Approfondimento sul Pattern Matching 2D

Liva Giovanni

Università di Udine

2 giugno 2014

Argomenti

1. Lavori Precedendi
   - Exact vs Approximate
   - Approximate Dictionary Matching
   - Baker and Bird
   - Zhu and Takaoka
2. 2D con FA
3. 2D con PDA

Definizioni di Base

Dimensioni -¿ (m*m') Testo 2D; m Testo 1D
(n*n') Pattern 2D -¿ n Pattern 1D
k errori ammessi
D_H = hamming
P = pattern
T = testo

Exact vs Approximate

Il pattern matching esatto prevede di cercare l'occorrenza di una stringa all'interno di un testo. L'uso di automi per questo approccio è abbastanza naturale come si vede dal seguente algoritmo:

Algorithm 1 Creazione FA :: Pattern - Matching Esatto

1: $\delta(q_0, a) = \{q_0\}, \forall a \in A$
2: $\delta(q_0, p_1) = \{q_0, q_1\}$
3: **for** $i = 1$ to $m - 1$ **do**
4: $\quad \delta(q_i, p_{i+1}) = \{q_{i+1}\}$
5: **end for**

L'automa risultante ha $m + 1$ stati. Ogni stato q_i indica che si è letto il prefisso del pattern fino all'i-esimo carattere

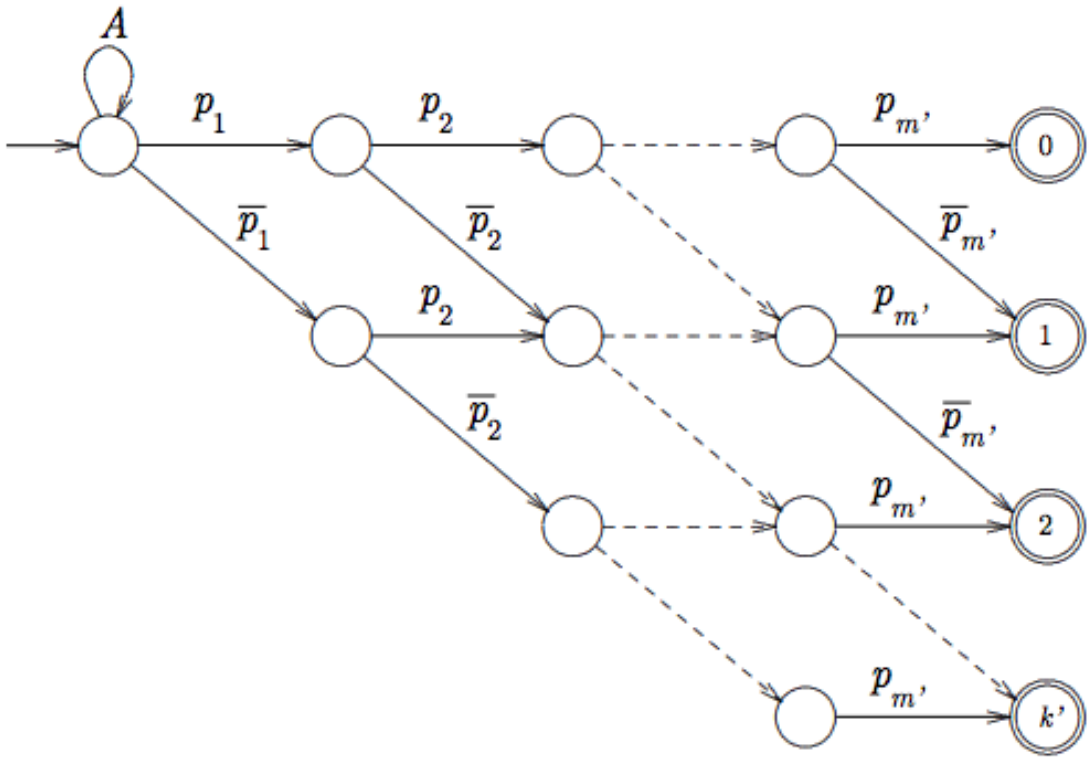
Exact vs Approximate

Il pattern matching approsimato si basa sulla distanza di Hamming che permette di quantificare il numero di errori ammessi. La costruzione di tale automa prevede l'uso di $k+1$ copie di automi per il Pattern Matchin esatto, $M_0, \ldots, M_k$.
L'idea è quella che ogni M_i rappresenta il pattern accettato con i errori. I vari M_i sono collegati con una transizione dallo stato q_j allo stato q_{j+1} che corrisponde all'azione di *sostituzione* nel calcolo della distanza di Hamming, etichettata con il simbolo $\overline{p}_{j+1}$ corrispondente al carattere complementare in posizione $j+1$ in P.

Exact vs Approximate

Theorem

L'automa per il Pattern Matching approsimato ha $(k+1)(m+1-k/2)$ *stati*



Approximate Dictionary Matching

Definition

Sia π un dizionario di s pattern, $\pi = \{p_1, \ldots, p_s\}$
Sia $m = min\{|p_1|, \ldots, |p_s|\}$
Sia k il numero di errori ammessi per ogni pattern, $k < m$
L'automa $\mathcal{A}$ per l'approximate dictionary matching riconosce il linguaggio
$L(\mathcal{A}) = \bigcup_{i=1}^{s} \{uv | u, v \in A^*, D_H(p_i, v) \leq k, p_i \in \pi\}$

Algorithm 2 Creazione FA :: Approximate Dictionary Matching

1: **for** $i = 1$ to s **do**
2: Costruisci M_i con la tecnica per il pattern matching approssimato
3: **end for**
4: Costruisci lo stato iniziale q_0
5: **for** $i = 1$ to s **do**
6: Aggiungi una transizione da q_0 a q_0^i e da q_0 a q_1^i etichettata come la transizione $q_0^i \rightarrow q_1^i$
7: **end for**

Baker and Bird

Definition

Sia π il dizionario ottenuto da PA, $\pi = \{p_i | p_i$ è la i-esima colonna di PA$\}$
Sia $\mathcal{A}$ l'automa ottenuto da PA con l'algoritmo di $Aho - Corasick$
Sia TA' il *textarray* ottenuto lanciando $\mathcal{A}$ su ogni colonna di TA e salvando lo stato corrente della run dell'automa

- Linearizzare TA' ottenendo T
- Linearizzare PA ottenendo P
- Usare KMP su P e T
- La complessità finale è $\mathcal{O}(mm' + nn')$

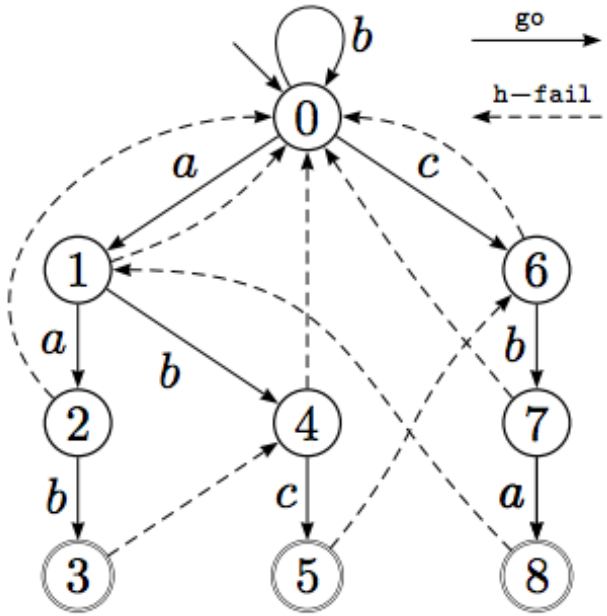
Esempio

$$PA = \begin{array}{|c|c|c|}\hline a & c & a \\ \hline b & b & a \\ \hline c & a & b \\ \hline \end{array}\ ,\ TA = \begin{array}{|c|c|c|c|c|c|c|}\hline b & b & a & b & b & a & b \\ \hline a & a & c & a & c & b & a \\ \hline b & b & b & a & c & a & c \\ \hline a & c & a & b & b & a & b \\ \hline c & a & a & c & a & b & a \\ \hline b & b & b & b & a & c & c \\ \hline a & c & c & a & b & a & b \\ \hline \end{array}\ , |PA| = (3 \times 3), |TA| = (7 \times 7).$$

Esempio

$$PA = \begin{array}{|c|c|c|}\hline a & c & a \\ \hline b & b & a \\ \hline c & a & b \\ \hline \end{array}$$

5 8 3



Esempio

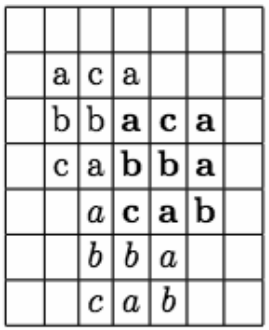
$TA=$

b	b	a	b	b	a	b
a	a	c	a	c	b	a
b	b	b	a	c	a	c
a	c	a	b	b	a	b
c	a	a	c	a	b	a
b	b	b	b	a	c	c
a	c	c	a	b	a	b

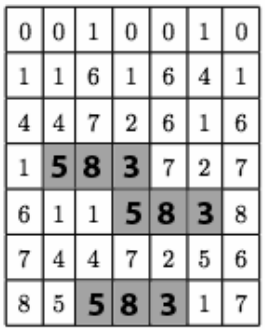
$TA'=$

0	0	1	0	0	1	0
1	1	6	1	6	4	1
4	4	7	2	6	1	6
1	5	8	3	7	2	7
6	1	1	5	8	3	8
7	4	4	7	2	5	6
8	5	5	8	3	1	7

Esempio



	a	c	a			
	b	b	**a**	**c**	**a**	
	c	a	**b**	**b**	**a**	
		a	**c**	**a**	**b**	
		b	*b*	*a*		
		c	*a*	*b*		



0	0	1	0	0	1	0
1	1	6	1	6	4	1
4	4	7	2	6	1	6
1	**5**	**8**	**3**	7	2	7
6	1	1	**5**	**8**	**3**	8
7	4	4	7	2	5	6
8	5	**5**	**8**	**3**	1	7

Zhu and Takaoka

Estende l'idea di Karp e Rabin alle due dimensioni usando la tecnica delle *impronte*.
I passi dell'algoritmo sono:

- Genera P' come array di impronte di P usando la funzione di hash per colonne
- Genera T' nello stesso modo di P' (Guardando solo m' caratteri per colonna)
- Lancia KMP su P' e T' e ogni volta che trova una occorrenza esegue il controllo sulle matrici
- Passa alla riga successiva aggiornando T'

Più il pattern ha la componente m' grande, migliore è l'efficenza di questo algoritmo che, nel caso pessimo è $\mathcal{O}(n'n + n'mm')$

2D Approsimato con FA :: Algoritmo Generico

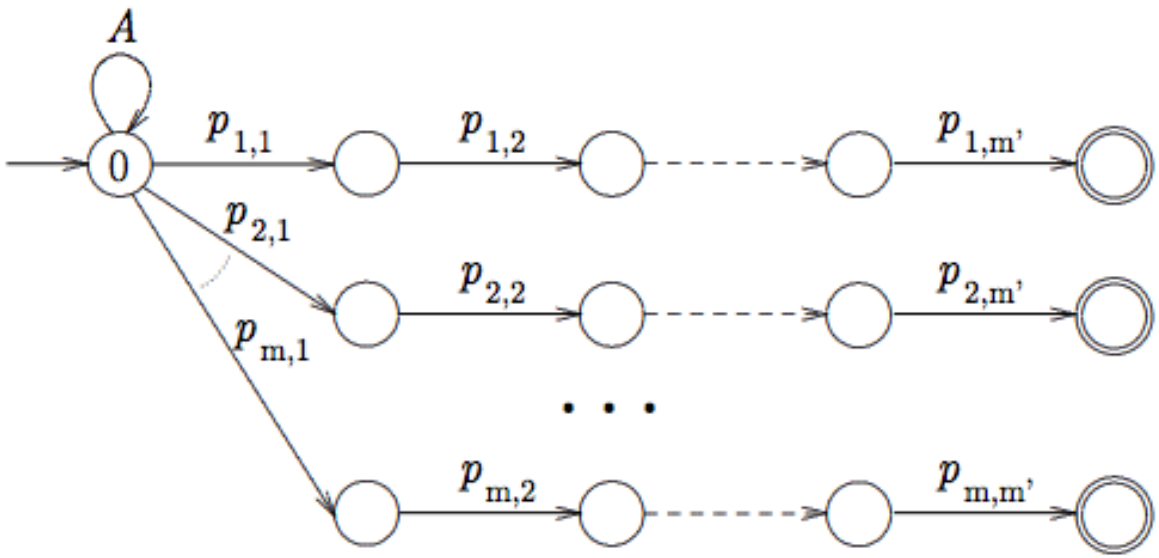
Definition

Sia π il dizionario ottenuto da PA, $\pi = \{p_i | p_i$ è la i-esima colonna di PA$\}$
Sia R la linearizzazione di PA
Sia $M(\pi) = (Q, A, \delta, I, F)$ l'automa costruito con la tecnica per l'approximate dictionary matching

- Costruiamo $M(\pi)$ con k' errori
- Costruiamo TA' ottenuto dai valori della run di $M(\pi)$ su TA
- Costruiamo R come l'insieme degli stati finali di $M(\pi)$ che rappresentano il match esatto di PA
- Costruiamo M' che cerca R con k errori (limitiamo la somma degli errori del punto 2)
- Lanciamo M' su TA' in modo che rilevi tutte le occorrenze di R

Caso Esatto :: Costruzione $M(\pi)$

$M(\pi)$ riconosce il linguaggio $L = A * E(\pi)$ con $E(\pi) = \{P \in A^{m'}; P \in \pi\}$
La costruzione di tale automa consiste in un *Trie* con un self loop sullo stato iniziale etichettato con A e m transizioni verso gli automi per i singoli pattern.



Caso Esatto :: Costruzione $M(\pi)$

Theorem

L'automa ha al più uno stato finale attivo dopo aver letto un carattere d'input.

Dimostrazione.

Siccome π è un insieme, non esistono duplicati di una stringa.
Visto che ogni stringa all'interno di π ha la stessa lunghezza, dopo ogni step dell'automa solo, al più, uno stato finale può essere attivo. □

Questo teorema vale grazie a come viene eseguita la simulazione di $M(\pi)$.

Caso Esatto :: Costruzione TA'

Grazie al teorema precedente possiamo costruire TA' usando la seguente formula:

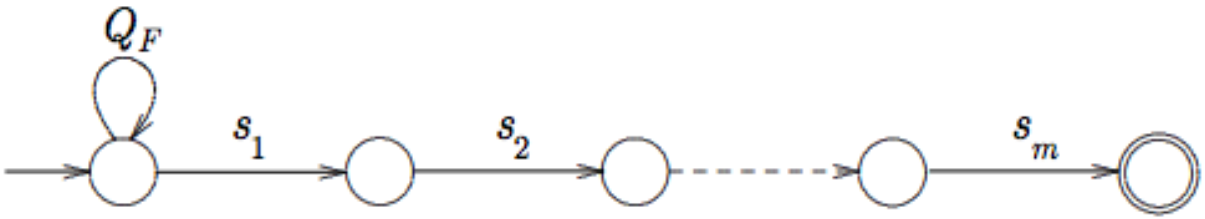
$$\forall i, j \ 1 \leq i \leq n, 1 \leq j \leq n' \ \ TA'[i,j] = \begin{cases} q & \text{se } q \in F \\ 0 & \text{se } q \notin F \end{cases}$$

Caso Esatto :: Costruzione R

La stringa R è ottenuta linearizzando PA grazie all'automa $M(\pi)$ costruito in precedenza.
L'i-esimo carattere di R è ottenuto leggendo lo stato finale di $M(\pi)$ su input PA[i].

Caso Esatto :: Costruzione M'

Costruiamo l'automa M' con il solito algoritmo sull'alfabeto $F \cup \{0\}$ che riconosca il linguaggio $L(M') = (F \cup \{0\}) * E(R)$ dove
$E(R) = \{R \mid R \in F^m\}$, $F = \{s1, \ldots, s_{|\pi|}\}$, $|F| = |\pi|$.



Caso Esatto :: Conclusioni

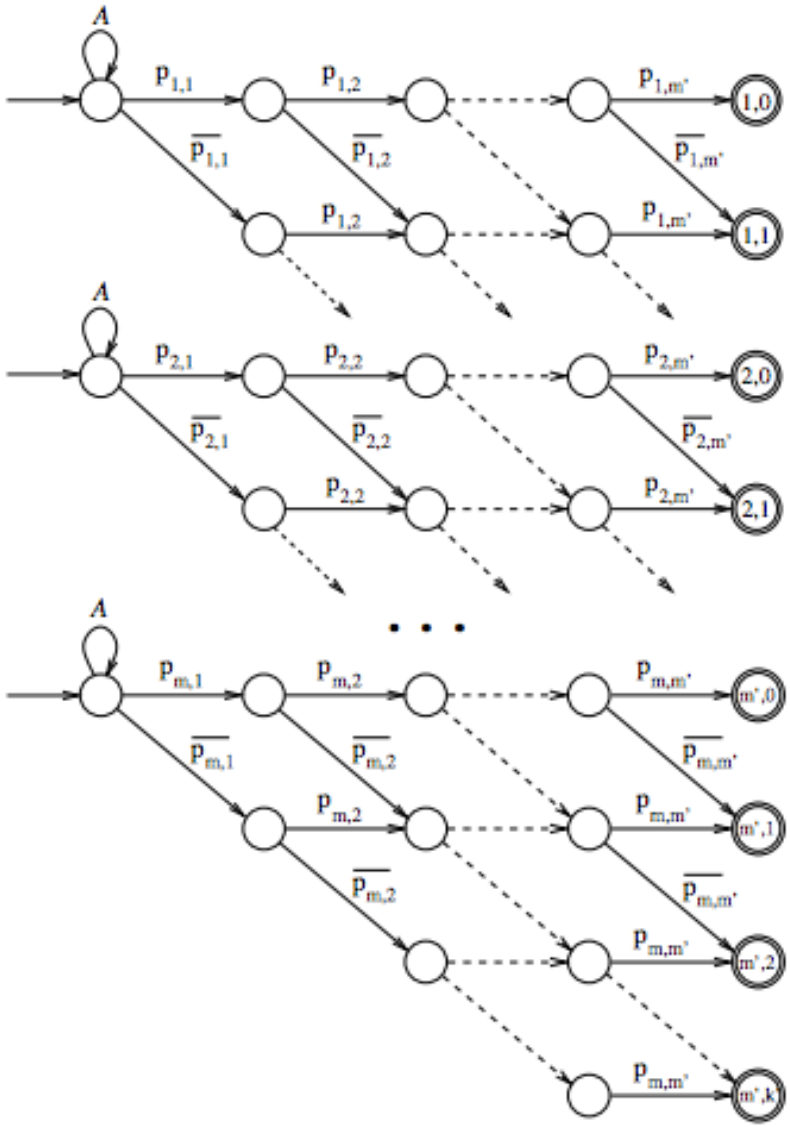
- Tutti gli automi presentati sono non deterministici
- Bisogna utilizzare una simulazione per non dover determinizzare gli automi
- Opera in tempo lineare: $\mathcal{O}(mm' + nn')$
  - $\mathcal{O}(mm')$ per $M(\pi)$
  - $\mathcal{O}(nn')$ per TA'
  - $\mathcal{O}(m)$ per R ed M'
  - $\mathcal{O}(nn')$ per il pattern matching

Caso Approssimato

Dobbiamo modificare $M(\pi)$ in modo che accetti il linguaggio $L(M) = A * H_k(\pi)$ dove:
$H_k(\pi) = \{X \mid X \in A^*, D_H(X, P) \leq min(k, m' - 1) \wedge P \in \pi\}$
Dobbiamo usare il minimo perchè k potrebbe essere più grande della dimensione di una colonna/righa.
Bisogna costruire l'automa per l'Approximate Dictionary Matching con l'algoritmo visto in precedenza con una modifica, gli stati finali sono etichettati con copie (s, x) dove:

- s = l'indice del pattern riconosciuto
- x = il numero di errori con il quale viene riconosciuto

I singoli automi del ADM



Caso Approssimato

Theorem

L'automa per il ha $|\pi|(k+1)(m+1-k/2)$ *stati*

Theorem

L'automa potrebbe avere più di uno stato finale attivo dopo aver letto un carattere d'input.