6 L'algorithme de Earley

Earley (1970), dans Grosz et al. (1986).

Stratégie mixte : on commence top-down et au moment d'arriver au niveau de la phrase, on complète les sous-arbres d'une façon bottom-up.

Différences principales entre CYK et Earley :

- Dans l'algorithme de Earley on met des arcs passifs et des arcs actifs dans le chart.
- L'algorithme de Earley ne pose pas de contraintes pour les grammaires : l'algorithme de Earley ne demande pas une grammaire sous forme normale de Chomsky et les grammaires peuvent même avoir de récursions de gauche ou de droite, des productions de la forme $A \to \epsilon$ et des cycles.
- Earley adopte une stratégie mixte. Cela évite un des problèmes typiques d'une stratégie bottom-up comme CYK, notamment de générer des sous-arbres dont on n'a pas besoin plus tard.

Pour représenter des arcs on utilise des productions divisées : $A \to \alpha.\beta$. Une telle production indique que α et la partie déjà reconnue et β la partie encore à reconnaître.

Définition 6 $G = \langle V, T, P, S\rangle$ *soit une grammaire hors contexte.*

$P_D := \{A \to \alpha.\beta \mid A \to \alpha\beta \in P, \alpha, \beta \in (V \cup T)^*\}$ *est l'ensemble des productions divisées de* G.

Le cas $\beta = \epsilon$ correspond à un arc passif, le cas $\beta \neq \epsilon$ à un arc actif. α s'appelle la *partie fermée*, β la *partie ouverte* de l'arc en question.

Une production divisée $A \to \alpha.\beta$ est dans $chart(i, j)$ ssi

(a) $S \overset{*}{\Rightarrow} w_1 \dots w_i A\gamma$ pour quelque $\gamma \in (V \cup T)^*$, et

(b) $\alpha \overset{*}{\Rightarrow} w_{i+1} \dots w_j$

La première condition est parfois appelée *condition top-down*, la deuxième *condition bottom-up*.

Idée de l'algorithme : On commence avec les productions $S \to \beta$ avec partie gauche S. Pour chacune d'elles on met $S \to .\beta$ dans $chart(0, 0)$. Ensuite on applique trois procédures différentes, `expand`, `complete` et `shift` :

- `expand` fait des expansions top-down,
- `shift` reconnaît des mots,
- et `complete` fait des réductions bottom-up.

Exemple 23

Considérez une grammaire hors contexte $G = \langle V, T, S, P\rangle$ *avec*

- $V = \{$*NP, AP, N, Det, A*$\}$
- $T = \{$*le, petit, chien*$\}$
- $S =$*NP*
- *Productions P :*
  NP $\to$ *det n* *NP* $\to$ *det A n*
  Det $\to$ *le* *N* $\to$ *chien* *A* $\to$ *petit*

le petit chien

nouvelles entrées dans le chart	*procédure*
$NP \to .\ Det\ N \in chart(0,0)$	
$NP \to .\ Det\ A\ N \in chart(0,0)$	
$Det \to .\ le \in chart(0,0)$	*expand*
$Det \to le\ . \in chart(0,1)$	*shift*
$NP \to Det\ .\ N \in chart(0,1)$	*complete*
$NP \to Det\ .\ A\ N \in chart(0,1)$	*complete*
$N \to .\ chien \in chart(1,1)$	*expand*
$A \to .\ petit \in chart(1,1)$	*expand*
$A \to petit\ . \in chart(1,2)$	*shift*
$NP \to Det\ A\ .\ N \in chart(0,2)$	*complete*
$N \to .\ chien \in chart(2,2)$	*expand*
$N \to chien\ . \in chart(2,3)$	*shift*
$NP \to Det\ A\ N\ . \in chart(0,3)$	*complete*

La procédure `expand` ajoute des nouvels arcs cycliques (position intitiale = position terminale) au chart : Si $A \to \alpha.B\beta \in chart_{i,j}$, elle fait une expansion pour B en ajoutant $B \to \alpha.\gamma \in chart_{j,j}$ pour chaque production avec partie gauche B.

Procédure expand

Données globales : CFG $G = \langle V, P, S \rangle$ et chart $chart$, longueur n d'une phrase
Input : -
Output : true/false
Effet supplémentaire : modification de $chart$
Variable locale : $changed$, valeur initiale false
Méthode :
  for all i, j, $0 \leq i \leq j \leq n$
    for all $A \to \alpha.B\beta$ from $chart_{i,j}$:
      for all $B \to \gamma$ from P :
        if $B \to .\gamma \notin chart_{j,j}$
        then
          $chart_{j,j} := chart_{j,j} \cup \{B \to .\gamma\}$
          $changed :=$ true
  return($changed$)

La procédure `complete` fait une réduction bottom-up au moment d'avoir trouvé un arc passif de j à k pour une catégorie B qui est la première catégorie de la partie ouverte d'un arc actif $A \to \alpha.B\beta$ de i à j :

Procédure complete

Données globales : chart $chart$, longueur n d'une phrase
Input : -
Output : true/false
Effet supplémentaire : modification de $chart$
Variable locale : $changed$, valeur initiale false
Méthode :
  for all i, j, k, $0 \leq i \leq j \leq k \leq n$
    for all $A \to \alpha.B\beta$ from $chart_{i,j}$:
      for all $B \to \gamma.$ from $chart_{j,k}$:
        if $A \to \alpha B.\beta \notin chart_{i,k}$
        then
          $chart_{i,k} := chart_{i,k} \cup \{A \to \alpha B.\beta\}$
          $changed :=$ true

```
   return(changed)
```

La procédure `shift` fait un pas à droite si le premier élément de la partie ouverte d'un arc avec position terminale $j-1$ est w_j :

Procédure shift

Données globales : chart $chart$, phrase $w = w_1 \ldots w_n$
Input : i, j
Output : -
Effet supplémentaire : modification de $chart$
Méthode :
   for all $A \to \alpha.t\beta$ from $chart_{i,j-1}$ avec $t = w_j$:
      $chart_{i,j} := chart_{i,j} \cup \{A \to \alpha t.\beta\}$

La version de l'algorithme de Earley qui suit suppose une grammaire sans des règles $A \to \epsilon$ et sans dérivations $A \overset{n}{\Rightarrow} A, n > 0$.

Algorithme recognize-Earley-1

Données globales : CFG $G = \langle V, T, P, S\rangle$
Input : phrase $w = w_1 \ldots w_n$
Output : true/false
Variable locale : $chart$: matrice $n+1 \times n+1$. Valeur initiale : $chart_{i,j} = \emptyset$ pour $0 \leq i \leq j \leq n+1$.
$changed$, valeur initiale true
Méthode :
   for all $S \to \gamma$ from P:
      $chart_{0,0} := chart_{0,0} \cup \{S \to .\gamma\}$
   while $changed$
      $changed :=$ expand()
   for all j from $\{1, \ldots n\}$:
      for all i from $\{0, \ldots, j\}$:
         shift(i, j)
         $changed := true$
         while $changed$
            $changed :=$ expand() $\vee$ complete()
   if il existe un γ tel que $S \to \gamma. \in chart_{0,n}$
   then return(true)
   else return(false)

Désavantage : relations entre addition d'un nouvel arc et procédures `expand`, `complete` et `shift` ne sont pas suffisamment prises en compte. Idée : enchaînement des arcs. Si on ajoute un arc passif, cet arc peut tout-de-suite être utilisé par `complete`, si on ajoute un arc actif, on peut tout-de-suite appliquer `expand` pour ajouter de nouvels arcs actifs. `expand` et `complete` vont être appelé avec un arc comme argument.

$\Rightarrow$ procédure `closure` appelé pour ajouter un nouvel arc :

Procédure closure

Données globales : chart $chart$
Input : i, j, production divisée $A \to \alpha.\beta$
Output : -
Effet supplémentaire : modification de $chart$
Méthode :
   if $A \to \alpha.\beta \notin chart_{i,j}$

```
   then
      chart_{i,j} := chart_{i,j} ∪ {A → α.β}
      if β = ε
      then complete(i, j, A → α.β)
      else expand(i, j, A → α.β)
```

Nouvelles procédures `expand` et `complete` :

Procédure expand

```
Données globales :  CFG G = ⟨V,T,P,S⟩ et chart chart
Input :  i, j, production divisée A → α.Bβ
Output :  -
Méthode :
   for all B → γ from P :
      closure(j, j, B → .γ)
```

Procédure complete

```
Données globales :  chart chart
Input :  j, k, production divisée B → γ.
Output :  -
Méthode :
   for all i, 0 ≤ i ≤ j
      for all A → α.Bβ from chart_{i,j}:
         closure(i, k, A → αB.β)
```

Une division en dictionnaire D et productions P permet d'éviter qu'`expand` génère un grand nombre d'arcs inutiles (par exemple pour tous les noms possibles). Pour cela il faut modifier `shift` :

Procédure shift

```
Données globales :  dictionnaire D, CFG G, phrase w = w_1 ... w_n
Input :  j
Output :  -
Méthode :
   for all ⟨w_j, X⟩ from D :
      closure(j − 1, j, X → w_j.)
```

Avec ces procédures modifiées, l'algorithme principal devient :

Algorithme recognize-Earley-2

```
Données globales :  CFG G = ⟨V,T,P,S⟩, dictionnaire D
Input :  phrase w = w_1 ... w_n
Output :  true/false
Variable locale :  chart :  matrice n+1 × n+1.  Valeur initiale :  chart_{i,j} = ∅ for
all 0 ≤ i ≤ j ≤ n + 1.
Méthode :
   for all S → γ from P:
      closure(0, 0, S → .γ)
   for all i from {1, ..., n} :
      shift(i)
   if il existe un γ tel que S → γ. ∈ chart_{0,n}
   then return(true)
   else return(false)
```

Une différence entre l'algorithme 2 et l'algorithme 1 est que le `shift` de l'algorithme 2 ajoute des arcs purement bottom-up. Par conséquent, `shift` peut ajouter des arcs qui ne satisfaisont pas la condition top-down qui dit que pour chaque $A \to \alpha.\beta \in chart_{i,j}$ il y a une dérivation $S \overset{*}{\Rightarrow} w_1 \dots w_i A\gamma$ pour quelque $\gamma \in (V \cup T)^*$. Mais en général, le nombre d'arcs inutiles ajoutés par `shift` dans algorithme 2 et beaucoup plus petit que le nombre d'arcs inutiles ajoutés par l'`expand` d'algorithme 1 pour les catégories lexicales.

Dans l'algorithme 2, l'ordre dans lequel des arcs sont ajoutés est différent, c'est une stratégie depth-first (qui est une conséquence de la structure récursive de l'algorithme):

Exemple 24 *Grammaire et phrase de l'exemple précédant :*

nouvelles entrées dans le chart	*procédure*
$NP \to .\ Det\ N \in chart(0,0)$	
$Det \to .\ le \in chart(0,0)$	*expand*
$NP \to .\ Det\ A\ N \in chart(0,0)$	
$Det \to le\ . \in chart(0,1)$	*shift*

...

L'algorithme Earley-2 ne marche pas avec des grammaires avec productions $A \to \epsilon$.

Exemple 25

$G = \langle\{S,\ A,\ B\}, \{a,\ b\}, \{S \to \epsilon,\ S \to S\ A\ B\},\ S\rangle$, $D = \{\langle a,\ A\rangle, \langle b,\ B\rangle\}$.

$w = a\ b$

Seules entrées dans le chart :

$S \to .\ \in chart(0,0)$
$S \to .\ S\ A\ B \in chart(0,0)$

$S \to S\ .\ A\ B \notin chart_{0,0}$ *et la phrase n'est pas reconnue.*

Solutions qui permettent des règles $A \to \epsilon$:

1. Au moment d'ajouter $A \to \alpha.B\beta \in chart_{i,j}$ on teste si $B \to .\ \in chart_{i,i}$. Si oui, on ajoute aussi $A \to \alpha B.\beta \in chart_{i,j}$.
2. Avant d'appliquer l'algorithme, on calcule l'ensemble des catégories $X \in V$ avec $X \overset{*}{\Rightarrow} \epsilon$ (ceci est décidable). Chaque fois qu'on ajoute un arc $A \to \alpha.B\beta \in chart_{i,j}$, on teste si B se trouve dans cet ensemble. Si oui, on ajoute aussi $A \to \alpha.\beta \in chart_{i,j}$ (plus efficace que la première solution).
   Problème : la construction d'une analyse devient plus difficile parce que après avoir supprimé une catégorie X avec $X \overset{*}{\Rightarrow} \epsilon$, on ne connaît plus l'arbre de dérivation pour la partie $X \overset{*}{\Rightarrow} \epsilon$.

Intégration d'un look-ahead : Dans les deux versions de l'algorithme de Earley, `expand` génère trop d'arcs inutiles. On peut éviter cela par l'utilisation d'un Look-ahead:

$A \to \alpha.\beta \in chart_{i,j}$ ssi

(a) $S \overset{*}{\Rightarrow} w_1 \dots w_i A\gamma$ pour quelque $\gamma \in (V \cup T)^*$,
(b) $\alpha \overset{*}{\Rightarrow} w_{i+1} \dots w_j$, et
(c) $\beta \overset{*}{\Rightarrow} w_{j+1}\sigma$ pour quelque $\sigma \in (V \cup T)^*$.

Mais : Ceci fonctionne seulement si on n'a pas de productions $X \to \epsilon$. Si $\beta \overset{*}{\Rightarrow} \epsilon$, ce n'est pas nécessairement β qui mène vers w_{j+1}, la condition serait alors trop stricte.

$\Rightarrow$ on utilise une relation $FIRST$ qui spécifie l'ensemble de premiers symboles terminaux qui peuvent être dérivés à partir d'une suite de symboles β. Si $\beta \overset{*}{\Rightarrow} \epsilon$, ϵ fait aussi partie de cet ensemble.

Définition 7 (Relation $FIRST$)

$G = \langle V, T, P, S \rangle$ soit une grammaire hors contexte, $\beta \in (V \cup T)^*$.

$FIRST(\beta) := \{a \mid \beta \stackrel{*}{\Rightarrow} a\alpha, a \in T, \alpha \in (V \cup T)^*\} \cup \{\epsilon \mid \beta \stackrel{*}{\Rightarrow} \epsilon\}$.

On peut précompiler la relation $FIRST$ pour tous les β avec une production $A \to \alpha\beta \in P$. Puis, on peut modifier l'algorithme tel qu'un arc actif avec partie ouverte β est entré dans $chart_{ij}$ seulement si $FIRST(\beta) \cap \{t \mid \langle w_{j+1}, t \rangle \in D\} \neq \emptyset \vee \epsilon \in FIRST(\beta)$.

Obtenir une analyse : On peut enrichir les arcs dans le chart par une représentation implicite de la structure : Tous les arcs reçoivent des indexes. Chaque arc généré par `complete` contient, à côté de la production divisée, des pairs d'indexes dont le premier est l'index de l'arc actif et le deuxième celui de l'arc passif utilisé pour générer ce nouvel arc. On peut avoir plusieurs pairs parce qu'il peut y avoir plusieurs possibilités de générer un arc par `complete`.

Exemple 26

Productions : $AP \to AP\ AP$, $AP \to A$

Phrase w = petit joli mignon

index	*arc*	*pairs d'indexes*	*procédure*
1	$AP \to .\ AP\ AP \in chart(0,0)$		
2	$AP \to .\ A \in chart(0,0)$		*expand*
3	$A \to petit\ . \in chart(0,1)$		*shift*
4	$AP \to A\ . \in chart(0,1)$	$\langle 2,3 \rangle$	*complete*
5	$AP \to AP\ .\ AP \in chart(0,1)$	$\langle 1,4 \rangle$	*complete*
6	$AP \to .\ AP\ AP \in chart(1,1)$		*expand*
7	$AP \to .\ A \in chart(1,1)$		*expand*
8	$A \to joli\ . \in chart(1,2)$		*shift*
9	$AP \to A\ . \in chart(1,2)$	$\langle 7,8 \rangle$	*complete*
10	$AP \to AP\ AP\ . \in chart(0,2)$	$\langle 5,9 \rangle$	*complete*
11	$AP \to AP\ .\ AP \in chart(1,2)$	$\langle 6,9 \rangle$	*complete*
12	$AP \to AP\ .\ AP \in chart(0,2)$	$\langle 1,10 \rangle$	*complete*
13	$AP \to .\ A \in chart(2,2)$		*expand*
14	$A \to mignon\ . \in chart(2,3)$		*shift*
15	$AP \to A\ . \in chart(2,3)$	$\langle 13,14 \rangle$	*complete*
16	$AP \to AP\ AP\ . \in chart(1,3)$	$\langle 11,15 \rangle$	*complete*
17	$AP \to AP\ AP\ . \in chart(0,3)$	$\langle 5,16 \rangle, \langle 12,15 \rangle$	*complete*

A partir d'un chart avec des telles pairs d'indexes on peut générer toutes les analyses possibles: On commence par les arcs passifs qui couvre toute la phrase. Pour chaque pair d'indexes, le deuxième index spécifie quelle production est utilisé pour dériver le symbole à droite. Le premier index spécifie ou chercher les dérivations des symboles à gauche.