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:C_ρ→ρ
          descompone:δ→C_{δ×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». $\otimes$ 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.

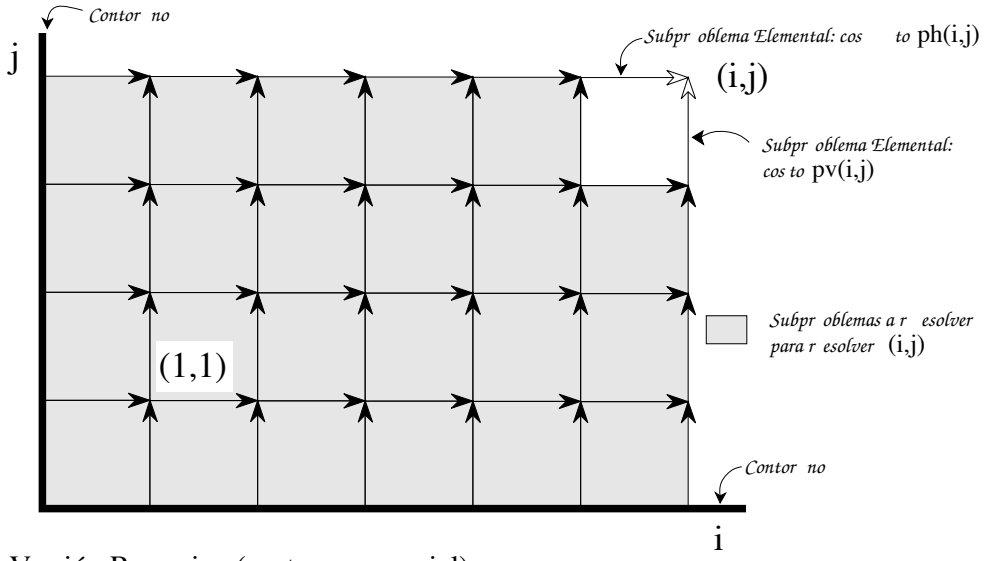
Con el fin de reducir la complejidad de exponencial a *polinómica*, evitando recalcular subproblemas ya calculados, se transforma este esquema recursivo en uno *iterativo*:

```
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 $\otimes$ 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 $O(k \cdot m)$, 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)

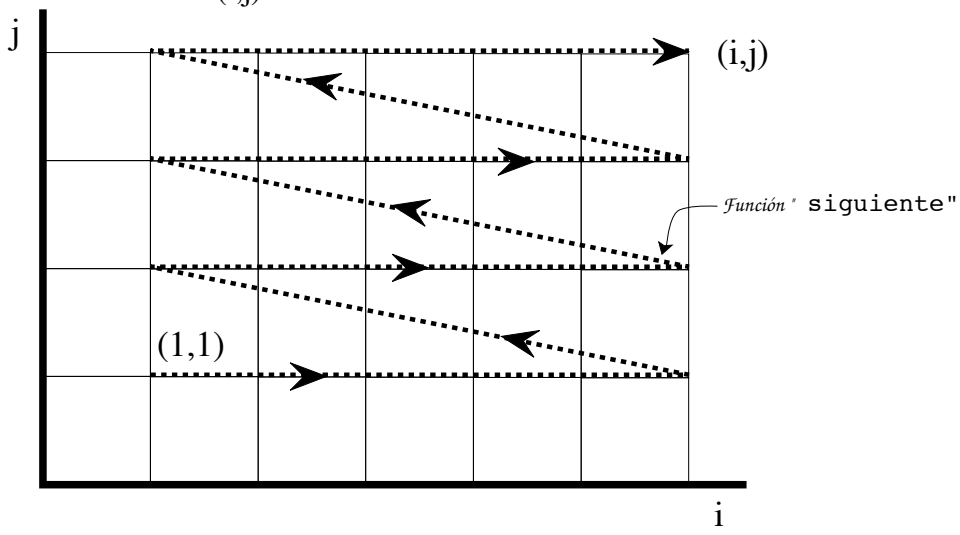


Figura 4.1 Versión Recursiva e Iterativa de un problema simple de PD (camino de coste mínimo en la Cuadrícula entre (1,1) e (i,j) en un grafo dirigido y ponderado) [Torró,89].

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

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=(V,A) como un conjunto V de*vértices* (también llamados *nodos*) y un conjunto A={(u,v): u∈V, v∈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*:

$$V=\bigcup_{i=1..K} V_i \qquad V_i \cap V_j=\varnothing; \quad \forall i,j=1,\ldots,K; \quad i\neq j; \quad k\geq 2$$

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

$$\forall (u,v)\in A \quad \exists i,\ 1\leq i\leq K-1 :\ u\in V_i \ \wedge\ v\in V_{i+1}$$

- Existe un conjunto de vértices iniciales $I\subseteq V_1$ y un conjunto de vértices finales $F\subseteq V_K$.

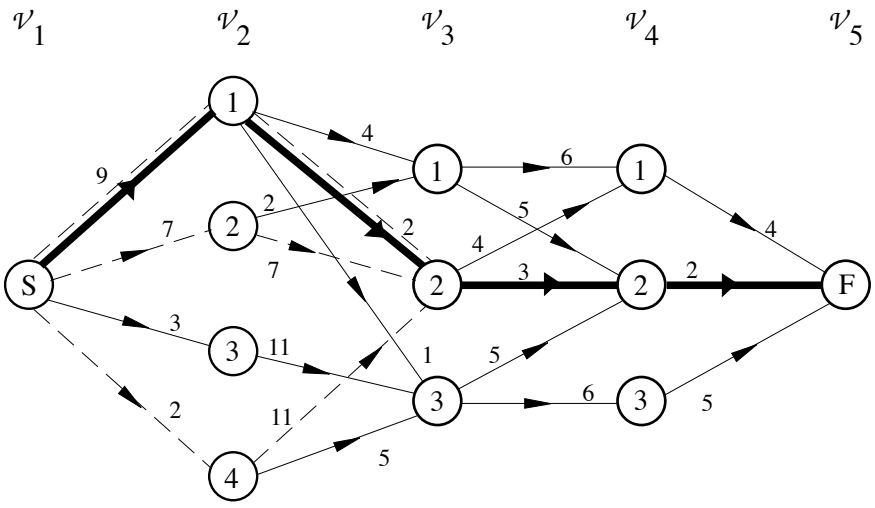


Figura 4.2 Un grafo multietapa ponderado de 5 etapas, un estado inicial y uno final. El trazo grueso indica el camino de mínimo coste [Horowitz,78]. Los arcos en trazo punteado representan el subgrafo del nodo (3,2).

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 V_j. Se define $U_{j,w}$ como el conjunto de todos los vértices de una etapa V_{j-1} conectados al vértice w de la etapa V_j:

$$U_{j,w} = \{ u \in V_{j-1} : (u,w) \in A \} \quad 1<j\leq K; \quad w \in V_j$$

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 $O(|V| \cdot M)$, siendo $O(|V|)$ 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 $A=(V,Q,\delta,q_o,F)$ y la cadena $\alpha=a_1,a_2,...,a_n$, el trellis asociado $T(V,A)$ 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:

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

- 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:

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

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

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

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

- 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:

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

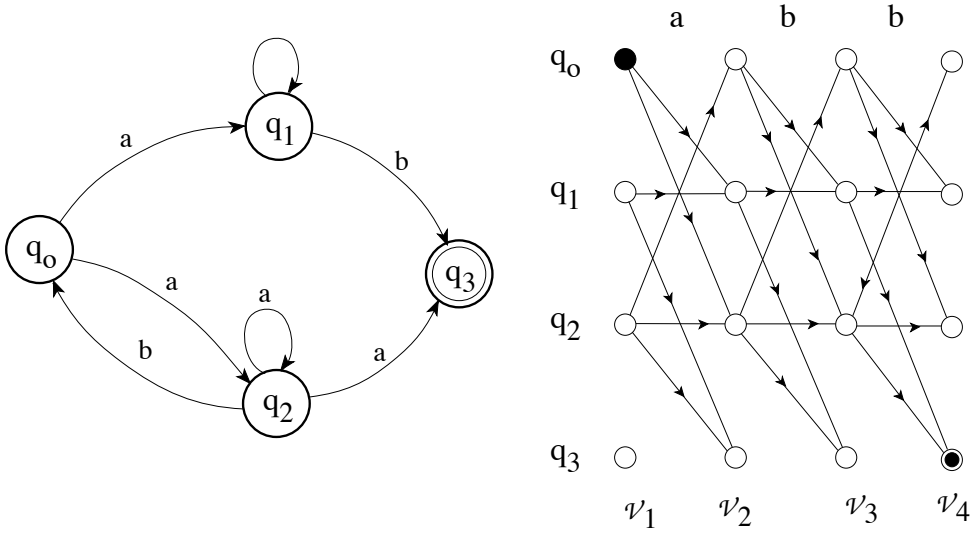


Figura 4.3 Un autómata finito y el trellis correspondiente para la cadena "abb" [Torró,89].

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:

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

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

$$C(j,q) = \underset{\forall u\in U_{j,q}}{\texttt{max}} (C(j-1,u)\cdot\rho((j-1,u), (j,q)))$$

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,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  p:τ×σ×τ→R /* probabilidad de una transición */
Variables P,P':C_τ×R  /* Vectores, indexados por t */
          q,q':τ
Método
  ∀q'∈Q hacer P'[q']:=0 fin∀
  P'[q_o]:=1;
  para j:=1..|α| hacer
     ∀q∈Q hacer P[q]:=    max     {P'[q']·p(q',a_j,q)}
                       ∀q'∈δ^-1(q,a_j)
     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].