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 $G=(V,A)$; $A=\{(u,v): u\in V, v\in V\}\subseteq V^2$.

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∈V como:

$$N_c(p,u) = \begin{cases} 1 & \text{si } (p,u)\in A \\ \sum_{\forall x\in V:(x,u)\in A} N_c(p,x) & \text{si } (p,u)\notin A \end{cases}$$

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.

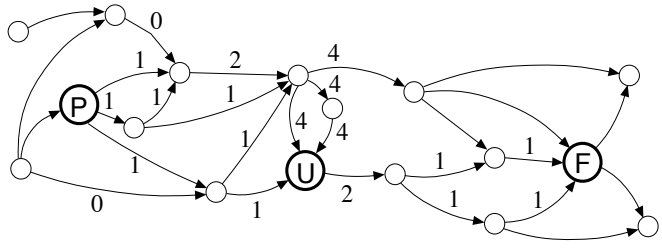
El **número de caminos inversos** en G, que llegan a p desde u (que van de u a p), se obtiene simplemente invirtiendo el sentido de las aristas y efectuando el cálculo al revés:

$$N_{ci}(u,p) = \begin{cases} 1 & \text{si } (p,u)\in A \\ \sum_{\forall x\in V:(p,x)\in A} N_{ci}(u,x) & \text{si } (p,u)\notin A \end{cases}$$

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∈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)∈A, viniendo del vértice p∈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∈A que viene de los vértices p y va a f; p,f∈V (número total de caminos que van de p a f y pasan por t) se obtiene como:

$$T_t(p,t,f) = N_{ct}(p,t)\cdot N_{cti}(t,f)$$

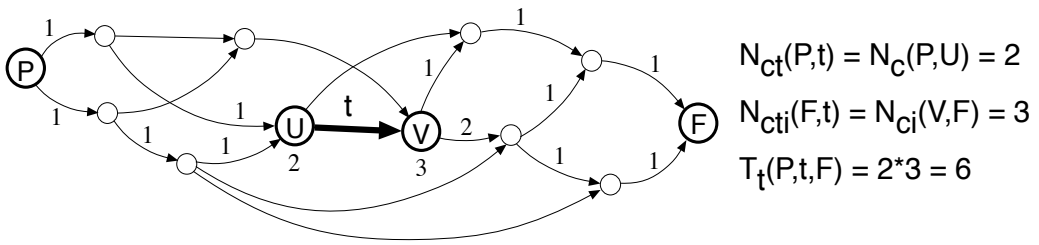


Figura 10.2 Tráfico por una arista t=(u,v).

Si $t_i=(u,x_i)$∈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:

$$C_l(p,f) = \begin{cases} 1 & \text{si } (p,f)\in A \\ 1 + \max\limits_{\forall x\in V:(x,f)\in A} C_l(p,x) & \text{si } (p,f)\notin A \end{cases}$$

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] $p,f \in V$ 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:

$$\sum_{i=1}^{m} P_E(t_i) = 1$$

[1] Toda la discusión que sigue considera las propiedades del grafo únicamente en relación con los caminos que van de p a f.

es decir, estas probabilidades son consistentes.

Proposición: Si $\mathbf{C}_{pf}$ es el conjunto de caminos de p a f en G, definimos la **probabilidad estructural de un camino** $C \in \mathbf{C}_{pf}$, formado por la secuencia de k aristas $t_1,t_2..t_k$; como:

$$P_E(C)=\prod_{j=1}^{k} P_E(t_j)\ ; \qquad \text{entonces} \qquad \sum_{\forall C\in \mathbf{C}_{pf}} P_E(C) = 1$$

En efecto, si el camino pasa por los nodos $p,u_1,u_2,\ldots,u_{k-1},f \in \mathbf{V}$, su probalidad estructural será :

$$P_E(C)= \frac{T_t(p,t_1,f)}{T(p,p,f)} \cdot \frac{T_t(p,t_2,f)}{T(p,u_1,f)} \cdot \ldots \cdot \frac{T_t(p,t_k,f)}{T(p,u_{k-1},f)} =$$

$$\frac{N_c(p,p)\cdot N_{ci}(u_1,f)}{T(p,p,f)} \cdot \frac{N_c(p,u_1)\cdot N_{ci}(u_2,f)}{T(p,u_1,f)} \cdot \ldots \cdot \frac{N_c(p,u_{k-1})\cdot N_{ci}(f,f)}{T(p,u_{k-1},f)} =$$

$$\frac{1\cdot T(p,u_1,f)\cdot \ldots \cdot T(p,u_{k-1},f)\cdot 1}{T(p,p,f)\cdot T(p,u_1,f)\cdot \ldots \cdot T(p,u_{k-1},f)} = \frac{1}{N_c(p,f)} \quad \textbf{c.q.d.}$$

Es decir, la probabilidad estructural de cualquier camino de p a f es:

$$P_E(C) = \frac{1}{N_c(p,f)}\ ; \quad \forall C\in \mathbf{C}_{pf}$$

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:

$$P_E(t)=\frac{I_E(t)}{I_E(u)} \qquad \text{si } \exists\, x\in V : t=(u,x)$$

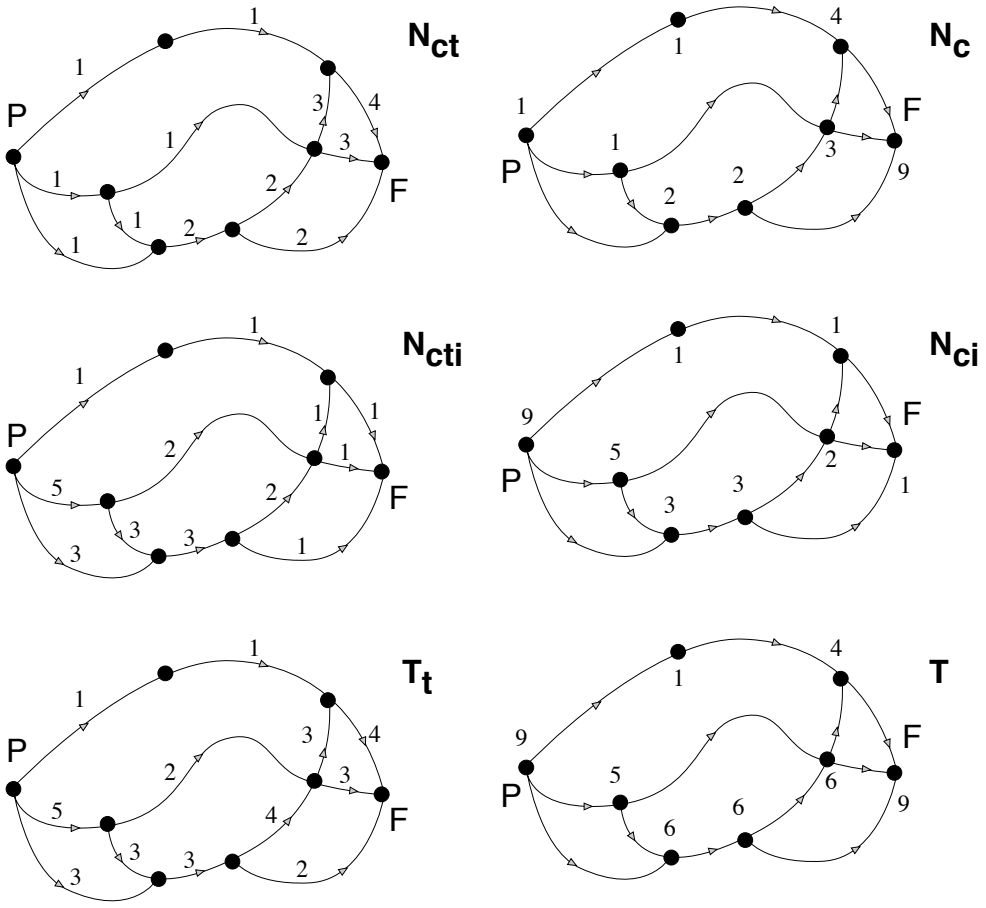


Figura 10.3 De arriba abajo: número de caminos, número de caminos inversos y tráfico por arista (a la izquierda) y vértice (a la derecha) en un grafo con vértices principio (p) y final (f). La importancia estructural por arista y vértice son proporcionales a los tráficos.

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.

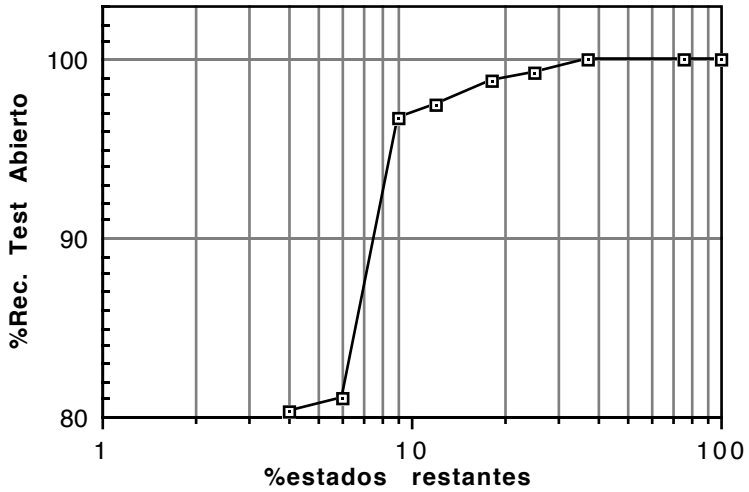


Figura 10.4 Tasa de reconocimiento en función del porcentaje estados restantes en un experimento de reconocimiento de dígitos hablados (H6). Las abcisas de los punto representados son 72%, 46%, 42%, 31%, 29%, 22%, 21%, 16% y 15%.

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.

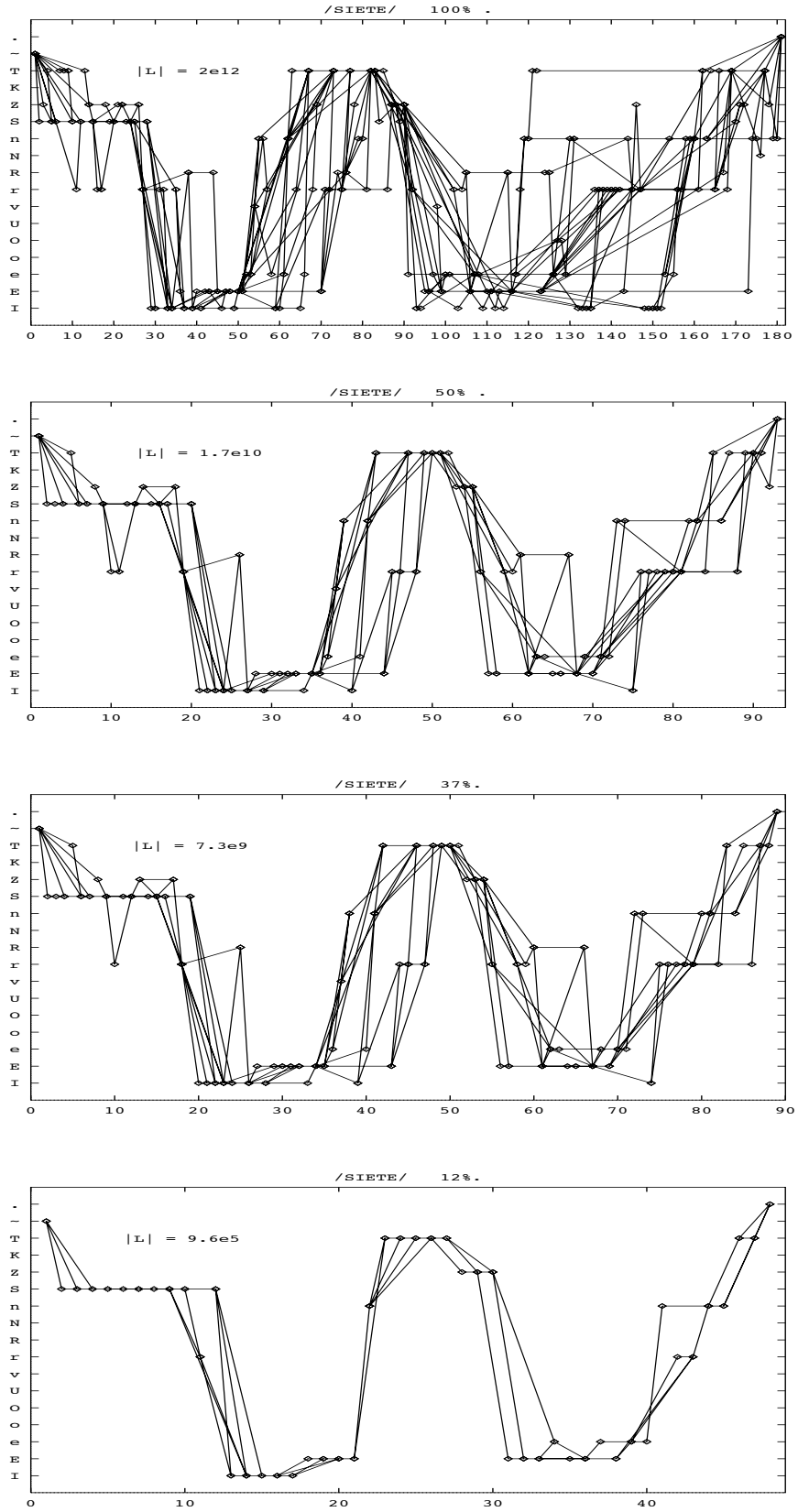


Figura 10.5 Diversas etapas de la simplificación de un autómata inferido por ECGI. El autómata con 37% de estados aún proporciona un 100% de aciertos en reconocimiento. El de 12% aún un 80%.

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.