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, V_i y V_{i+1} y los vértices afectados por la regla en cuestión).

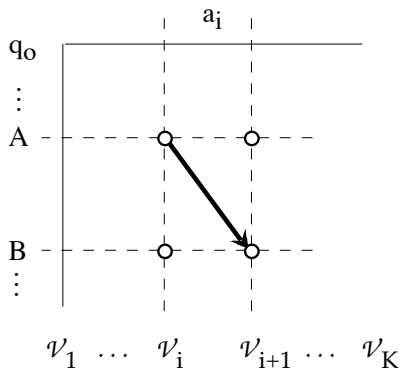


Figura 5.1 Subregión de trellis correspondiente al análisis sintáctico de la regla A→aB al presentarse el símbolo i-ésimo a_i (etapas V_i y V_{i+1}).

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;

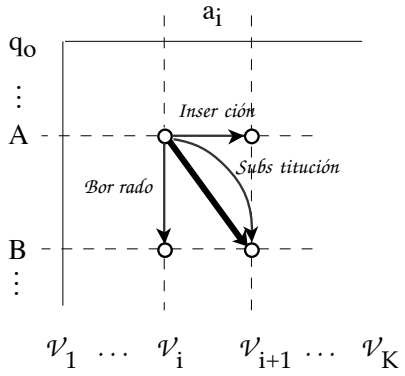


Figura 5.2 Subregión del correspondiente al análisis sintáctico de la regla A→aB al presentarse el símbolo i-ésimo a_i (etapas V_i y V_{i+1}), y aristas (trazo punteado) asociadas a los errores de borrado, inserción y substitución.

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:

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

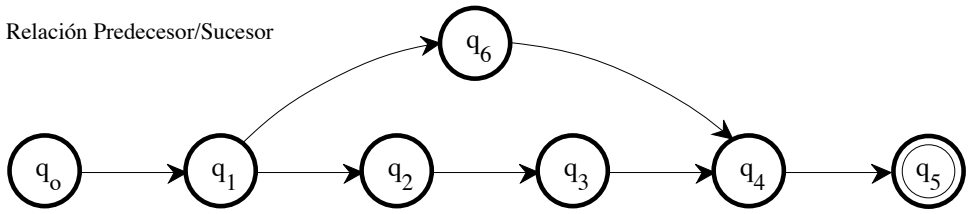
Veremos a continuación, que si una gramática cumple esta propiedad, es posible utilizar el algoritmo de `Viterbi` para llebar a cabo el análisis sintáctico corrector de errores. Veremos también que en el caso más general de una gramática regular *con* circuitos, el algoritmo **no** es aplicable.

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

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

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 **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_o	q_1	q_2	q_6	q_3	q_4	q_5
q_o	q_1	q_6	q_2	q_3	q_4	q_5
q_o	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", $A=(V,Q,q_o,F,\delta)$ */
$V:C_\sigma$ $Q:C_\tau$ $q_o:\tau$ $F:C_\tau$ $\alpha=a_1a_2\ldots a_n:L_\sigma$
Resultado $R:R$
Auxiliar /* ψ es el tipo "error" */
$p:\tau\times\sigma\times\tau\rightarrow R$ /* probabilidad de una transición */
$p_e:\tau\times\psi\times\tau\rightarrow R$ /* probabilidad de error en una transición */
sig:$\tau\rightarrow\tau$ /* ordenación de estados*/
Variables $P,P':C_{\tau\times R}$ /* Vectores, indexados por τ */
$q,q':\tau$

```
Método
  ∀q'∈Q hacer P'[q']:=0 fin∀
  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_o.

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

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 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

---

[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.

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=(V,A), con circuitos, se puede recurrir al siguiente algoritmo de PD:

```
Algoritmo MINCIRCUITOS
Datos    /* τ es el tipo "vértice", G=(V,A) */
         Co:C_τxR        /* Vector de costos iniciales,
indexado por τ */
Variables C:C_τxR        /* Vector, indexado por τ */
          HayModif:B     /* Hay estados modificados */
          v,v':τ
          antes:R
Método
  C:=Co
  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 $|V|-1$ y que, por lo tanto, la complejidad máxima es $(|V|-1)|V|$. 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.

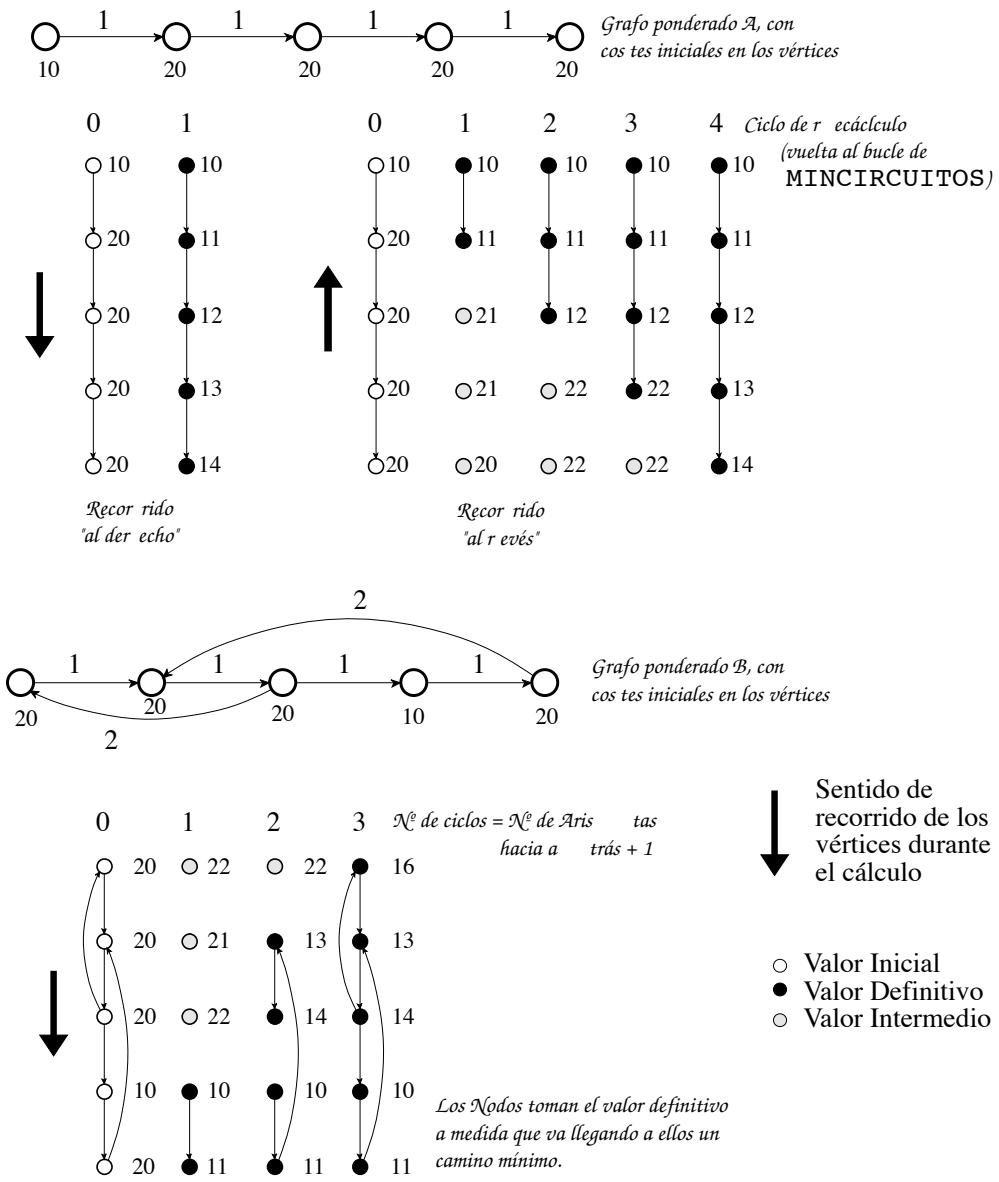


Figura 5.4 Ejemplos de recorrido por `MINCIRCUITOS` de los vértices de dos grafos, el primero según dos órdenes de evaluación distintos. Se muestran los grafos, con los costes iniciales en cada vértice y los pesos por arista. Debajo, se muestran los costes acumulados por vértice en cada ciclo del bucle 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,qo,F,δ) */
Datos   /* τ es el tipo "estado", σ el "símbolo", ψ es el
tipo "error */
        V:Cσ  Q:Cτ    qo:τ    F:Cτ  α=a1a2…an:Lσ
resultado R:R
Auxiliar   p:τxσxτ→R /* probabilidad de una transición */
           pe:τxψxτ→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 fin∀;      P'[q_o]:=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=a_j */
               max     {P'[q']·p(q',a_j,q)},
           ∀q'∈δ^-1(q,a_j)
           /* reglas de sustitución: q->α_j q', a≠a_j */
                    max          {P'[q']·p(q',s_a|a_j,q)}
           ∀q'|q'∈δ^-1(q,a),a∈V, a≠a_j
           /* regla de inserción: q->a_j q */
           {P'[q]·p(q,i_a_j,q)},
           /* reglas de borrado: q->q' */
                     max            {P[q']·p(q',b_a,q)}
           ∀q'|q'∈δ^-1(q,a),a∈V,q'∈Modif
              }
           Modif:=Modif+{q}
     fin∀
     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',b_a,q)}  }
              ∀q'|q'∈δ^-1(q,a),a∈V
           si P[q]≠antes entonces HayModif:=verdad
        fin∀
     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].

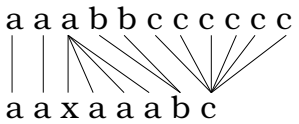


Figura 5.5 Traza de edición entre la cadena "aaabbcccccc" y la cadena "aaxaaabc". Los trazos continuos representan reglas de no error, los discontinuos reglas de error.

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].

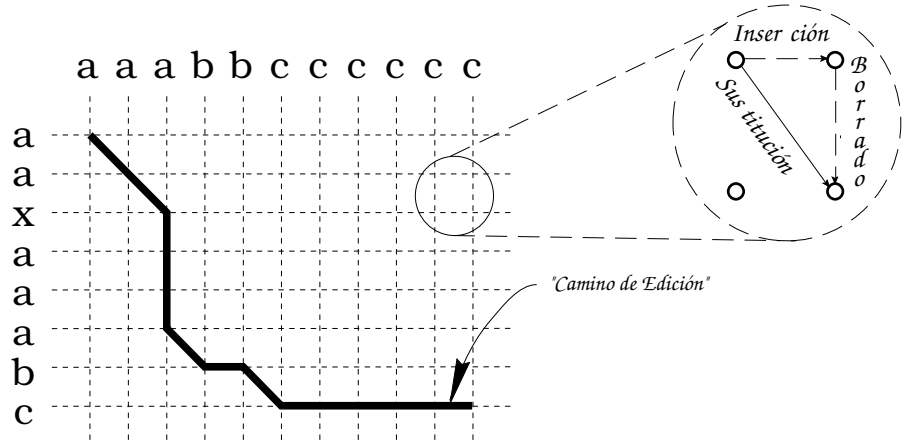


Figura 5.6 Camino de edición (trazo grueso) entre las mismas cadenas de la figura anterior ("aaabbcccccc" y "aaxaaabc"). Según un tramo del camino sea diagonal, vertical u horizontal representa un error de substitución, de inserción o borrado respectivamente. Si los símbolos a los lados de la rejilla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

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).

---

[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.