocimiento en (casi) todos los experimentos, a pesar de que (como se comprobó examinando uno por uno los listados de inferencia de los 50 autómatas de HLKO11) se descartaron en algunos casos hasta el 56% de las muestras de aprendizaje (este porcentaje resultó ser muy variable: del 1.3% a 56%). Los peores resultados de la versión determinista en el experimento básico en el el caso estocástico, se compensan con los mejores resultados para el caso no estocástico.

En cuanto al tamaño de los autómatas, se reduce de aproximadamente un 11% (en realidad un 8%, si se considera que el factor de ramificación aumenta ligeramente: 1.97 a 2.06 en HLKO11), pero es fácil comprobar que en gran parte es debido al menor número de muestras que se han utilizado en su inferencia (ver figura 11.1). Posiblemente las muestras descartadas son muestras con muchas pequeñas diferencias, compensadas en reconocimiento por el modelo de error. La estructura de los autómatas

inferidos por la versión determinista es bastante similar a los inferidos por la versión no determinista (figura 11.2).

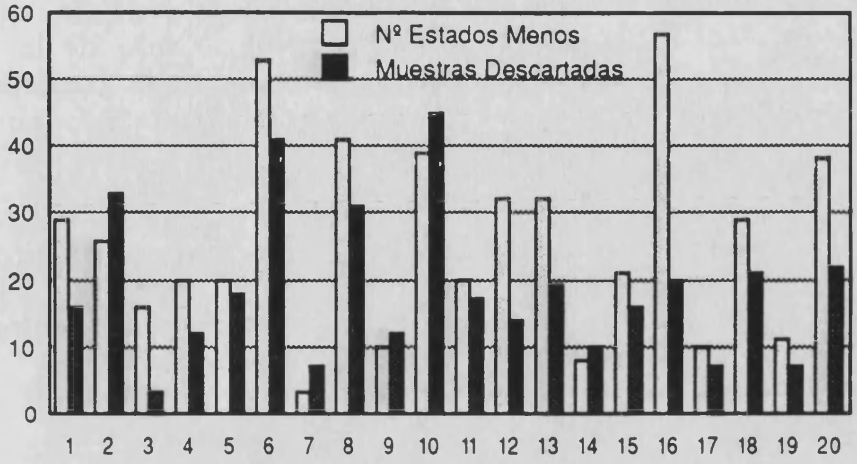


Figura 11.1 Diferencia entre el número de estados de la versión no determinista y la determinista (Número de estados menos) de 20 autómatas inferidos mediante ECGI. Comparación con el número de muestras descartadas en la inferencia de los 20 autómatas deterministas.

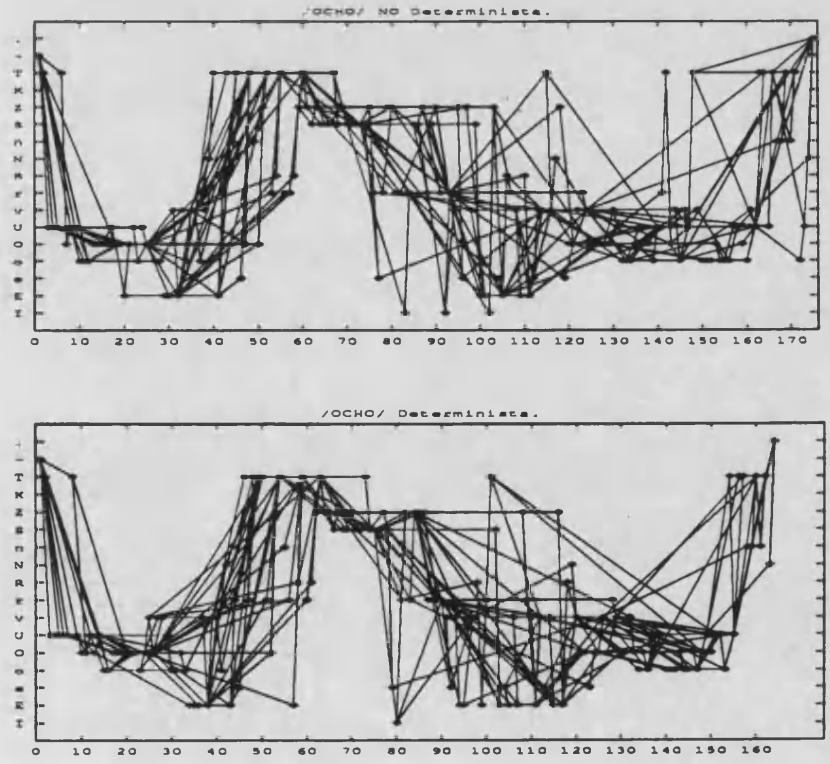


Figura 11.2 Autómata del dígito hablado /ocho/ inferido por la versión no determinista de ECGI y el inferido a partir del mismo conjunto de muestras con la versión determinista.

11.3 Cadena mediana

Dada una gramática (autómata) inferida por ECGI, la intuición sugiere que debe de ser posible encontrar una *derivación o camino medio* estructural que de alguna manera refleje la secuencia de reglas (estados o transiciones) o cadena más próxima (en el sentido de la corrección de errores) a todas las demás cadenas del lenguaje de la gramática (autómata). Esta es la que llamaremos *cadena mediana* de la gramática (autómata).

11.3.1 Mediana generalizada y mediana de un conjunto

Si $C=\{a_1,a_2,\ldots,a_n\}$ es un conjunto de n números reales, entonces, la media aritmética de C, que se escribe m, satisface la relación:

$$m= \underset{\forall x}{\operatorname{argmin}} \left| \sum_{i=1}^{n} (a_i - x) \right|$$

Por otra parte, la mediana M se define como el elemento de C tal que, si los a_i estuvieran ordenados, M tendría el mismo número de elementos anteriores que posteriores, es decir:

$$M= \underset{\forall x\in C}{\operatorname{argmin}} \sum_{i=1}^{n} |a_i - x|$$

Se puede generalizar esta última definición a un subconjunto S de un dominio arbitrario $\mathcal{D}$, dotado de una medida de disimilitud, también arbitraria. Si $d(x_i,x_j)$ es la disimilitud entre $x_i,x_j\in S\subseteq\mathcal{D}$, la *mediana generalizada* M_g de S satisfará la relación [Kohonen,85]:

$$M_g= \underset{\forall x\in \mathcal{D}}{\operatorname{argmin}} \sum_{i=1}^{|S|} d(a_i,x)$$

Si además se exige que M_g pertenezca a S, se la denominará *mediana del conjunto.*

Alternativamente, si en vez de una medida de disimilitud, existe una función $p:S\rightarrow[0,1]$, $\sum_{i=1}^{|S|} p(x_i)=1$, que proporciona la probabilidad de que $x_i\in S$, el *elemento más probable* M_p:

$$M_p= \underset{\forall x}{\operatorname{argmax}}\ p(x)$$

representa la moda de S.

Si la distribución en S es normal, M_p será también la media y la mediana de S, y M_g será a la vez la moda y la media.

11.3.2 La cadena más probable

Utilizando las definiciones del apartado anterior, [Kohonen,85] propone un método para obtener la cadena mediana de un conjunto de cadenas. El método de Kohonen es relativamente costoso, pues supone el buscar primero la mediana *del conjunto* (por el procedimiento más directo: calculando la matriz de distancias entre todas las cadenas del conjunto) y luego realizar variaciones sistemáticas (errores) sobre ésta hasta dar con la mediana.

Por otro lado, si la inferencia realizada por ECGI es adecuada, se espera que la cadena mediana del conjunto R^+ esté muy próxima a la cadena más probable de la gramática inferida por ECGI (se está suponiendo que R_+ tiene una distribución aproximadamente "normal" en el espacio V^*).

La posibilidad de obtener la cadena mediana de R_+ a partir del autómata generado por ECGI proporciona no sólo un nuevo método para obtener la cadena mediana (útil para extraer la cadena real a partir de un conjunto de muestras extremadamente ruidosas, ver ejemplos más adelante), sino que confirma que el autómata generado por ECGI modeliza adecuadamente R_+.

11.3.3 Consideraciones prácticas

En el estudio que se presenta a continuación, llevado a cabo para verificar hasta qué punto la cadena más probable de la gramática (del lenguaje del autómata) está próxima a la cadena mediana, se tuvo que recurrir a un algoritmo en cierto modo similar al de Viterbi. La principal diferencia es que en este caso no se dispone de una cadena que determine qué transiciones escoger y cuándo terminar. No hay una etapa del trellis por cada símbolo de una cadena a reconocer, sino que se genera una nueva etapa cada vez que los caminos a través del autómata progresan de una transición a la siguiente. Se mantiene una lista de estados activos en la etapa anterior (un estado se puede activar más de una vez, a medida que van llegando a él caminos de distintas longitudes y total acumulado mayor al actual del estado). En cada iteración se escoge como camino máximo para un estado el que provenga de un estado activo y tenga el mayor total acumulado, si éste es mayor que el actual del estado. Se termina cuando el único estado activo sea el final. El máximo número de iteraciones viene dado por la longitud del camino más largo del grafo.

Como alternativa a este proceso, relativamente costoso (aunque menos que el propuesto por kohonen), se han estudiado dos definiciones: la *cadena más frecuente* y la *cadena localmente más frecuente*, que permiten obtener aproximaciones a la cadena más probable de una gramática, pero utilizando algoritmos de coste (en el segundo caso muy) inferior y proporcionando, en ocasiones, mejores resultados.

11.3.4 La cadena más frecuente

La secuencia de transiciones más frecuente es aquel camino en el autómata (cadena del lenguaje) que maximiza la suma de las frecuencias de utilización por R_+ de las transiciones (reglas) que utiliza.

Sean $f_c(r_i)=i=1..k$, las frecuencias de las k transiciones (reglas) que utiliza el camino $C \in \mathcal{C}(p,f)$, donde $\mathcal{C}(p,f)$ son todos los caminos que van de p (el axioma) a f (el estado final del autómata[1]). La secuencia de transiciones más frecuente viene dada por:

$$C_{ET}(p,f) = \underset{\forall C \in \mathcal{C}(p,f)}{\operatorname{argmax}} \left\{ \sum_{i=1}^{k(c)} f_c(r_i) \right\}$$

Esta definición, en contraste con la de la cadena más probable, utiliza directamente la información frecuencial, sin la normalización que implican las probabilidades.

La cadena *localmente* más frecuente es en realidad un procedimiento extremadamente *subóptimo* para obtener la cadena más frecuente: se empieza desde el estado inicial (o final si el autómata está representado de atrás hacia delante), y se escoge como estado siguiente aquel al que lleva la transición (regla) con más frecuencia en R_+ (mayor f(r)); se repite esto mismo hasta llegar al estado final. Este procedimiento no implica ninguna optimización global y es por lo tanto extremadamente rápido y sencillo de implementar, habiendo proporcionado resultados extremadamente buenos (véase apartado siguiente).

---

[1] Recuérdese que los autómatas generados por ECGI sólo tienen un estado final. La definición es inmediatamente generalizable a un caso en el que existieran múltiples estados finales.

11.3.5 Comprobación empírica

Se llevaron a cabo una serie de experimentos para estudiar el buen comportamiento de las definiciones anteriores. En todos los casos se partió de grafos (autómatas) generados por ECGI a partir de muestras sintéticas. Estas cadenas sintéticas (palabras escritas con letras mayúsculas) se generaron produciendo errores aleatorios sobre una cadena dada (50% de distorsión: tantos errores como la mitad del número de caracteres de la cadena). Los errores eran los clásicos de borrado, inserción y substitución. El conjunto de símbolos insertables o que sustituían a otro estaba formado por las 26 letras mayúsculas inglesas (ni CH,Ñ,LL,RR ni acentos). El procedimiento de distorsión era sencillo: dado el porcentaje de distorsión se obtiene el número de errores a provocar; se escoge entonces al azar la posición y el tipo de error para cada uno de ellos.

Se hizo en todos los casos la suposición de que la cadena media del conjunto de muestras era la cadena original. Esta suposición es bastante plausible, dado el método de generación de las cadenas, sobre todo cuando el número de muestras es grande. Las palabras alteradas se escogieron arbitrariamente, en principio con el único criterio de que fueran de longitud diferente entre sí (aunque tres de ellas son las mismas que utiliza [Kohonen,85]). Fueron, en orden creciente de longitud: "IAPR", "HECTOR", "HELSINKI" y "RECOGNITION". Se generaron 4 autómatas para cada palabra, a partir de conjuntos formados por respectivamente 10, 20, 50 y 100 muestras alteradas; esto permitió comprobar si las definiciones eran válidas independientemente del tamaño de los autómatas y si su comportamiento mejoraba al aumentar la cantidad de información frecuencial disponible. En la figura 11.3 se muestran los conjuntos de 10 cadenas utilizados para generar los 4 primeros autómatas.

CIANR	HETCR	CHEINNI	RGFKFGNITION
IAP	HEPTOR	HLELSIKI	RECOXSNIIMOI
IAPI	HECTOR	HESENKC	RIECOXGNIFON
LAPR	HEVOR	VELSKKI	JEOGNITIGQN
ILP	HETUOR	CEELTSINKMI	RESONIGIOR
RIAPR	HSCOR	ELNSGXNKI	REOINITIGGN
IALR	HTUCTOR	GBHEKLSINK	RCIORGNITVIHN
IAR	FJHECTO	HTOSINI	RECOGNIN
IAPD	GETOQR	HXLSIKY	ECOTNRITIIN
IUAR	HETOFR	HEKLUSNKK	GRECPOGINITKO

Figura 11.3 Ejemplo de muestras de las palabras "IAPR", "HECTOR", "HELSINKI" y "RECOGNITION" alteradas aleatoriamente y utilizadas para los experimentos de cadena mediana (para generar los autómatas correspondientes a 10 muestras).

Los resultados obtenidos en los experimentos son extremadamente esperanzadores. La **cadena más probable** proporciona la cadena original en el 56% de los casos estudiados (9 de 16), la **cadena más frecuente** la

proporciona en el 62% de los casos examinados (10 de 16), siendo notablemente la mejor definición la versión subóptima de esta última, la **cadena más frecuente por transiciones**, la que proporciona la cadena original en 87% de los casos (14 de 16). Con las tres definiciciones (y sobre todo con la última) los resultados mejoran al aumentar el número de muestras (y por lo tanto la información contenida en el autómata), proporcionando las tres la cadena original cuando se aprende el autómata con 100 muestras. Como referencia, en la tabla 11.2 se muestran todos los resultados correspondientes a la palabra "HELSINKI". En la tabla 11.3 se muestran las cadenas localmente más frecuente obtenidas en los experimentos. Obsérvese que sólo en dos casos se ha producido un error con respecto a las cadenas originales.

Tabla 11.2 Cadenas más frecuentes localmente, más frecuentes y más probables de los autómatas genenerados por ECGI a partir de conjuntos de 10,20,50 y 100 muestras "ruidosas" de las palabras "HELSINKI".

HELSINKI (10 cadenas ruidosas)	
Cadena más frecuente localmente:	HELSINKI
Cadena más frecuente:	HELSINKI
Cadena más probable:	HEINNI
HELSINKI (20 cadenas ruidosas)	
Cadena más frecuente localmente:	HELSINKI
Cadena más frecuente:	HEBELSINNKLMI
Cadena más probable:	HELJNK
HELSINKI (50 cadenas ruidosas)	
Cadena más frecuente localmente:	HELSINKI
Cadena más frecuente:	HELSSINKI
Cadena más probable:	HELSINKI
HELSINKI (100 cadenas ruidosas)	
Cadena más frecuente localmente:	HELSINKI
Cadena más frecuente:	HELSINKI
Cadena más probable:	HELSINKI

Tabla 11.3 Cadenas localmente más frecuentes (CLMF) de los autómatas genenerados por ECGI a partir de conjuntos de 10,20,50 y 100 muestras "ruidosas" de las palabras "IAPR", "HECTOR", "HELSINKI", "RECOGNITION".

Palabra	NºMuestras	CLMF
HELSINKI	10	HELSINKI
HELSINKI	20	HELSINKI
HELSINKI	50	HELSINKI
HELSINKI	100	HELSINKI
IAPR	10	IAR
IAPR	20	IAPR
IAPR	50	IAPR
IAPR	100	IAPR
RECOGNITION	10	RECOGNITION
RECOGNITION	20	RECOGNITION
RECOGNITION	50	RECOGNITITION
RECOGNITION	100	RECOGNITION
HECTOR	10	HECTOR
HECTOR	20	HECTOR
HECTOR	50	HECTOR
HECTOR	1000	HECTOR

En conclusión, la cadena localmente más frecuente, obtenida de manera simple, proporciona un procedimiento muy directo y de muy baja complejidad para obtener una buena aproximación a la cadena más probable (y por lo tanto, en muchos casos, a la mediana), no sólo del lenguaje de un autómata inferido por ECGI, sino también del conjunto de muestras R_+, utilizadas para su construcción. Nótese sin embargo que un estudio más completo (utilizando mucho mayor número de palabras, de muestras y variando la distorsión) sería conveniente para centrar mejor los márgenes de funcionamiento del método.

La proximidad de la cadena más probable a la cadena mediana en la gran mayoría de los casos estudiados, proporciona además una demostración empírica de que el ECGI generaliza de manera progresiva y homogénea alrededor del conjunto de muestras, manteniendo el "centro de gravedad" del lenguaje (según Levenshtein y las probabilidades) próximo al centro del conjunto de muestras.

Como muestra práctica de ello, se adjuntan (figuras 11.4 y 11.5) la representación gráfica de las cadenas localmente más frecuentes de los lenguajes de algunos (uno por clase) de los autómatas inferidos durante los experimentos con dígitos manuscritos e impresos (rejilla 6).

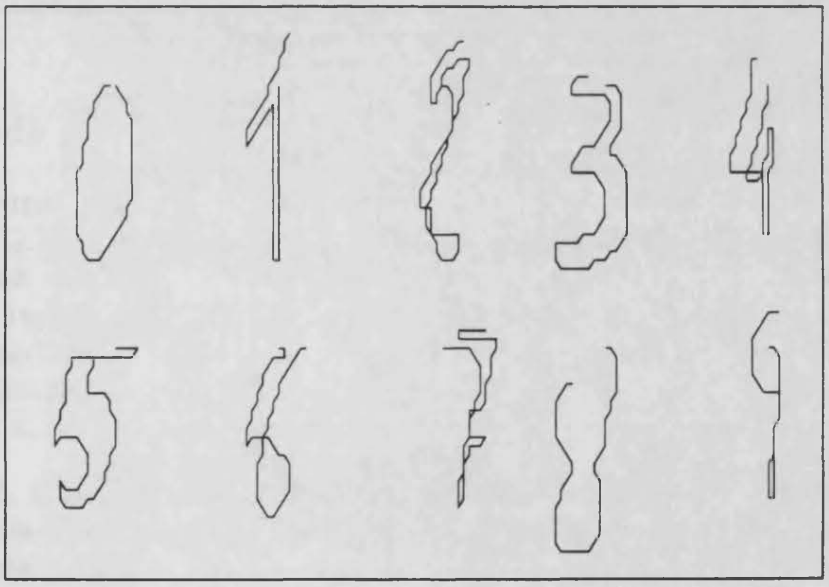


Figura 11.4 Cadenas localmente más frecuentes de los lenguajes de 10 autómatas (uno por clase) inferidos por ECGI a partir de imágenes de dígitos manuscritos (rejilla 6)

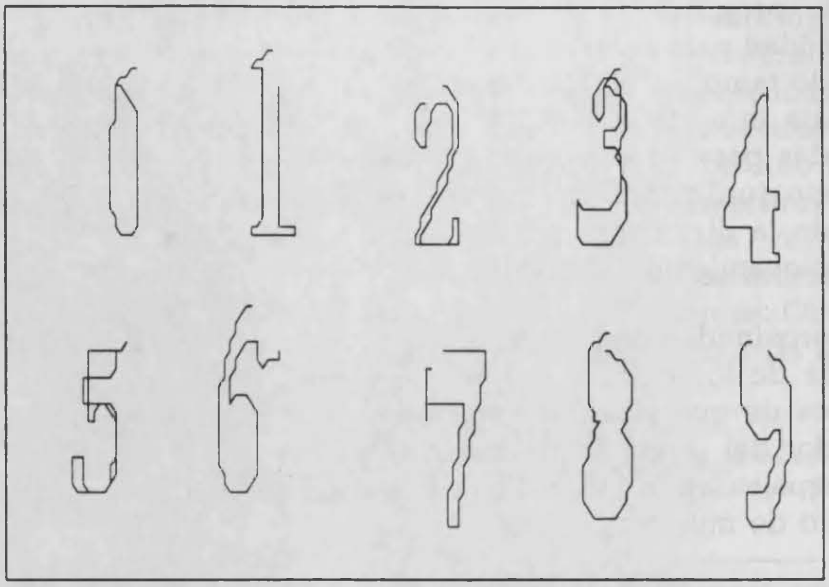


Figura 11.5 Cadenas localmente más frecuentes de los lenguajes de 10 autómatas (uno por clase) inferidos por ECGI a partir de imágenes de dígitos impresos (rejilla 6)

12

Comparaciones

12.1 Métodos similares de IG

La idea de utilizar un método de corrección de errores como punto de partida para la inferencia de modelos estructurales no es exclusiva de ECGI. Se resumen a continuación los trabajos de cuatro autores en este campo (únicos conocidos por el autor).

12.1.1 Método de Thomason

En el mismo congreso en que ECGI se presentó por primera vez [Rulot,87] M.G.Thomason expuso, también por primera vez, un algoritmo basado en una idea similar a ECGI, pero destinado específicamente a la inferencia de redes de Markov, método que fué publicado unos meses más tarde [Thomason,86] [Thomason,87]. Posteriormente [Gregor,88] adaptó el método añadiendo la posibilidad de imponer ciertas restricciones (puntos de identificación forzosos) al proceso de inferencia.

El modelo inferido es parecido al ECGI. No tiene circuitos, son los estados los que llevan los símbolos y sólo existe un estado final. Las redes son igualmente no deterministas y ambiguas [Gregor,88]. Por otro lado, al ser un modelo Markoviano, es estocástico desde un principio y tiene asociadas probabilidades a las transiciones y los estados.

Sin embargo, a diferencia del ECGI, se autorizan estados etiquetados con el símbolo nil, y los modelos son mucho más lineales. Ello es debido a que en construcción sólo se aplican criterios locales: se añaden directamente las reglas de error (ver figura 12.1).

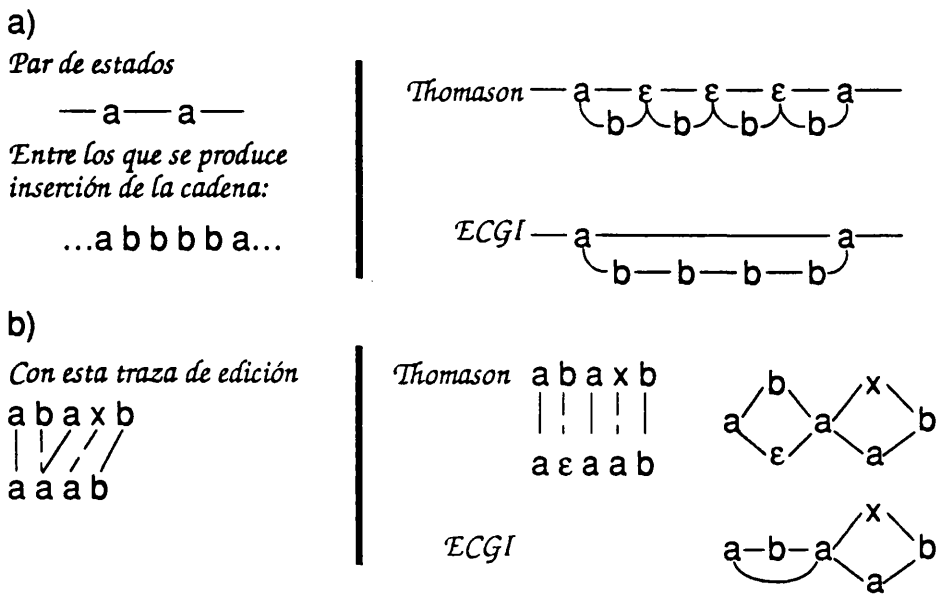


Figura 12.1 Inferencia mediante el algoritmo de Thomason y mediante ECGI a) en el caso de una serie de inserciones, y b) en un caso simple de sustitución inserción (ε es el símbolo nil).

En la documentación no se han encontrado claras referencias a otros heurísticos de construcción, aunque en [Gregor,88] se afirma que, en caso de duda en la selección del mejor camino, se examinan hasta 26 condiciones en orden predeterminado.

En la comparación se utiliza una distancia probabilística basada en las probabilidades de las transiciones y de los nodos (frecuencia relativa de uso de los mismos hasta ese momento por R_+). Las probabilidades de las transiciones de error se definen con coste fijo (se aplica el mismo modelo de deformación que en ECGI) (ver figura 12.2).

El modelo de error es el mismo en reconocimiento, no pareciendo preocuparse en ningún momento los autores de la estocasticidad de la gramática expandida. Sí que se preocupan de renormalizar en cada paso las probabilidades, manteniendo en todo momento la estocasticidad de la gramática que se infiere y que aprovechan durante la inferencia. Ello, desde luego implica un coste superior al ECGI, pues obliga a una normalización y cálculo de logaritmo por cada transición de la gramática expandida; esto en reconocimiento es obviable, pues la normalización puede hacerse de una vez por todas.

En cuanto a los resultados obtenidos, el método se ha utilizado en la inferencia de muestras sintéticas [Thomason,87], cromosomas [Gregor,88] y palabra hablada [Thomason,86].

En el modelo de Thomason, si f_{ij} es la frecuencia de utilización (hasta ese momento, por R^+) del arco desde el nodo s_i al s_j, entonces la probabilidad de una transición es:

$$p_{ij}=\frac{f_{ij}}{f_i} \; ; \quad f_i=\sum_j f_{ij}$$

La relación de recurrencia de programación dinámica se escribe (para el estado i y el símbolo j-ésimo de la cadena k-ésima v_{kj}):

$$d(i,j)=\max \{$$

/* Borrado */ $\quad d(i,j-1)\cdot\frac{1}{f_i+1}$

/* Substitución */ $\quad d(i-1,j-1)\cdot\frac{f_{i-1,i}+1}{f_{i-1}+1}\cdot\begin{cases} 1 \text{ si } v_{kj}=s_i \\ \frac{1}{f_i+1} \text{ si } v_{kj}\neq s_i \end{cases}$

/* Inserción */ $\quad d(i-1,j)\cdot\frac{f_{i-1,i}+1}{f_{i-1}+1}\cdot\frac{1}{f_i+1}$

$\}$

Figura 12.2 El algoritmo de Thomason (fase de comparación).

En el caso de la palabra hablada (los 10 dígitos ingleses), único en el que se puede comparar con ECGI, el método de Thomason obtiene, en los experimentos presentados [Thomason,86], resultados notablemente peores que ECGI: 86.4% de reconocimiento con autómatas de unos 180 estados de media, inferidos cada uno a partir de 180 muestras. Las muestras tenían una longitud media de 7 caracteres, de un conjunto de 30 símbolos que representaban distintas características acústicas (sonora, energía alta en distintos rangos de frecuencia, etc...). Hay que notar sin embargo, que muestras de este tipo no parecen precisamente las más adecuadas para métodos como el ECGI y similares, que buscan posiciones relativas y longitudes: las cadenas son demasiado cortas con demasiados símbolos.

12.1.2 Método de Falaschi

Recientemente A.Falaschi ha presentado un nuevo método, en el que utiliza ideas de construcción y simplificación de autómatas muy relacionadas con las utilizadas por ECGI [Falaschi,90] [Falaschi,90a], al que referencia explícitamente como un método similar.

Al igual que para el método de Thomason, el objetivo del método de Falaschi es la inferencia de redes de Markov. El método también está basado

en la programación dinámica, la cual utiliza en tres de los 5 pasos de que consta el método:

1) Ordenar las cadenas de R^+. La ordenación se basa en la probabilidad de las cadenas (obtenida a partir de la frecuencia de los símbolos que las componen) y en su desviación respecto a la longitud media.

2) Alineamiento de todas las cadenas mediante programación dinámica a una longitud fija e igual a la de la cadena mejor según la ordenación. La alineación se optimiza respecto a todas las cadenas anteriores, que se ponen en paralelo en uno de los ejes del trellis. La distancia escoge en cada punto el símbolo más parecido (al de la nueva muestra) que se encuentra en ese sitio en cualquiera de ellas, con la restricción añadida de que intenta evitar borrados de símbolos muy frecuentes e inserción de símbolos cerca de los extremos de la cadena.

3) "Colapso" del autómata canónico así obtenido, uniendo todos los estados con el mismo símbolo en la misma posición y almacenando la frecuencia de uso de cada nuevo estado y transición.

4) Generación, a partir del autómata colapsado, de una secuencia de muestras "más probables", también mediante programación dinámica y aplicando una distancia basada en la frecuencia de uso de los estados y transiciones. Después de cada optimización, que proporciona una nueva cadena óptima para la red en ese paso, se suprime la *transición menos probable* antes de realizar el siguiente alineamiento.

5) Construcción, a partir de las muestras generadas del autómata, mediante el siguiente algoritmo [Falaschi,90]:

```
Algoritmo Falaschi-Construcción
Datos     Sk∈V*, k=1..Ns        /* muestras generadas */
Método
  Generar el grafo inicial a partir de So asignando a
          cada símbolo su correspondiente estado.
  para k=1..Ns hacer
     para h=1..k-1 hacer
        Alinear Sk con Sh
        si la distancia es mínima          /* ∀h */
        entonces Dopt:=Dkh   /* derivación de Sk con Sh */
     fin para
     Añadir los estados relacionados por Dopt que
               tengan distinto símbolo.
     Generar las transiciones indicadas por Dopt
              entre estados presentes y presentes y/o
              añadidos.
  fin para
fin Falaschi-Construcción
```

La distancia para el alineamiento se define en este último paso como:

$$D(n,m) = d(S_k(n),S_h(m)) + \min \{ \; D(n-1,m-1),$$

$$D(n-1,m)+1, \; /*\text{inserción}*/$$

$$D(n,m-1)+1 \; /*\text{borrado}*/ \}$$

con $d(S_k(n),S_h(m))=2+\varepsilon$, $\varepsilon>0$ si $S_k(n)\neq S_h(m)$ y 0 si no (distancia entre el carácter n de la cadena k y el m de la cadena h).

Obviamente este último paso es el que más relacionado está con ECGI aunque utiliza una distancia, en absoluto equivalente, en la que los distintos errores tienen distinto coste (p.e: da ventaja a inserción seguida de borrado frente a sustitución, según Falaschi para forzar la inserción seguida de borrado si esto permite alinear dos símbolos iguales).

Falaschi no da aquí por terminado el proceso de aprendizaje, y reestima los parámetros del modelo de Markov inferido, al que añade un modelo duracional original [Falaschi,90b], utilizando procedimientos clásicos (Viterbi, Forward-Backward). El mismo modelo duracional es el que se emplea en reconocimiento, y es por lo tanto totalmente distinto al usado en construcción y al empleado por ECGI.

Con todo lo dicho, se hace muy difícil una comparación de resultados, aunque el mismo Falaschi admite que los suyos son pobres: 70% (pero se trata de una tarea de reconocimiento de *fonemas*).

El proceso de generación de muestras "sintéticas" recurre a un método de simplificación progresiva de una red, muy parecido al que utiliza ECGI para simplificar autómatas (ver capítulo 10). El criterio para encontrar la transición que debe borrarse recurre a la búsqueda del segundo camino mínimo, en el que se aplica un heurístico poco fiable (que el segundo camino no empieza y termina en los mismos estados que el primero [Falaschi,90a]). Curiosamente Falaschi no intenta usar el autómata "colapsado" directamente en reconocimiento (aunque sea simplificándolo).

Las estructuras inferidas deben de ser muy parecidas a las construídas por ECGI (Falaschi no introduce el símbolo nil en sus modelos), y si el método de generar muestras sintéticas funciona, tendrían que ser menos complejas (a menos de utilizar ECGI con reducción de estados), debiéndose obtenerse resultados comparables. Nótese que el ECGI con reducción de estados se ha utilizado para la misma tarea de inferir redes de Markov [Casacuberta,90], que luego son reestimadas mediante Baum-Welch. La diferencia principal con el método de Falaschi reside en que éste reduce la información *antes* de la construcción.

12.1.3 Método de Chirathamjaree

Por su interés y relación bastante directa con las ideas de ECGI, es necesario mencionar el trabajo de Chirathamjaree y Ackroyd [Chirathamjaree,80], en el que se presenta un interesante método incremental para la inferencia de gramáticas independientes del contexto.

Las gramáticas que infiere este método quedan restringidas a reglas de la forma:

$A \rightarrow aB$; $A \rightarrow Ba$; $A \rightarrow a$;

(gramatica lineal a izquierda y derecha); también están libres de circuitos y en ellas todas las cadenas generables a partir de un no terminal dado son de la misma longitud (siendo el axioma una excepción).

El método, que se inicializa generando la gramática canónica trivial para la primera cadena, aprovecha las peculiaridades impuestas a la gramática y rellena, a cada nueva muestra y mediante un algoritmo de programación dinámica, una matriz m_{ijk} en la que cada elemento contiene *el coste mínimo de generar, a partir del no terminal k-ésimo de la gramática, la subcadena de longitud i, que empieza en el símbolo j-ésimo de la muestra.*

Basándose en esta matriz, se aplica todo un conjunto de reglas para añadir a la gramática el mínimo número de no terminales y reglas necesarios para generar la nueva muestra, induciéndose, como en el caso de ECGI, una generalización.

El método puede evidentemente utilizarse para generar gramáticas regulares. El coste de evaluación de la matriz cúbica no es mencionado (sólo es necesario calcular la mitad de ella) y, desgraciadamente, los autores no proporcionan ningún resultado de aplicación a un caso real. Todo ello hace imposible ninguna comparación sin la implementación efectiva del método.

12.1.4 Método de Marco

En [Marco,88] se presenta una generalización de la idea de ECGI, aplicándola a gramáticas independientes del contexto. Se aplica el mismo modelo de error que ECGI para extender la gramática, aunque para el análisis sintáctico se hace necesario emplear el algoritmo de Eearley [Fu,82]. Un procedimiento que involucra un "traductor incorporado" permite evitar de construir el árbol de derivación a la hora de obtener la derivación óptima. Una vez obtenida esta última, se añaden a la gramática *las reglas de error*, con el fin de que esta última pueda reconocer a la cadena muestra. Los autores no llegan a realizar la extensión estocástica y no presentan resultados en aplicaciones prácticas.

12.2 Métodos en Reconocimiento del Habla

En reconocimiento del habla se han utilizado muy a menudo, y con muy buenos resultados dos métodos muy diferentes, y con los que es conveniente comparar ECGI: los *modelos ocultos de Markov* y el *alineamiento temporal no lineal.*

12.2.1 Modelos Ocultos de Markov

Para el reconocimiento sintáctico de cadenas de símbolos, es muy corriente la utilización de los modelos ocultos de Markov (Hidden Markov Models: HMM), siendo gracias a éstos con los que se han obtenidos algunas de los mejores tasas de aciertos en reconocimiento de formas. Ello es debido a su gran flexibilidad de definición estructural, a la existencia de muy eficaces y bien conocidos métodos de entrenamiento, a la capacidad intrínseca que poseen de tratar con el tiempo (longitud) y su facilidad de combinación para formar redes más complejas.

Es normal en reconocimiento de la palabra utilizar HMM muy reducidos (5 a 20 estados por palabra, a veces 30), constreñidos a estructuras lineales de izquierda a derecha y al sencillo modelo de error (transiciones permitidas) utilizado en este trabajo. Con estas limitaciones, se obtienen fácilmente tasas superiores al 90% en reconocimiento de dígitos [Rabiner,83].

Dicho esto, es inmediato preguntarse si las estructuras inferidas por el ECGI no son simplemente una versión compleja de los HMM, dada la equivalencia formal que existe entre los LAS estocásticos y los HMM.

Esto ha sido estudiado, y en [Casacuberta,90] se presenta los resultados mostrados en la tabla 12.1, obtenidos con el experimento HLKO (ver capítulo 8):

Tabla 12.1 Comparación entre el ECGI y los modelos de Markov HMM8 (estándar 8 estados) y HMM8S (estándar 8 estados con transiciones de borrado: transiciones de un estado al siguiente del siguiente estado). Experimento HLKO de reconocimiento de dígitos hablados.El número de símbolos es 15, y los autómatas de ECGI tienen un factor de ramificación 1.97.

Método	%Reconocimiento	NºParámetros	Nº Estados
ECGI	99.8	642	196 (media)
HMM8	98.5	135	8
HMM8S	98.7	141	8

En la que se evidencia que para un mismo conjunto de datos ECGI obtiene una tasa de reconocimiento 1.9% mejor que un HMM estándar de 8 estados.

Para obtener con modelos de Markov una eficiencia similar a ECGI se postula que probablemete serían necesario disponer de más de 3 HMM por palabra, entrenados mediante técnicas adecuadas de clustering [Rabiner,89], o utilizar HMM mucho más largos, del orden de 30 estados. Esta última aproximación, llevada a cabo en [Casacuberta,91] efectivamente permite obtener los mismos resultados que ECGI (tabla 12.2, con el mismo experimento); e incluso a veces mejores con sólo 20 estados (tabla 12.3, experimento LLKO).

Tabla 12.2 Comparación entre el ECGI y los modelos de Markov HMM10 (estándar 10 estados) y HMM30S (estándar 30 estados con transiciones de borrado: transiciones de un estado al siguiente del siguiente estado). Experimento HLKO de reconocimiento de dígitos hablados.El número de símbolos es 15, y los autómatas de ECGI tienen un factor de ramificación 1.97.

Método	%Reconocimiento	NºParámetros	Nº Estados
ECGI	99.8	642	196 (media)
HMM10	99.7	168	10
HMM30S	99.8	537	30

Tabla 12.3 Comparación entre el ECGI y los modelos de Markov HMM10, HMM20 y HMM30 (estándar 10, 20 y 30 estados respectivamente) y HMM20S (estándar 20 estados con transiciones de borrado: transiciones de un estado al siguiente del siguiente estado). Experimento LLKO de reconocimiento del EE-set hablado. El número de símbolos es 32, y los autómatas de ECGI tienen un factor de ramificación 1.62.

Método	%Reconocimiento	NºParámetros	Nº Estados
ECGI	73.8	2360	785 (media)
HMM10	67.6	338	10
HMM20	75.0	678	20
HMM20S	62.7	697	20
HMM30	73.7	1018	30

Otras alternativas, que también aumentan el coste espacio-temporal de los HMM hasta aproximarlos al ECGI involucrarían un mejor modelizado de la duración (p.e.: Probabilidades de transición continuas [Falaschi,90b]).

La mucho mayor complejidad temporal del ECGI, debida al gran número de estados, puede reducirse drásticamente utilizando las técnicas presentadas en [Torró,89] y [Torró,90] (Sólo estados alcanzados, Búsqueda en Haz, Reconocimiento Anticipado) consiguiéndose, con el mismo conjunto de datos, las mismas tasas de reconocimiento, visitando tan sólo un 9% de los estados (17 estados en este caso). La complejidad espacial del ECGI no es tan grande como parece, en relación con los HMM: el *número de*

parámetros de los HMM de 30 estados es de 537, y para los autómatas de ECGI con 196 estados es 642. El incremento muy grande del número de parámetros que se observa para el ECGI, en el caso del experimento con EE-letras, es debido al mucho mayor número de símbolos, que no sólo aumenta mucho la variabilidad (y por lo tanto el número de estados de los modelos inferidos por ECGI), sino además multiplica el número de probabilidades de error (la talla de la tabla de sustitución crece con el cuadrado del número de símbolos). Téngase además en cuenta que, con las técnicas presentadas en el capítulo 10, es posible reducir drásticamente el número de estados de los modelos inferidos por ECGI. Una reducción que llegue ai un número de estados eqivalente en los HMM –30– todavía consigue un 99% de aciertos, y ello, con un número de parámetros mucho menor y peor estimados que los HMM (puesto que la estimación de parámetros se ha hecho con el modelo sin reducir).

12.2.2 Alineamiento Temporal No lineal, K-Vecinos

Una técnica aún más clásica que los HMM, y también relacionada de algún modo con el ECGI, consiste en utilizar el alineamiento temporal no lineal (DTW) [Casacuberta,87] de la cadena muestra con un conjunto de N patrones (utilizando la regla de decisión K-NN), generamente obtenidos mediante técnicas de clustering a partir de las muestras de R_+.

Mediante DTW se consiguen porcentajes de reconocimiento iguales o superiores a los obtenidos mediante HMM, estando generalmente la diferencia justificada por la cantidad de datos disponibles (la gran cantidad de parámetros a estimar probabilísticamente siempre ha sido el talón de aquiles de los modelos estocásticos) y por la utilización de símbolos discretos o vectores de parámetros reales. En [Rabiner,83] se realiza un estudio comparativo muy detallado y exahustivo, donde se muestra que DTW (12 patrones/clase) llega a obtener hasta un 3% de ventaja sobre los HMM (5 estados) si se les entrena con 100 a 200 patrones (la tarea es otra vez el reconocimiento de dígitos ingleses hablados, cuantificando con 64 símbolos). Utilizando vectores en vez de símbolos se mejora el DTW en otro 3%, una mejora que probablemente se extendería a los HMM en el mismo caso.

La diferencia entre los dos modelos, es en el fondo la diferencia entre un modelo paramétrico (estructura fija, gran número de parámetros a estimar estadísticamente) y otro no paramétrico (estructura desconocida, comparación mediante distancia con patrones almacenados), y los resultados de [Rabiner,83] indican que se pueden considerar complementarios.

ECGI, de alguna manera es una forma compacta de almacenar *todos* las cadenas de R^+ como patrones, postulando además una generalización sobre ellas. En reconocimiento ECGI lleva efectivamente a cabo DTW "en

paralelo" para todos los patrones (y su generalización). Ahora bien, en el caso de los dígitos hablados, los trellis asociados a ECGI tienen un número menor de vértices que en el caso de DTW con sólo 5 patrones/clase (≈130 vértices por símbolo de entrada, longitud media de las muestras: 30), obteniendo resultados equivalentes (mejores), aún en el caso de tener muy pocos símbolos discretos (≈15); gracias precisamente a su poder de generalización.

De hecho, ECGI con símbolos discretos ha demostrado proporcionar resultados equivalentes o mejores que un DTW con vectores de parámetros y 18 a 20 patrones por clase, utilizando el mismo corpus de datos de dígitos hablados (aunque distribuídos de manera distinta) [Vidal,88b]. En el caso multilocutor, DTW obtuvo un 99% de aciertos, ECGI (experimento H2 capítulo 8) 100%; en el caso independiente del locutor DTW obtuvo 97% y ECGI un 98% a 100% (experimentos H1, H3, H4 y H5 y H6).

La complejidad temporal crece en ambos casos linealmente con el número de patrones/estados, y aunque es cierto que la complejidad de un DTW se puede reducir grandemente utilizando métodos eficientes de búsqueda de los vecinos más próximos [Vidal,88a], también es cierto, como se ha visto antes, que la complejidad de ECGI se puede reducir considerablemente.

Es posible considerar al ECGI como un modelo intermedio entre DTW y HMM, que pretende aprovechar las ventajas de uno y otro (utilizar todas las muestras como patrones de referencia, obtener directamente de ellas el modelo estructural y los valores de los parámetros).

Del mismo modo que DTW, mediante ECGI se podría obtener provecho de la utilización de vectores de parámetros en vez de símbolos discretos; de hecho actualmente ésta es una de las primeras mejoras que se prevee implementar (ver epílogo y apéndice A).

12.3 Métodos en Reconocimiento de Imágenes

Los mismos experimentos de reconocimiento de formas realizados con dígitos manuscritos e impresos han sido realizados utilizando métodos de parametrización y reconocimiento más clásicos en reconocimiento de imágenes (manuscritos [Vidal,91] [Vidal,92a], impresos [Vidal,92]).

Como parámetros se emplearon *momentos geométricos* [Teague,80] [The,88]: 7 momentos centrales normalizados del área del objeto (dígito) y 7 momentos centrales normalizados del contorno del mismo. Estos momentos aseguran una invarianza a la escala y a la traslación, pero no a la

rotación ni a la reflexión. Estos dos últimos tipos de invarianza implicarían el recurso a *momentos invariantes*, los cuales resultan inadecuados en nuestro caso, pues algunos dígitos se pueden convertir en versiones muy similares de otros a través de una serie de reflexiones y giros [Vidal,91].

Los 14 parámetros así obtenidos forman un punto en un espacio 14-dimensional, el cual se clasifica siguiendo la regla de los k vecinos más próximos (k-NN) [Duda,73]), empleando como prototipos *todas* las muestras de aprendizaje (200 por clase para el caso de los dígitos manuscritos, 280 en el caso de los impresos, y un valor de k igual a 45 y 40 respectivamente).

Para poder tener en cuenta la diferente naturaleza de los grupos de componentes de los vectores, se escoge una distancia ponderada, concretamente, la *distancia de Mahalanobis* con matriz de covarianza diagonal:

$$d(x,y)=\left(\sum_{i=1}^{m}\frac{(xi-yi)}{\sigma_i}\right)^{1/2}$$

donde x,y son los vectores a comparar, y σ_i, $1\leq i\leq m$ (m=14 en nuestro caso), son las varianzas de cada componente estimadas a partir de la totalidad del conjunto de datos.

Los experimentos realizados se corresponden con los experimentos de "leaving-k-out" de dígitos manuscritos e impresos. En el caso de los impresos, se realizaron eliminando el tipo Helvética del conjunto de datos. La tabla 12.4 evidencia la superioridad de ECGI para las tareas planteadas.

Tabla 12.4 Resumen de los resultados comparativos para los experimentos "leaving-k-out" de dígitos manuscritos e impresos. K-NN indica los resultados del método clásico, el resto son resultados de ECGI para distintas rejillas. El tamaño de los modelos es el número de estados para ECGI, y el número de prototipos por clae para K-NN (K=45 para los dígitos manuscritos, k=40 para los impresos).

Método	ECGI Estocástico, Sólo substituciones.				K-NN
Resolución (Pixels)	Rej4	Rej6	Rej8	Rej10	1
Dígitos manuscritos					
Tamaño medio de los modelos	310	223	176	144	200
%Aciertos	98,0	98,15	96,92	96,31	81,5
Dígitos impresos					
Tamaño medio de los modelos	249	170	130	111	240
%Aciertos	99,18	98,75	98,14	97,32	90,2

Por otra parte, también con el mismo corpus de dígitos manuscritos, se han llevado a cabo experimentos de reconocimiento con redes neuronales [Marzal,91] con resultados (independientes del escritor) del orden del 73% de aciertos. En el mismo trabajo se intenta asimismo la tarea de reconocimiento mediante Modelos de Markov, consiguiendo en este caso llegar hasta un 86,8% de aciertos, resultados todos ellos inferiores a los proporcionados por ECGI.

En el caso más general, y considerando el estado del arte expuesto en el capítulo 1 (98 a 99% en dígitos manuscritos aislados), se puede afirmar que el ECGI se sitúa entre los mejores métodos utilizados para el reconocimiento de dígitos, ello aún sin introducir refinamientos específicos a la tarea (parametrización más adecuada, invarianza a la rotación, etc...) que podrían aumentar notablemente la sus capacidades de reconocimiento.

Epílogo

Conclusiones

En este trabajo se ha propuesto, estudiado e implementado, un nuevo método incremental de inferencia gramatical: el método ECGI.

Tal como se ha explicado detalladamente en el capítulo 6, ECGI recurre a una poderosa herramienta de comparación de objetos y formas, la *corrección de errores* (implementada mediante técnicas de programación dinámica), para construir modelos de formas y (simultáneamente, si se necesario) reconocer nuevos objetos. Gracias a esta herramienta, ECGI es capaz de encontrar qué subestructuras es necesario añadir al modelo para incorporarle, con un coste mínimo de modificación, cada una de las nuevas muestras.

Junto con otros cuatro métodos, unos igualmente recientes (métodos de Thomason y Falaschi), otros poco estudiados (métodos de Chirathamjaree y de Marco) (capítulo 12), ECGI forma una nueva clase de métodos de inferencia. Esta clase, compuesta por los métodos que emplean la misma técnica que ECGI u otras técnicas similares, también basadas en la programación dinámica y en la corrección de errores, contribuye a engrosar el (hoy por hoy más bien parco) conjunto de los métodos de inferencia gramatical. De entre éstos, el método ECGI se coloca entre los que mejores resultados han obtenido en aplicaciones prácticas.

Si bien, como ha quedado patente en su lugar (capítulos 3 y 6), los algoritmos de la clase del ECGI son métodos heurísticos y difícilmente caracterizables, los resultados obtenidos (capítulo 8) demuestran que el ECGI está sorprentemente bien adaptado a determinadas aplicaciones reales de reconocimiento de formas. Incluso en condiciones de forzada suboptimalidad (experimentos con autómatas deterministas, capítulo 11), y con información muy escasa y de pobre calidad (experimentos piloto,

capítulo 8) se obtienen tasas de reconocimiento elevadas, a veces idénticas a las obtenidas en las mejores condiciones. Ello permite introducir nuevos heurísticos adaptados al problema, en la forma (por ejemplo) de distintos criterios de distancia (relacionados o no con los tres estudiados en el capítulo 6), en la confianza de que, aún siendo su evaluación subóptima, se van a lograr resultados de alcance práctico.

Los resultados que proporciona ECGI en aplicaciones de reconocimiento de formas (capítulo 9) son perfectamente comparables, e incluso generalmente superiores, a los obtenidos aplicando, a los mismos objetos (particularmente imágenes y palabras), técnicas más convencionales como pueden ser los Modelos de Markov y varios modelos basados en la teoría de la decisión (DTW,K-Vecinos, momentos geométricos con distancia de Mahalanobis, etc.) (capítulo 12). En experimentos de reconocimiento de dígitos hablados independientes del locutor se obtiene sin dificultad un 99,8% de aciertos, mientras que para diccionarios hablados notablemente más difíciles (EE-letras) aún se consigue un 73,6% de aciertos. En reconocimiento de imágenes planas los resultados son del orden del 98,2% de aciertos (dígitos manuscritos) y del 99,9% (dígitos impresos), habiendo sido obtenidos estos últimos resultados sin ninguna adaptación de ECGI al problema y con un método relativamente simple de parametrización.

Por otra parte, la complejidad (temporal y espacial) de ECGI en tareas de reconocimiento de formas ha resultado ser similar, y a menudo inferior, a la de estos mismos métodos convencionales, siendo el coste del reconocimiento de una muestra de $O(|\alpha| \cdot B \cdot |Q|)$, donde α es la cadena muestra, B el factor de ramificación medio (entre 1,6 y 3,8) y Q el conjunto de estados del modelo (talla normalmente entre 200 y 700 estados). Para el caso de que la complejidad de ECGI fuera, a pesar de todo, una desventaja, se ha estudiado la posibilidad de simplificar los modelos inferidos (capítulo 10), comprobándose que es factible reducir la complejidad espacial de los mismos en un 40% a 60%, sin merma de la tasa de reconocimiento. Además, trabajos realizados en paralelo con éste demuestran que es posible reducir, también sin pérdida de eficacia, la complejidad temporal en un factor 10.

Asimismo, en los diversos experimentos presentados en los capítulos 8 y 9, ha quedado patente que es posible reducir la complejidad temporal de ECGI en la fase de reconocimiento en un 30-40%, sencillamente y sin reducción de eficacia (e incluso con un aumento de ésta), con sólo restringir el modelo de error, prohibiéndole inserciones y borrados. Ello es tanto más cierto cuanto con más detalle se haya inferido el modelo (muchas muestras y/o resolución)

Discutiendo las condiciones de convergencia de ECGI (capítulo 9) se ha mostrado que los modelos inferidos por ECGI nunca dejan realmente de crecer. Sin embargo, un razonamiento y una demostración empírica confirman que en condiciones reales, y siempre que las muestras no presenten una variabilidad excesiva, el ECGI converge asintóticamente

proporcionando modelos eficaces con un número de muestras relativamente reducido (≈200).

ECGI infiere gramáticas regulares (de tipo SANSAT: "SAme Non-terminal, then SAme Terminal", ver capítulo 2) sin circuitos, en general ambiguas y no deterministas. Se ha propuesto un método (muy heurístico y subóptimo) que permite obligarle a inferir gramáticas deterministas (y por lo tanto no ambiguas) *sin pérdida alguna* de eficacia en reconocimiento (capítulo 11).

Por otra parte, simultáneamente con todos estos estudios centrados en el método ECGI, y como consecuencia más o menos directa de ellos, se han desarrollado otros trabajos que pueden considerarse más marginales. Estos trabajos han proporcionado resultados que tienen interés, no sólo en el marco de ECGI, sino en otros ámbitos más generales del reconocimiento sintáctico de formas y/o la teoría de lenguajes. En particular:

Se ha mostrado la factibilidad y convergencia de un algoritmo que permite la corrección de errores en todo tipo de gramáticas regulares, independientemente de que tengan o no circuitos (véase algoritmo `ciclico`, capítulo 5).

Se ha demostrado que toda gramática regular tiene una gramática SANSAT equivalente y un autómata de estados etiquetados (LAS: "LAbelled State automata") también equivalente (capítulo 2). Ello proporciona un método de modelización de lenguajes que en ocasiones puede ser útil en reconocimiento de formas.

Se ha definido una nueva manera de asegurar la consistencia de las probabilidades de una gramática estocástica, expandida mediante un modelo de error convencional. Este modelo asegura la consistencia *de las reglas de error* y reduce la complejidad del análisis sintáctico cuando se utiliza el algoritmo de Viterbi. El modelo asume probabilidades de inserción independientes de la posición y del número de reglas asociadas a un no terminal.

Se ha introducido una técnica que permite la simplificación progresiva de gramáticas regulares estocásticas (o no), siempre que no tengan circuitos. Accesoriamente, se ha definido una cantidad (el "tráfico") que permite cuantificar la importancia estructural de un vértice o un arco pertenecientes a un grafo sin circuitos.

Se ha propuesto y comprobado un método para obtener la "cadena mediana" de un conjunto de cadenas (capítulo 11), utilizando la cadena más probable de la gramática inferida por ECGI para dicho conjunto de cadenas. Como alternativa para simplificar el cálculo de la cadena más probable, se han propuesto la *cadena más frecuente* y la *cadena localmente más frecuente*. Esta última permite obtener una buena aproximación a la cadena

mediana sin necesidad de recurrir a una búsqueda por optimización en la gramática inferida por ECGI.

Perspectivas

Muchas de las perspectivas que, en su momento, abrió ECGI al proporcionar los primeros resultados esperanzadores, han sido exploradas durante los años transcurridos desde entonces hasta la finalización de este texto (apéndice A): mejoras de complejidad temporal mediante búsqueda en haz, reconocimiento anticipado y examen tan sólo de los estados alcanzados [Torró,89]; utilización de las probabilidades durante el reconocimiento y reestimación estocástica de las mismas [Castaño,90]; modificación del algoritmo de inferencia para permitirles utilizar reglas que no pertenencen todas a la misma derivación [Miralles,91]; aplicación de ECGI para la inferencia de estructuras de modelos de Markov [Casacuberta,90] [Casacuberta,90a], para la inferencia de modelos de unidades subléxicas [Carpi,90] [Sanchis,91] [Tarazona,91], para la inferencia de modelos de lenguajes [Prieto,91] [Prieto,91a]; extensión de ECGI para la utilización de modelos semicontinuos [Arévalo,90],... La mayoría de estos trabajos han abierto a su vez nuevos posibles caminos y han dejado por resolver algunas cuestiones. En cualquiera de estos campos queda pues trabajo por hacer, habiendo demostrado el ECGI, no sólo ser una herramienta eficaz de reconocimiento, sino un método flexible, aplicable a muchos campos del reconocimiento de formas y suceptible de variadas extensiones y mejoras.

En general, las direcciones de más interés hacia la que se orientan, hoy por hoy, los trabajos sobre ECGI, se centran alrededor de la mejora de su capacidad de reconocimiento, recurriendo a las mismas técnicas que se utilizan en los modelos de Markov cuando es imposible sacar más partido a los símbolos provenientes de la cuantificación vectorial: el empleo de modelos (semi)continuos. Con estas aproximaciones, se espera mejorar notablemente las tasas de reconocimiento de ECGI a costa, eso sí, de un cierto aumento de la complejidad temporal (aunque probablemente pueda mantenerse inferior a la de los modelos de Markov en el mismo caso).

Por otra parte, existe la posibilidad extremadamente interesante de utilizar ECGI en reconocimiento del habla continua. La idea en este caso consiste en concatenar modelos aprendidos mediante el ECGI (de fonemas u otras unidades subléxicas) siguiendo reglas aprendidas por el mismo ECGI (modelos de lenguaje). En esto también se sigue la aproximación seguida anteriormente por otros autores, que construyen (manualmente) modelos de lenguajes a partir de la composición de modelos de Markov [Lee,88]. Los trabajos ya realizados en este campo (ver apéndice A) han ofrecido perspectivas interesantes, con buenos (no excelentes) resultados.

Apèndice A

Otros trabajos sobre ECGI

En paralelo con este trabajo se han realizado, en el interior del mismo grupo de investigación, varios estudios e implementaciones práticas destinadas a mejorar las capacidades del ECGI desde diversos puntos de vista y a utilizarlo en diversas aplicaciones. Estos trabajos, derivados de éste, lo complementan, por lo que se reseñan a continuación.

Isabel Alfonso, en [Alfonso,91] llevó a cabo una implementación de ECGI que funciona a tiempo real, aprovechando las técnicas desarrolladas por F.Torró para disminuir la complejidad temporal de ECGI. Esto ha permitido comparar los tiempos de respuesta reales de ECGI con los de otras métodos de reconocimiento de formas (modelos de Markov, Redes neuronales), obteniéndose resultados significativamente favorables a ECGI.

Manuél Arévalo, en [Arévalo,90] implementa una version semicontinua de ECGI, es decir, se estima una función de "densidad de probabilidad" en cada estado del modelo, que es utilizado entonces, no para inferir o reconocer a partir de símbolos discretos, sino a partir directamente de los vectores de parámetros (coeficientes cepstrales en este caso).

Francisco Casacuberta, en [Casacuberta,91] y junto con Begonia Mas, Enrique Vidal y H.Rulot en [Casacuberta,90], [Casacuberta,90a] utilizan los procedimientos de simplificación de autómatas descritos en el capítulo 10 para obtener, mediante simplificación de modelos inferidos por ECGI, la estructura de unos modelos de Markov que posteriormente entrenan con las técnicas clásicas de este campo. Finalemente, comparan los resultados obtenidos con los que proporciona ECGI, consiguiendo demostrar (ver capítulo 12) que los modelos inferidos por ECGI son, por lo menos, igualmente eficaces que modelos de Markov de no menos de 30 estados y que los métodos de simplificación propuestos en este trabajo conservan la información estructural de los modelos inferidos.

Asunción Castaño, en [Castaño,90] introdujo técnicas de reestimación estocástica para mejorar la fiabilidad de las probabilidades de los modelos inferidos por ECGI. Excepto para las primera N (N arbitrario) muestras, la

inferencia es estocástica (se utilizan las probabilidades en la búsqueda de la derivación óptima). Cada N muestras siguientes se reestiman las probabilidades, utilizando el nuevo conjunto de frecuencias. El proceso se repite hasta que se cumpla cierto criterio de parada (p.e.: convergencia de las probabilidades). Los resultados obtenidos en reconocimiento, si bien no mejoran los de ECGI convencional, muestran la factibilidad de la idea de la reestimación de probabilidades.

Mª Isabel Galiano, en [Galiano, 90] introdujo un nuevo heurístico que le permitía simplificar autómatas inferidos por ECGI. A estos modelos simplificados, se les añadía luego un modelo duracional en los estados y se procedía a una reestimación de probabilidades. Las tasas de reconocimiento obtenidas no fueron todo lo buenas que se esperaba, aunque una revisión de la idea ha justificado la puesta en marcha de otro proyecto, actualmente en curso y que aún no ha proporcionado resultados.

Felipe Miralles, en [Miralles,91] modifica los heurísticos de inferencia de ECGI, para permitir que la que la secuencia de reglas identificada como la que genera la cadena más próxima a la cadena muestra *pueda no ser una derivación permitida* por la gramática actual en el momento de recibir la muestra (sólo se requiere que sea una combinación de fragmentos de derivaciones permitidas). La modificación realizada mantiene las características de las gramáticas inferidas por ECGI y produce en general modelos menos complejos, aunque con una mayor generalización (tamaño del lenguaje). Se introduce también un modelo de aprendizaje con muestras positivas y negativas. Sin embargo, los resultados de reconocimiento no han mostrado una clara mejora con respecto al ECGI convencional.

Natividad Prieto, H.Rulot, Emilio Sanchis y Enrique Vidal en [Prieto, 88] utilizaron el mismo corpus de letras habladas descrito en el capítulo 9, pero parametrizado con menos precisión (frecuencia de muestreo relativamente baja, clustering con pocos símbolos), obteniendo los resultados que se presentan en la tabla A.1. Estos resultados, como era de esperar por la inferioridad de la parametrización, son inferiores a los presentados en el capítulo 8, aunque a pesar de todo comparables a los que se obtienen con otros métodos de reconocimiento. La razón principal por la que se han incluído aquí es debido a que permiten observar como las tasas de reconocimiento emperoran generalmente (-0.5%) en el **caso de utilizar 10 tablas de sustitución** en vez de una única tabla promedio (ver capítulo 7) (aunque mejoran mucho más en un caso: +3%).

Tabla A.1 Tasas de aciertos y número de estados en los experimentos con las letras (aprendizaje 8 repeticiones por locutor y letra, 10 locutores, test 2 repeticiones -diferentes- por locutor y letra).

Experimento:	(L1) H,I,J,K,L,M,N,O,P,Q	(L2) F,L,LL,M,N,Ñ,R,S
Numero medio de estados	240	346
No estocástico	80,0%	60%
Estocástico		
Modelo de error Completo	83,5%	65,0%
Idem, pero 10 Tablas Sus.	86,5%	64,5%
Sólo substitución	79,0%	60,6%
Idem, pero 10 Tablas Sus.	78,5%	60,6%
Sin probabilidades de error	83,5%	61,2%

Natividad Prieto y Enrique Vidal, en [Prieto,91] y [Prieto,91a] proponen un método para la aplicación de ECGI a la inferencia de *modelos de lenguaje* y a la comprensión del habla continua. En estas técnicas se usa de forma homogénea el mismo algoritmo ECGI, tanto para el aprendizaje de un modelo de lenguaje semántico, como para el de los modelos acústicos asociados a las distintas categorías semánticas. Estos trabajos abren una nueva perspectiva en el desarrollo de aplicaciones de reconocimiento del habla continua en universos de semántica limitada, los cuales son los que suelen encontrarse en situaciones reales de aplicación práctica.

Emilio Sanchis y Francisco Casacuberta, en [Sanchis,90] y [Sanchis,91], y junto con Mª Isabel Galiano en [Galiano,91], utilizan el ECGI para la inferencia de modelos para unidades del habla de tamaño inferior a la palabra (fonemas, sílabas,...), con el fin de generar grandes vocabularios por concatenación de dichos modelos (siguiendo reglas posiblemente también inferidas por ECGI, aplicando los procedimientos propuestos en por N.Prieto). Otros trabajos basados también en la misma idea son los de *Sofía Carpi* [Carpi,90] y *Joaquín Tarazona* [Tarazona,91], que aprenden modelos de fonemas y los utilizan luego para llevar a cabo la decodificación acústico-fonética mediante el algoritmo de un paso (J.Tarazona utiliza una versión semicontinua de ECGI, como M.Arévalo en [Arévalo,91]).

Francesc Torró, en [Torró,90] propone y estudia varios métodos que permiten reducir drásticamante la complejidad temporal de ECGI durante el reconocimiento. Utilizando técnicas de búsqueda en haz, reconocimiento anticipado y analizando sólo los estados alcanzados, obtiene las mismas tasas de aciertos visitando tan sólo un 9% de los vértices del trellis.

Apéndice B

Tablas complementarias

B.1 Número de estados de los autómatas inferidos

Se muestran a continuación una serie de tablas que contienen el número de estados de los autómatas inferidos en la mayoría de los experimentos presentados en el capítulo 8.

B.1.1 Dígitos Hablados:

	H1:	H2:	H3:	H4:	H5:	H6:	piloto
0	133	171	157	130	169	153	126
1	152	144	141	118	149	159	100
2	121	135	131	113	150	129	73
3	102	120	121	97	126	107	95
4	171	182	171	178	165	162	129
5	233	270	238	210	258	243	136
6	104	121	114	126	116	104	85
7	185	209	183	186	196	181	104
8	136	149	203	140	141	147	105
9	196	220	203	171	242	202	110
Total	1533	1721	1599	1469	1712	1587	1063

HLKO11	H11:	H22:	H33:	H44:	H55:
0	193	175	187	178	180
1	179	177	194	182	186
2	150	155	140	138	141
3	126	141	131	121	131
4	213	228	217	203	222
5	340	314	326	305	332
6	131	148	156	123	149
7	236	250	236	216	223
8	188	175	186	174	175
9	225	242	223	220	243
	1981	2005	1996	1860	1982

B 1.2 Letras Habladas:

L1:		L2:		L3:				L4:			
H	854	F	763	LL	882	M	784	LL	592	M	561
I	332	L	797	RR	914	N	696	RR	696	N	468
J	676	LL	882	CH	495	O	271	CH	387	O	243
K	316	M	784	A	304	P	472	A	281	P	356
L	797	N	696	B	514	Q	271	B	393	Q	210
M	784	Ñ	901	C	781	R	708	C	583	R	497
N	696	R	708	D	541	S	817	D	388	S	584
O	271	RR	967	E	478	T	470	E	346	T	346
P	531	S	817	F	763	U	235	F	574	U	199
Q	309			G	624	V	697	G	436	V	496
				H	838	W	1202	H	626	W	810
				I	332	X	681	I	249	X	544
				J	676	Y	1218	J	452	Y	825
				K	316	Z	959	K	276	Z	670
				L	752	Ñ	901	L	568	Ñ	631
5566		7315		17502				14287			

LLKO	L11:	L22:	L33:	L44:	L55:
F	823	787	759	753	721
L	849	732	766	731	768
LL	915	847	860	828	762
M	780	756	811	734	751
N	721	690	674	638	619
Ñ	885	870	895	834	834
R	717	663	702	672	653
RR	954	924	953	905	898
S	843	790	757	774	736
	7487	7059	7177	6869	6742

B.1.3 Dígitos Manuscritos:

Experimento con Rejilla 4

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12
cero	225	218	222	218	219	220	218	214	213	218	163	152
uno	250	263	246	242	238	239	233	235	208	202	237	271
dos	364	357	361	391	386	379	383	391	389	386	381	365
tres	351	365	355	368	357	345	353	370	371	365	363	349
cuat	315	340	352	357	342	313	325	353	359	356	355	335
cinco	380	349	354	357	349	341	349	352	358	359	344	336
seis	257	252	255	255	247	235	242	248	247	252	257	273
siete	365	370	367	387	383	376	382	385	383	381	359	349
ocho	309	295	310	331	322	319	322	302	298	319	324	320
nueve	279	281	285	285	262	255	285	283	280	284	275	285
	3096	3090	3107	3191	3105	3022	3092	3133	3106	3122	3058	3035

Experimento con Rejilla 6

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12
cero	149	149	157	153	152	152	153	153	151	152	112	106
uno	168	179	190	190	188	189	185	186	171	166	188	181
dos	268	245	251	263	257	262	263	271	268	268	264	273
tres	278	258	248	246	236	235	239	245	246	244	243	245
cuat	245	239	251	248	240	211	222	244	246	248	248	236
cinco	261	268	274	276	274	273	274	278	280	279	273	254
seis	197	183	182	176	174	166	172	177	176	181	182	184
siete	257	283	267	271	263	259	264	279	278	275	245	261
ocho	229	211	215	243	235	236	235	223	223	237	241	229
nueve	205	206	211	218	193	190	204	204	203	207	200	196
	2257	2221	2246	2284	2212	2173	2211	2260	2242	2257	2196	2165

Experimento con Rejilla 8

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12
cero	118	120	121	119	120	118	119	118	117	118	85	91
uno	137	140	132	147	146	144	137	138	134	126	142	136
dos	202	205	197	206	208	208	207	212	213	211	208	204
tres	223	206	210	202	205	198	191	199	205	203	200	211
cuat	179	185	193	193	186	168	176	194	198	193	190	180
cinco	210	217	218	219	213	213	218	226	226	228	220	192
seis	152	140	142	136	127	128	140	142	141	144	146	153
siete	208	217	217	218	226	223	225	229	225	230	212	198
ocho	159	162	170	177	174	171	172	165	166	169	172	183
nueve	163	153	166	168	139	139	169	165	165	166	160	160
	1751	1745	1766	1785	1744	1710	1754	1788	1790	1788	1735	1708

Experimento con Rejilla 10

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12
cero	94	91	96	91	90	96	95	95	94	96	70	73
uno	111	116	115	112	108	109	110	110	97	94	109	111
dos	187	155	163	177	177	174	176	180	186	184	176	171
tres	185	173	173	173	168	164	161	167	166	163	164	168
cuat	146	145	150	151	148	140	144	151	154	153	152	151
cinco	170	176	186	193	189	183	185	189	190	190	187	159
seis	119	119	122	120	121	119	118	120	122	123	124	126
siete	174	183	181	187	183	181	182	181	183	183	172	152
ocho	147	144	136	148	141	139	145	134	132	135	141	137
nueve	141	136	133	134	113	117	137	132	131	132	128	127
	1474	1438	1455	1486	1438	1422	1453	1459	1455	1453	1423	1375

Experimento con Rejilla 8

	0	1	2	3	4	5	6	7	8	9	(Clase correcta)
cero	.	.	.	.	.	.	.	.	.	.	
uno	.	.	.	.	6	.	.	12	.	5	
dos	.	5	.	.	.	.	.	1	.	.	
tres	.	.	.	.	.	.	.	.	3	18	
cuat	.	.	.	.	.	.	.	.	1	8	
cinco	.	.	.	.	.	.	.	.	.	13	
seis	3	.	.	.	.	.	.	.	3	.	
siete	.	.	1	.	5	.	.	.	4	.	
ocho	2	.	.	.	.	2	2	.	.	1	
nueve	.	19	.	.	16	.	.	.	18	.	
Errores por clase:											
	5	24	1	0	27	2	2	13	29	45	

148 errores de 4800 muestras: 96.92% aciertos

Experimento con Rejilla 10

	0	1	2	3	4	5	6	7	8	9	(Clase correcta)
cero	.	.	.	.	.	.	.	.	13	2	
uno	.	.	.	.	8	.	1	11	3	10	
dos	.	16	.	.	.	.	.	1	.	.	
tres	.	.	.	.	.	.	.	.	6	26	
cuat	.	.	.	.	.	.	.	1	1	6	
cinco	.	.	.	4	.	.	.	.	.	.	
seis	.	.	.	.	.	.	.	.	7	.	
siete	.	.	1	.	4	.	.	.	3	5	
ocho	6	.	.	.	.	.	.	.	.	10	
nueve	.	2	.	1	19	.	.	.	10	.	
Errores por clase:											
	6	18	1	5	31	0	1	13	43	59	

177 errores de 4800 muestras: 96.31% Aciertos

B.2.2 Dígitos impresos (sólo substitución)

Experimento con Rejilla 4

	0	1	2	3	4	5	6	7	8	9	(Clase correcta)
cero	.	.	.	.	.	.	.	.	.	.	
uno	.	.	.	.	.	.	.	.	.	.	
dos	.	.	.	.	.	.	.	.	.	.	
tres	.	.	.	.	.	.	.	.	.	.	
cuat	.	.	.	.	.	.	.	.	.	.	
cinco	.	.	.	.	.	.	.	.	.	.	
seis	.	.	.	.	.	.	.	.	.	.	
siete	.	1	.	.	.	.	.	.	.	.	
ocho	.	.	.	.	.	.	.	.	.	.	
nueve	.	.	.	.	.	.	.	.	.	.	
Errores por clase:											
	0	1	0	0	0	0	0	0	0	0	

1 errores de 3200 muestras: 99.97%

Experimento con Rejilla 6

debe ser->	0	1	2	3	4	5	6	7	8	9	(Clase correcta)
cero	.	.	.	.	.	.	.	.	1	.	
uno	.	.	.	.	.	.	.	.	.	.	
dos	.	.	.	.	.	.	.	.	.	.	
tres	.	.	.	.	.	.	.	.	.	.	
cuat	.	.	.	.	.	.	.	.	.	.	
cinco	.	.	.	.	.	.	.	.	.	.	
seis	.	.	.	.	.	.	.	.	.	.	
siete	.	11	.	.	.	.	.	.	.	.	
ocho	1	.	.	.	.	.	4	.	.	1	
nueve	.	.	.	.	.	.	.	.	.	.	
Errores por clase:	1	11	0	0	0	0	4	0	1	1	

18 errores de 3200 muestras: 99.44%

Experimento con Rejilla 8

	0	1	2	3	4	5	6	7	8	9	(Clase correcta)
cero	.	3	.	.	.	.	.	.	7	.	
uno	.	.	.	.	.	.	.	.	.	.	
dos	.	.	.	.	.	.	.	.	.	.	
tres	.	.	.	.	.	.	.	.	.	.	
cuat	.	.	.	.	.	.	.	.	.	.	
cinco	.	.	.	.	.	.	.	.	.	.	
seis	.	.	.	.	.	.	.	.	.	.	
siete	.	4	.	.	.	.	.	.	.	.	
ocho	1	.	.	2	.	.	8	.	.	1	
nueve	.	.	.	9	.	.	.	.	.	.	
Errores por clase:	1	7	0	11	0	0	8	0	7	1	

35 errores de 3200 muestras: 98.91%

Experimento con Rejilla 10

	0	1	2	3	4	5	6	7	8	9	(Clase correcta)
cero	.	1	.	.	.	.	.	.	7	.	
uno	.	.	.	.	.	.	.	.	.	.	
dos	.	.	.	.	.	.	.	.	.	.	
tres	.	.	.	.	.	.	.	.	1	.	
cuat	.	.	.	.	.	.	.	.	.	.	
cinco	.	.	.	.	.	.	.	.	.	.	
seis	1	.	.	.	.	.	.	.	.	.	
siete	.	1	.	.	.	.	.	.	.	.	
ocho	.	.	.	.	.	.	2	.	.	.	
nueve	.	.	.	.	.	.	.	.	.	.	
Errores por clase:	1	2	0	0	0	0	2	0	8	0	

13 errores de 3200 muestras: 99.59%

Bibliografía

[Aho,72] A.Aho, T.G.Peterson: *"A Minimum Distance Error-Correcting Parser for Context-Free Languages"*. SIAM Journal, Vol.1, Nº4, pp.305, 1972.

[Aho,73] A.Aho, J.Ullman: *"The Theory of Parsing, Traslation and Compiling"*. Vol.1, "Parsing", Prentice-Hall, 1973.

[Alfonso,91] I.Alfonso: *"Sistema On-Line de aprendizaje-Reconocimiento de Palabras Aisladas"*. Proyecto fin de carrera, DSIC, Universidad politécnica de Valencia, 1991.

[Alpuente,89] M.Alpuente, F.Casacuberta: *"Esquema de Traducción Dirigido por la Sintaxis Para la Corrección de Errores en Lenguajes Regulares"*. Rev. de Informática y Automática, Vol.22, Nº2, 1989.

[Angluin,80] D.Angluin: *"Inductive Inference of Formal Languges from Positive Data"*. Information and Control, 45, pp.337-350, 1980.

[Angluin,82] D.Angluin: *"Inference of Reversible Languages"*. Journal of the ACM, Vol.29, Nº 3, pp.741-765.

[Angluin,83] D.Angluin, C.H.Smith: *"Inductive Inference: Theory and Methods"*. Computing Surveys, Vol.15, Nº 3, 1983.

[Arévalo,90] M.Arévalo: *"Extensiones de algoritmo ECGI en el marco del Reconocimiento Automático del Habla"*. Proyecto fin de carrera, DSIC, Universidad Politécnica de Valencia, 1990.

[Averbuch,87] A.Averbuch, L.Bahl, P.Brown & al.: *"Experiment with the Tangora 20.000 word Speech Recognizer"*. IEEE-ICASSP proc., pp.701-704, 1987.

[Bahl,75] L.Bahl, F.Jelinek: *"Decoding for Channels with Insertions, Deletions and Substitutions with Applications to Speech Recognition"*. IEEE Trans. on Information Theory, Vol IT-21, Nº 4,July 1975.

[Bahl,87] L.Bahl, P.F.Brownm P.V.de Souza, R.L.Mercer: *"Speech Recognition With Continous-Parameter Hidden Markov Models"*. Computer Speech and Language, Vol.2, pp.219-234, 1987.

[Baker,91] J.Baker: *"Large Vocabulary Speaker-Adaptative Continous Speech Recognition Research Overview at Dragon Systems"*. EUROSPEECH proc., pp.29-32, 1991.

[Bamberg,91] P.Bamberg, A.Demedts, J.Elder, C.Huang & al.: *"Phoneme-Based Trainig for Large-Vocabulary Recognition in Six European Languages"*. EUROSPEECH proc., pp. 175-181, 1991.

[Baptista,88] G.Baptista, K.Kulkarni: *"A High Accuracy Algorithm for Recognition of Handwritten Numerals"*. Pattern Recognition, Vol.21, Nº4, pp.287-291, 1988.

[Bellman,57] R.Bellman: *"Dynamic Programming"*. Princeton University Press, Princeton N.J., 1957.

[Benedi,89] J.M.Benedi. *"Estudio de un Sistema de Reconocimiento Automático del Habla: Tabarca"*. Tesis Doctoral. Univ. Pol. de Valencia. 1989.

[Biermann,72] A.W.Biermann, J.A.Feldman: *"On the Synthesis of Finite-state Machines from Samples of Their Behavior"*. IEEE Trans. on Computers, Vol.C-21, pp.592-597, 1972.

[Bouloutas,91] A. Bouloutas, G.W.Hart & M.Schwartz: *"Two Extensions to the Viterbi Algorithm"*, IEEE-Trans. on Information Theory, Vol 37, nº 2, March 1991.

[Bribiesca,80] E.Bribiesca, A.Guzman: *"How to desribe pure form and how to measure differences"*, Pattern Recognition, Vol.12, Nº 2, pp. 101-102, 1980.

[Brown,88] R.M.Brown, T.H.Fay, C.L.Walker: *"Handprinted Symbol Recognition System"*. Pattern Recognition, Vol.21, Nº2, pp.91-118, 1988.

[Carpi,90] S.Carpi: *"Aprendizaje de Modelos Fonéticos para el Reconocimiento Automático del Habla"*. Proyecto fin de carrera, DSIC, Universidad Politécnica de Valencia. 1990.

[Casacuberta,87] F,Casacuberta, E.Vidal: *"Reconocimiento Automático del Habla"*. Marcombo. 1987.

[Casacuberta,90] F.Casacuberta, E.Vidal, B.Mas, H.Rulot: *"Learning the structure of HMM's Through Grammatical Inference Techniques"*. IEEE-ICASSP proc., pp.717-720, April 1990.

[Casacuberta,90a] F.Casacuberta, E.Vidal, H.Rulot, B.Mas: *"Use of the Grammatical Inference Algorithm ECGI for Finding the Topology of Hidden Markov Models"*. International Conference of Speech Technologies. VERBA/90 proc. pp.99-105, Jan 22-24, 1990.

[Casacuberta,91] F.Casacuberta: *"Modelos de Markov Ocultos y Reconocimiento de Palabras Aisladas"*; DSIC-II/2/91, Universidad Politécnica de Valencia. 1991.

[Castaño,90] M.A.Castaño: *"Reestimación Estocástica en el Algoritmo de Inferencia Gramatical ECGI"*. Proyecto Fin de Carrera, DSIC, Univ. Pol. de Valencia, 1990.

[Chandhuri,86] R.Chandhuri, A.N.V.Rhao: *"Aproximating Grammar Probabilities: Solution of a Conjecture"*. Journal of the ACM, Vol 33, Nº 4, pp.702-705, 1986.

[Chirathamjaree,80] C.Chirathamjaree, M.H.Ackroyd: *"A Method for the Inference of Non-Recursive Context-Free Grammars"*. Int. Journal of Man-Machine Studies, pp.379-387, Vol.12, 1980.

[Cerf-Danon,91] H.Cerf-Danon, S.DeGennaro,M.Ferretti,J.Gonzalez,E.Keppel: *"TANGORA - a Large Vocabulary Speech Recognition System for Five Languages"*. EUROSPEECH proc., pp.183-192, 1991.

[Cohen,82] P.Cohen, E.Feigenbaum: *"The Handbook of Artificial Intelligence"*. Vol. 1,2,3, Pitman Books, 1982.

[Davies,81] E.R. Davies and A.P. Plummer: *"Thinning Algorithms: A Critique and a New Methodology"*, Pattern Recognition. Vol. 14, pp. 53-63, 1981.

[D'Orta,87] P.D'Orta, M.Ferreti, A.Martinelli, S.Melecrinis, S.Scarci, G. Volpi: *"A Speech Recognition System for the Italian Language"*. IEEE-ICASSP proc. pp.841-843, 1987.

[Duda,73] Duda and Hart: *"Pattern Classification and Scene Analysis"*, Wiley, 1973.

[Eilemberg,74] S.Eilemberg: *"Automata, Languages and Machines"*. Vol.A, Academic Press, New York, 1974.

[Falaschi,90] A.Falashi: *"Some experiments on HMM Structure Inference"*. EUSIPCO-90, (Barcelona) proc.: "Signal Processing V: Theories and Applications". L.Torres & al. eds., pp.1375-1378, Elsevier-1990.

[Falaschi,90] A.Falaschi: *"Phonetic Structure Inference of Phonemic HMM"*. NATO-ASI "Speech Recognition and Understanding: Recent Avances, Trends and Applications", July 1-13, Cetraro, Italy, 1990.

[Falaschi,90a] A.Falaschi: *"Continously Variable Transition Probability HMM for Speech Recognition"*. NATO-ASI "Speech Recognition and Understanding: Recent Avances, Trends and Applications", July 1-13, Cetraro, Italy, 1990.

[Forney,73] G.D.Forney: *"The Viterbi Algorithm"*. pp.268-278, Proc. IEEE, vol.61,1973.

[Freeman,74] H. Freeman: *"Computer processing of line drawing images"*, Computer Surveys, Vol. 6, pp. 57-98, 1974.

[Fu,74] K.S.Fu: *"Syntactic Methods in Pattern Recognition"*. Academic Press, 1974.

[Fu,75] K.S.Fu, D.L.Booth: *"Grammatical Inference: Introduction and Survey"*. Parts 1 & 2, IIEEE Trans. System, Man and Cibernetics. Vol.SMC-5, Nº 5, pp.95-11, 409-423, 1975.

[Fu,82] K.S.Fu: *"Syntactic Pattern Recognition and Applications"*. Prentice Hall, 1982.

[Fu,83] K.S.Fu: *"A Step Towards Unification on Syntactic and Statistical Pattern Recognition"*. IEEE Trans. Pattern Analisys and Machine Intelligence, Vol PAMI-5, Nº2, pp.200-205, 1983.

[Fuyisaki,90] T.Fujisaki, T.E.Chefalas, J.Kim, C.C.Tappert: *"On-Line Recognizer for Runon Handprinted Characters"*. IEEE-ICASSP proc., pp.450-454, 1990.

[Galiano,90] I.Galiano: *"Un Proyecto Duracional en Inferencia Gramatical"*. DSIC//90, Universidad Politécnica de Valencia. 1990.

[Galiano,91] I.Galiano, F.Casacuberta, E,Sanchis: *"On the Structure of Subwords Units for a Speaker-Independent Continous Speech Task"*. EUROSPEECH Proc. Genova, Sep. 1991.

[García,88] P.García: *"Explorabilidad Local e Inferencia Inductiva de Lenguajes Regulares y Aplicaciones"*. Tesis Doctoral, DSIC-Universidad Politécnica de Valencia, 1988.

[García,90] P.García, E.Vidal: *"Inference of K-Testable Languages in the Strict Sense and Application to Syntactic Pattern Recognition"*, IEEEIEEE Trans. Pattern Analisys and Machine Intelligence, Vol.PAMI-12, Nº 9, pp.920-925, September 1990.

[Gold,67] E.M.Gold: *"Language Identification in the limit"*. Information and Control, Vol.10, pp.447-474, 1967.

[Gonzalez,78] R.Gonzalez, M.Thomason: *"Syntactic Pattern Recognition. An Introduction"*. Addison-Wesley, 1978.

[Gray,84] R.M.Gray: *"Vector Quantization"*. IEEE ASSP Magazine, 4-29, Abril, 1984.

[Gregor,88] Jens Gregor: *"Inference of Markov Networks with Forced Landmarks"*. M.Sc. Thesis, Institute of Electronic Systems, University of Aalborg. June 1988.

[Harrison,78] M.A.Harrison: *"Introduction to Formal Language Theory"*, Addison-Wesley, 1978.

[Horowitz,77] E.Horowitz, S.Sanhi: *"Fundamentals of Data Structures"*. Pitman International, 1977.

[Horowitz,78] E.Horowitz, S.Sahni: *"Fundamentals of Computer Algorithms"*. Computer Science Press Inc., 1978.

[Hunt,67] E.B.Hunt: *"Artificial Intelligence"*. Academic Press, 1975.

[Juang,85] B.H.Juang, L.R.Rabiner, S.E.Levinson, M.M.Sondhi: *"Recent Developments in the Application of Hidden Markov Models to Speaker-Independent Isolated Word Recognition"*, IEEE-ICASSP proc., pp. 9-12, 1985.

[Jouvet,86] D.Jouvet, J.Monné, D.Doubois: *"A New Network-Based Speaker-Independent connected-Word Recognition System"*. IEEE-ICASSP proc., pp.1109-1112, 1986.

[Kimura,91] F.Kimura, M.Shridar: *"Handwritten Numerical Recognition Based on Multiple Algorithms"*. Pattern Recognition. Vol.24, Nº10, pp.969-963, 1991.

[Kohonen,85] T.Kohonen: *"Median Strings"*. Pattern Recognition Letters, Nº3, pp.309-313, 1985.

[Kundu,89] A.Kundu, Y.He, P.Bahl: *"Recognition of Handwritten word: Firs and Second Order Hidden Markov Model Based Approach"*. Pattern Recognition, Vol,22, Nº3, pp.283-297, 1989.

[Kurosawa,86] Y.Kurosawa, Haruo Asada: *"Attributed String Matching with Statistical Constraints for Character Recognition"*. IEEE-ICASSP proc., pp.1063-1067, 1986.

[Lam,88] L.Lam, C.Y.Suen: *"Structural Classification and Relaxation Matching of Totally Unconstrained Handwritten ZIP-Code Numbers"*. Pattern Recognition, Vol.21, Nº1, pp.19-31, 1988.

[Lee,88] K.F.Lee: *"Large Vocabulary Speaker Independent Continous Speech Recognition"*. Tech. Report CMU-CS-88-148. Carnegie-Mellon Univ. 1988.

[Levine,82] *"The Use of Tree Derivatives and a Sample Support Parameter for Inferring Tree Systems"*. IEEE Trans. Pattern Analisys and Machine Intelligence, Vol.PAMI-4, Nº1, pp.25-34, 1982.

[Levinson,83] S.E.Levinson, L.R.Rabiner, M.M.Sondhi: *"An Introduction to the application of the Theory of Probabilistic Functions of Markov Process to Automatic Speech Recognition"*. The Bell Systems Technical Journal. Vol,62, Nº4, April 1983.

[Lippmann,87] R.P.Lippmann: *"An introduction to Computing with Neural Nets"*. IEEE ASSP Magazine, April 1987.

[Loeb,87] E.P.Loeb, R.F.Lyon: *"Experiments in Isolated Digit Recognition with a Cochlear Model"*. IEEE-ICASSP proc., pp.1131-1135, 1987.

[Makhoul,75] J.Makhoul: *"Linear Prediction: A Tutorial Review"*. Proc. of the IEEE, Vol.63, pp.561-567, April 1975.

[Mantas,86] J.Mantas: *"An Overview of Character Recognition Methodologies"*. Pattern Recognition. Vol.19, Nº6, pp.425-430, 1986.

[Marco,88] M.Marco, F.Casacuberta: *"Un Nuevo Método de Inferencia Gramatical Para Gramáticas Independientes del Contexto"*. 3er Simp. de Reconocimiento de formas y análisis de imágenes SERFAI, pp.361-368, Oviedo, 1988.

[Maryanski,77] F.J.Maryanski, T.L.Booth: *"Inference of Finite-State Probabilistic Grammars"*. IEEE Trans. on Computers Vol. C-26, pp.531-536, 1977.

[Marzal,91] A.Marzal, P.Aibar, M.J.Castro, B.Mas: *"Diversas Aproximaciones al Reconocimiento de Dígitos Manuscritos"*. Research Report DSIC-II/24/91, Universidad Politécnica de Valencia (España), 1991.

[Marzal,91a] A.Marzal, E.Vidal: *"Computation of the Normalized Edit Distance and Applications"*. Presentado al IEEE Trans. Pattern Analisys and Machine Intelligence, July 1991.

[Miclet,79] L.Miclet: *"Inference de Grammaires Régulières"*. Thèse de Docteur Ingénieur, ENST Paris, 1979.

[Miclet,80] L.Miclet: *"Regular Inference with a Tail-Clustering method"*. IEEE Trans. System Man and Cybernetics, SMC Vol.10, pp.737-743, 1980.

[Miclet,86] L.Miclet: *"Structural Methods in Pattern Recognition"*. North Oxford Academic Press, 1986.

[Miclet,90] L.Miclet: *"Grammatical Inference"*. In "Syntactic and Structural Pattern Recognition and applications". H.Bunke, A.Sanfeliu (eds.). World Scientific, 1990.

[Miralles,91] F.Miralles: *"Algoritmo de Inferencia Gramatical para el Aprendizaje Automático de Modelos en RAH"*. Proyecto fin de Carrera. DSIC, Univ. Pol. de Valencia, 1991.

[Moore.83] R.K.Moore, M.J.Russell, M.J.Tomlinson. *"The Discriminative Network; a Mechanism for focusing Recognition in Whole-Word Pattern Matching"*. IEEE-ICASSP Proc. pp.1041-1044. 1983.

[Muggleton,84] S.Muggleton: *"Induction of Regular Languages from Positive Examples"*. Tech. Report, Turing Institute Research Memoranda, Glasgow, 1984.

[Nadal,90] C.Nadal, R.Legault, C.Y.Suen: *"Complementary Algorithms for the Recognition of Totally Uncostrained Handwritten Numerals"*. IEEE ICPR proc. pp.443-449, 1990.

[Ney,91] H.Ney, R.Billi: *"Prototipe Systems for Large-Vocabulary Speech Recognition: Poliglot and Spicos"*. EUROSPEECH-91, pp.193-200, 1991.

[Prieto,88] N.Prieto, H.Rulot, E.Sanchis, E.Vidal: *"Extensión Estocástica del Algoritmo ECGI y su Aplicación al Reconocimiento de Diccionarios Difíciles"*. 3er Simposium Nacional de Reconocimiento de Formas y Análisis de Imágenes. Oviedo. Septiembre 1988.

[Prieto,91] N.Prieto, E.Vidal: *"Automatic Learning of Structural Language Models"*. IEEE-ICASSP proc., pp.789-792, 1991.

[Prieto,91a] N.Prieto, E.Vidal: *"Learning Language Models Through the ECGI Method"*. EUROSPEECH-91, Vol.2, pp.395-398, 1991.

[Quinlan,86] J.R.Quinlan: *"Induction of Decision Trees"*. Machine Learning. Vol.1, pp.81-106. 1986.

[Rabiner,78] L.R.Rabiner, R.W.Schaffer: *"Digital Proccesing of Speech Signals"*. Prentice-Hall, 1978.

[Rabiner,83] L.R.Rabiner, S.E.Levinson, M.H.Sondhi: *"On the Application of Vector Quantization and Hidden Markov Models to Speaker-Independent, Isolated Word Recognition"*. The Bell System Technical Journal. Vol.62, Nº.4, April 1983.

[Rabiner,87] L.R.Rabiner, J.G.Wilpon: *"Some Performance Benchmarks for Isolated Word Speech Recognition Systems"*. Computer Speech and Language Vol.2, pp.343-357, 1987.

[Rabiner,89] L.R.Rabiner, C.H.Lee, B.H.Juang, J.G.Wilpon: *"HMM Clustering for Connected Word Recognition"*. IEEE-ICASSP proc. pp.405-408, 1989.

[Rabiner,89a] L.R.Rabiner: *"A Tutorial on Hidden Markov Models and Selected Applications"*. Proc. of the IEEE. Vol.77, Nº2, pp.257-286. Febrero 1989.

[Raudys,91] S.J.Raudys, A.K.Jain: *"Small Sample Effects in Statistical Pattern Recognition: Recomendations for Practitioners and Open Problems"*. IEEE Trans. Pattern Analisys and Machine Intelligence, 1991.

[Richetin,84] M.Richetin, F.Vernadat: *"Efficient Regular Grammatical Inference for Pattern Recognition"*. Pattern Recognition, Vol.17, Nº2, pp.245-250, 1984.

[Robinson,90] P.Robinson & al.: *"Character Witnesses"*. MacUser, Vol.6, Nº7, Julio 1990.

[Rulot,85] H.Rulot: *"Un Reconocedor de Palabras Aisladas Basado en la Función de Autocorrelación"*. Memoria de Licenciatura. Facultad de Físicas, Univ. de Valencia, Enero 1985.

[Rulot,87] H.Rulot, E.Vidal: *"Modelling (Sub)string-Length-Based Constraints throught a Grammatical Inference Method"*. In "Pattern Recognition: Theory and Applications". Eds: Devijver & Kittler, pp. 451-459, Springer Verlag, 1987.

[Rulot,88] H.Rulot, E.Vidal: *"An Efficient Algorithm for the Inference of Circuit-Free Automata"*. In "Syntactic and Structural Pattern Recognition", Eds. G.Ferrate et al., Springer Verlag, 1988.

[Rulot,89] H.Rulot, N.Prieto, E.Vidal: *"Learning Accurate finite-state Structural Models of words through the ECGI Algorithm"*, IEEE-ICASSP proc., pp. 643-646, Vol.1, 1989.

[Rusell,85] M.J.Rusell, R.K.Moore: *"Explicit Modelling of State Occupancy in Hidden Markov Models for Automatic Speech Recognition"*. IEEE-ICASSP proc., pp.5-8, 1985.

[Saloma,87] A.Saloma: *"Formal Languages"*. Computer Science Classics, Academic Press, 1987.

[Sanchis,90] E.Sanchis, F.Casacuberta: *"Learning Structural Models of Sublexical Units"*. In "Speech Recognition and Understanding: Recent Advances, Trends and Applications". Ed. P.Laface, Springer-Verlag. NATO-ASI Series, 1990.

[Sanchis,91] E.Sanchis, F.Casacuberta, I.Galiano, E.Segarra: *"Learning Structural Models of Subword units through Grammatical Inference Techniques"*. IEEE-ICASSP proc., pp.189-192, 1991.

[Sankoff,83] D.Sankoff: *"Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison"*. Addison-Wesley, 1983.

[Sekita,88] I.Sekita, K.Toraichi, R.Mori & al.: *"Feature Extraction of Handwritten Japanese Characters by Spline Fuctions for Relaxation Matching"*. Pattern Recognition, Vol.21, Nº1, pp.9.17, 1988.

[Shridhar,86] M.Shridhar, A.Badrelin: *"Recognition of Isolated and Simply Connected Handwritten Numerals"*. Pattern Recognition Vol.19, Nº 1, pp.1-12, 1986.

[Tanaka,86] E.Tanaka, T.Toyama, S.Kawai: *"High Speed Error Correction of Phoneme Sequences"*. Pattern Recognition. Vol.19, Nº5, pp.407-412, 1986.

[Tanaka,87] E.Tanaka: *"A High Speed String Correction Method Using a Hierarchical File"*. IEEE Trans. Pattern Analisys and Machine Intelligence, Vol.PAMI-9, Nº6, Nov. 1987.

[Tappert,90] C.C.Tappert, C.Y.Suen, T.Wakahara: *"The State of the Art in On-Line Handwritting Recognition"*. IEEE Trans. Pattern Analisys and Machine Intelligence, Vol.PAMI-12, Nº8, pp.787-809, 1990.

[Tarazona,91] J.Tarazona: *"Estimación de Parámetros de los Modelos Estructurales Aprendidos Mediante el Algoritmo ECGI"*. Proyecto fin de Carrera. DSIC-Univ. Politécnica de Valencia, 1991.

[Teague,80] M.R. Teague: *"Image analysis via the general theory of moments"*, J. Optical Society of America, Vol. 70, pp. 920-930, Aug. 1980.

[The,88] C.H. The and R.T. Chin: *"On image analysis by the methods of moments"*, IEEE Trans. Pattern Analisys and Machine Intelligence, Vol. PAMI-10, pp. 496-513, July 1988.

[Thomason,74] M.G.Thomason: *"Errors in Regular Languages"*. IEEE Trans. on Computers, Vol.C-23, Nº 6, pp.597-602, June 1974.

[Thomason,75] M.G.Thomason: *"Stochastic Syntax-Directed Translation Schemata for Correction of Errors in Context-Free Languages"*. IEEE Trans. on Computers, Vol.C-24, Nº12, December 1975.

[Thomason,86] M.Thomason, E.Granum, R.E.Blake: *"Experiments in Dynamic Programming Inference of Markov Networks with Strings Representing Speech Data"*. Pattern Recognition, pp.343-351, Vol.19, Nº 5, 1986.

[Thomason,87] M.G.Thomason, E.Granum: *"Sequential Inference of Markov Networks by Dynamic Programming for Structural Pattern Recognition"*. Pattern Recognition Letters Nº5, pp.31-39, 1987.

[Torró,89] F.Torró: *"Estudio de alternativas en la reducción de Complejidad del Algoritmo de Reconocimiento Basado en el método ECGI"*. Proyecto fin de Carrera. DSIC, Univ. Pol. de Valencia, 1989.

[Torró,90] F.Torró, E.Vidal, H.Rulot: *"Fast and Accurate Speaker Independent Speech Recognition Using Structurals Models Learnt by the ECGI"*. in "Signal Proccesing V: Theories and Applications". L.Torres, E.Masgrav, M.A.Lagunas (Eds.). Elsevier Science Pub., 1990.

[Valiant,84] L.G.Valiant: *"A Theory of the Learnable"*. Sixteenth Abbual ACM Symp. on Theory of Computing Proc., ACM, 1984.

[Venta,86] O.Venta and T.Kohonen: *"A Non-Stochastic Method for the Correction of Sentences"*. ICPR proc., pp.1214-1217, 1986.

[Vidal,85] E.Vidal, F.Csacuberta, E.Sanchís, J.Benedí: *"A general fuzzy-parsing scheme for speech recognition"*. in NATO-ASI "New systems and architectures for Automatic Speech Recognition and Synthesis", R.De Mori, C.Y.Suen (eds.), Springer-Verlag, pp.427-446, 1985.

[Vidal,88] E.Vidal, N.Prieto, E.Sanchis, H.Rulot: *"Application of the Error Correcting Grammatical Inference Method (ECGI) to multi-speaker IWR"*. In "Recent Advances in ASR and Dialog Systems", Ed. H.Niemman, Springer Verlag, 1988.

[Vidal,88a] E.Vidal, H.Rulot, F.Casacuberta, J.M.Benedí: *"On the use of a Metric-Space Search Algorithm (AESA) for fast DTW-based Recognition of Isolated Words"*. IEEE -ICASSP proc. Vol.36, Nº5, 1988.

[Vidal,88-3] E.Vidal, M.J.Lloret: *"Fast Speaker-Independent DTW Recognition of Isolated Words Using a Metric Space Search Algorithm (AESA)"*. Speech Communication Vol.7, Nº4, 1988.

[Vidal,90] E.Vidal, A.Marzal: *"A Review and New Approaches for Automatic Segmentation of Speech Signals"*. in "Signal Processing V: Theories and Applications". Eds. L.Torres, E.Masgrau, M.A.Lagunas. Elsevier, 1990.

[Vidal,91] E.Vidal, H.Rulot, J.M.Valiente, G.Andreu: *"Recognition of Planar Shapes Through the Error-Correcting Grammatical Inference Algorithm (ECGI)"*. Report DSIC-II/32/91 DSIC-Universidad Politécnica de Valencia, 1991.

[Vidal,92] E.Vidal, H.Rulot, J.M.Valiente, G.Andreu: *"Font-Independent Mixed-Size Digit Recognition Through the Error-Correcting Grammatical Inference Algorithm (ECGI)"*. Presentado al congreso ICPR-92, aceptado, pendiente de publicación.

[Vidal,92a] E.Vidal, H.Rulot, J.M.Valiente, G.Andreu: *"Application of the Error-Correcting Grammatical Inference Algorithm (ECGI) to Planar Shape Recognition"*. Presentado a la revista Pattern Analysis and Machine Recognition (PAMI), pendiente de aceptación.

[Wagner,74] R.Wagner & M.J. Fisher: *"The String-to-String Correction Problem"*. Journal of the ACM,21,p.p.168-173 (1974).

[Watanabe,88] T.Watanabe: *"Speaker-Independent Word Recognition Using Dynamic Programming Matching With Statistic Time Waping Cost"*. IEEE-ICASSP proc., pp.199-202, 1988.

[Wetherell,90] C.S.Wetherell: *"Probabilistic Languages: A Review an Some Open Questions"*. ACM Computing Surveys, Vol.12, Nº4, pp.361-379, Dec.1990.

[Wollmann,70] R.Wollmann: *"Trabajos Manuales para jóvenes"*. Editorial Labor S.A., 1970.

[Zhang,84] T.Y. Zhang and C.Y. Suen: *"A fast parallel algorithm for thinning digital pattern"*, Comm. ACM 27 (3), pp. 236-239, 1984.

UNIVERSITAT DE VALENCIA FACULTAD DE CIÈNCIES FÍSIQUES

Reunit el Tribunal que subscriu, en el dia de la data, acordà d'atorgar, per unanimitat, a aquesta Tesi Doctoral d'En/ Na/ N' Hector Rubio Segovia la qualificació d Apto Cum Laude

València a 26 d junio de 1992

El Secretari,

El President,

José B. Marquina

UNIVERSITAT DE VALÈNCIA
BURJASSOT
FACULTAT DE CIÈNCIES FÍSIQUES