Complexitatea algoritmului de String Matching privit ca arbore de decizie[1]

Xiaoyu He, Neng Huang, Xiaoming Sun
29 iunie 2018

Link-uri: https://arxiv.org/abs/1712.09738
http://drops.dagstuhl.de/opus/volltexte/2018/9508/

INTRODUCERE

Problema de string-matching constă în determinarea apariției unui pattern într-un string (a unei apariții sau a tuturor aparițiilor). În prezent există a serie de algoritmi care rezolvă această problemă optim, spre exemplu algoritmul KMP dezvoltat de Knuth, Morris și Pratt are complexitatea timp $O(n + m)$, unde n este lungimea string-ului în care se face căutarea și m este lungimea pattern-ului. Acest algoritm este optim deoarece complexitatea lui coincide cu complexitatea citirii inputului.

Considerând că avem o serie de informații apriori despre string-ul în care se face căutarea putem defini complexitatea ca numărul de caractere din string-ul de căutare interogate de algoritm. Spre exemplu peste alfabetul $\Sigma = \{0, 1\}$ această măsură reprezintă complexitatea arborelui de decizie[2] a problemei de căutare pe string-uri booleene.

NOTAȚII

Fie p un pattern peste alfabetul Σ cu $|\Sigma| = \sigma$, unde $p = p[1] \cdot p[2] \cdot p[3] \cdot \ldots \cdot p[m]$ și $\cdot$ reprezinta operatia de concatenare a caracterelor ($|p| = m$). Fie $p[i \ldots j]$ substring-ul de la indexul i pana la j inclusiv.

Fie A_p un algoritm de string-matching determinist care caută pattern-ul p în orice string s. Definim $D_p(n) = \min_{A_p} w(A_p, n)$ unde $w(A_p, n) = \max_{\substack{s \\ |s|=n}} (\langle A_p(s) \rangle)$ unde $\langle A_p(s) \rangle$ = numărul de caractere interogate de A_p primind string-ul s la intrare. Cu alte cuvinte $D_p(n)$ reprezintă numărul maxim de caractere interogate peste toate string-urile de lungime n, efectuate de cel mai performat algoritm de string-matching (numărul minim de caractere interogate pe worst-case). În plus, peste $\Sigma = \{0, 1\}$, $D_p(n)$ reprezintă complexitatea arborelui de decizie boolean a problemei de string-matching.

Definiția 1: Un pattern p este evaziv daca $\exists\ N_0 \in \mathbb{N}$ astfel încât $D_p(n) = n, n > N_0$. (Echivalent un pattern este evaziv dacă orice algoritm determinist interoghează toate caracterele inputului pentru a determina dacă p este în acesta).

Pentru a defini non-evazivitatea în paper se folosesc câteva teoreme și observații demonstrate în trecut, printre care:

[1] Titlul original: <<On the Decision Tree Complexity of String Matching>>

[2] Complexitatea unei probleme sau a unui algoritm privit ca model de arbore de decizie se numește complexitatea arborelui de decizie

1. $D_p(n) \leq D_p(n+1)$
2. Margine inferioară lineară pentru $D_p(n)$ dată de Rivest: $D_p(n) \geq n - |p| + 1, \forall n \epsilon \mathbb{N}$
3. Observația demonstrată a lui Rivest cum ca $\nexists p$ astfel încât $D_p(n) = n - |p| + 1$ si $D_p(n+1) = n - |p| + 2, \forall n \epsilon \mathbb{N}$ (limita inferioară a inegalității din teorema anterioara nu se atinge niciodată pentru n-uri consecutive).

Definiția 2: Un pattern p este non-evaziv dacă $\forall N_0 \in \mathbb{N}$ atunci $\exists n > N_0$ astfel încât $D_p(n) = n - |p| + 1$.

Definiția 3: Fie un pattern p cu $|p| = m$ și $k \in \mathbb{N}, cu\ k < m$. Spunem că p este $k - periodic$ sau p are perioada k dacă $p[i] = p[i+k]\ \forall\ i \in \overline{1 \dots m-k}$ (echivalent $p[1 \dots (m-k)] = p[(k+1) \dots m]$). Fie $\boldsymbol{Period}(p) = \{k | p\ este\ k - periodic\}$. Convențional un string p care nu are nici o perioada valida conform definiției va avea o singură perioadă și anume $|p|$.

TEOREMA LUI TUZA (Worst-case behavior of string-searching algorithms)

Fie mulțimea $BE(b) = \{p \in \Sigma^* \mid \mathrm{p} = \mathrm{b} \cdot \omega_1 \cdot \omega_2 \cdot \dots \cdot \omega_i \cdot \mathrm{b}, \omega_j \epsilon \Sigma, \mathrm{j} \epsilon \overline{1 \dots \imath}, i \geq 1, |b| > 0\} \cup \{p \in \Sigma^* \mid p[1 \dots |b|] = b,\ p[m - |b| + 1 \dots m] = b, 0 < |b| < |p| = m < 2|b|\ \}$ (string-uri care încep și se termină cu b dar sunt diferite de b, b este bifix pentru p).

Dacă $p \in BE(b)$ atunci fie $p(b) = u \cdot b \cdot v$ unde $u \cdot b = b \cdot v = p$.

Teorema lui Tuza: Fie $p \in BE(b)$. Dacă:

1. $p(b)$ nu conține un substring p', $|p'| = |p|$, altul decât sufixul sau prefixul lui $p(b)$ astfel încât $p'\ și\ p$ diferă prin cel mult 2 caractere, și
2. Pattern-ul pp nu conține un substring p', $|p'| = |p|$, altul decât sufixul sau prefixul lui pp astfel încât $p'\ și\ p$ difera prin cel mult 4 caractere, și
3. $n\ \geq \frac{|p|(2|p| - |b|)}{\gcd(|p|, |b|)}$

atunci $D_p(n) \geq n - k, unde\ k = n\ mod\ (\gcd(|p|, |b|))$.

Ca observație, atunci când $\gcd(|p|, |b|) = 1 \Rightarrow k = n\ mod\ 1 = 0$
$\Rightarrow D_p(n) \geq n \xRightarrow{D_p(n) \leq n} D_p(n) = n$. În plus teorema de mai sus pentru $\Sigma = \{0, 1\}$ stabilește și proporția pattern-urilor evazive care este cel puțin $0.5061 \cdot 2^m$ din totalul de 2^m string-uri de lungime m peste Σ.

UPPER BOUNDS

Lema 1 : Fie un pattern $p, cu\ |p| = m$ si $c = \gcd\big(\boldsymbol{Period}(p)\big)$ atunci $D_p(n) \leq n - (n\ mod\ c)$.

În continuare aceasta lema se va demonstra pe două cazuri: 1. Daca p este un pattern liber de bifixe și 2. p este un pattern oarecare.

1. DACA P ESTE UN PATTERN LIBER DE BIFIXE

Definiția 4: Un string $b, |b| = k$ se numește bifix al string-lui $p, |p| = m > k$ dacă b este atât prefix cat și sufix al lui p ($p[1 \dots (m-k)] = p[(k+1) \dots m]$). Un string p se numeste liber de bifixe dacă nu are alte bifixe decât pe el însuși. (Echivalent $p\ liber\ de\ bifixe\ \Leftrightarrow \nexists\ b\ astfel$ încât $p \in BE(b)$).

Lema 2: Un pattern $p, |p| = m$ are un bifix de lungime $k < m$ ⇔ p este $(m-k)-periodic$. Mai mult p este liber de bifixe ⇔ p are doar o perioadă, anume m.

Demonstrație: p are un bifix de lungime $k < m \xLeftrightarrow{Definitia\ 4} p[1 \dots (m-k)] = p[(k+1) \dots m]$ $\xLeftrightarrow{Definitia\ 3} p\ este\ (m-k)-periodic.$

$p\ este\ liber\ de\ bifixe \xLeftrightarrow{Definitia\ 4} \nexists k < m\ astfel$ încât $p[1 \dots (m-k)] = p[(k+1) \dots m]$ $\xLeftrightarrow{Definitia\ 3} p\ nu\ are\ perioad$ă $valid$ă $\xLeftrightarrow{Convenție\ definitia\ 4} Period(p) = \{m\}$

Lema 3 (Restricție a Lemei 1): Fie un pattern $p, |p| = m$ liber de bifixe atunci $D_p(n) \le n - (n \bmod c)$ ceea ce înseamnă ca p este non-evaziv, unde $c = \gcd\big(Period(p)\big) = \gcd(\{m\}) = m$.

Demonstrație: Fie următorul algoritm:

```
Input: string s of length n, bifix-free pattern p of length m
Output: whether p is a substring of s
 1: function FIND(s, p)
 2:     if n < m then
 3:         return false
 4:     i ← m, j ← m
 5:     query(s[m])
 6:     while j − i ≠ m − 1 do
 7:         if s[i..j] is a suffix of p then
 8:             query(s[i − 1]), i ← i − 1
 9:         else
10:             query(s[j + 1]), j ← j + 1
11:     if s[i..j] = p then
12:         return true
13:     else
14:         return FIND(s[m + 1..n], p)
15: end function
```

În mod evident dacă p nu este pattern în s atunci algoritmul face $\left[\frac{|s|}{|p|}\right]|p| = \left[\frac{n}{m}\right]m$ pași, și asta este unul din worst-cases ($\left[\frac{n}{m}\right]m$ deoarece de la linia 6 la linia 10 se fac m query-uri iar recursia se face din m în m caractere pe s, deci de $\left[\frac{n}{m}\right]$ ultimele $n - \left[\frac{n}{m}\right]m$ caractere fiind skip-uite) . În plus $\left[\frac{n}{m}\right]m = n - (n \bmod m)$. Și cum acest număr de query-uri este realizabil în worst-case atunci $D_p(n) \le n - (n \bmod m)$ ($\le$ deoarece dacă p se află în s mai la început acesta va fi găsit și algoritmul se va opri).

În continuare rămâne de arătat corectitudinea algoritmului. Pentru a arata acest lucru este suficient (datorită recursivității care scurtează noua problemă la un nou s de lungime $n-m$) să arătam că $\forall k, 1 \le k \le m\ avem$ că $s[k \dots (k+m-1)] \ne p$ (Adică în porțiunea $s[1 \dots (2m-1)]$ nu se poate match-ui pattern-ul p). Algoritmul încearcă sa potrivească in s patternul p începând cu poziția $|p|$ din s, căutând în vecinătatea $V(|p|, |p|) = \{1, 2, \dots, m, \dots 2m-1\}, |p| = m.$ În mod evident dacă $k = 1 \Rightarrow p = s[1 \dots m]\ iar\ algoritmul\ de\ mai\ sus\ va\ intra\ de\ fiecare\ dat$ă $pe\ prima\ ramur$ă $a\ if-ului\ \Rightarrow\ la\ final\ j = m(valoarea\ initial$ă$)\ iar\ i = 1\ ceea\ ce\ este\ corect\ \Rightarrow\ k > 1$

Dacă $1 < k < i$. În plus $j - i + 1 = m \Rightarrow k + m - 1 < j \Rightarrow$ s-a intrat de mai multe ori de cât ar fi trebuit pe a doua ramură a if-ului ⇒ s-a picat testul $s[i \dots j]\ prefix\ al\ lui\ p$ cel puțin o data, ceea ce este imposibil deoarece $s[i \dots k+m-1]$ este sufix al lui $p\ \forall\ i\ \epsilon \overline{k, m} \Rightarrow\ k \ge i$.

$Dacă$ $k = i$ atunci algoritmul pe liniile 11-12 ar întoarce ture ceea ce este o contradicție $\Rightarrow k \neq i$.
$Dacă$ $k > i$ se rezolva analog cazului $k < i$.

2. DACĂ p ESTE UN PATTERN OARECARE (CAZUL GENERAL)

Demonstrație la Lema1: Fie următorul algoritm, unde $c = \gcd\big(\boldsymbol{Period}(p)\big)$:

```
Input: string s of length n, pattern p of length m
Output: whether p is a substring of s
 1: function FIND(s, p)
 2:     if n < m then return false
 3:     i ← m, j ← m
 4:     query(s[m])
 5:     while j − i ≠ m − 1 do
 6:         if s[i..j] is a suffix of p then
 7:             query(s[i − 1]), i ← i − 1
 8:         else
 9:             query(s[j + 1]), j ← j + 1
10:     if s[i..j] = p then
11:         return true
12:     l ← m + c
13:     while l ≤ n do
14:         i ← l, j ← l
15:         query(s[l])
16:         repeat
17:             if s[i..j] is a suffix of p then
18:                 query(s[i − 1]), i ← i − 1
19:             else
20:                 query(s[j + 1]), j ← j + 1
21:         until c new characters have been queried OR j − i = m − 1
22:         if s[(j − m + 1)..j] = p then
23:             return true
24:         l ← l + c
25:     return false
26: end function
```

Acest algoritm reprezintă o generalizare a algoritmului anterior însă de aceasta dată string-ul de intrare nu mai este liber de bifixe ci poate avea o serie de perioade. Liniile 3-11 caută pattern-ul p în vecinătatea $V(|p|,|p|)$. Apoi liniile 12-25 caută pattern-ul p în vecinătățile $V(|p|+c,|p|), V(|p|+2c,|p|), \ldots V\left(|p| + \left[\frac{|s|-|p|}{c}\right]c, |p|\right)$

În mod evident dacă p nu este pattern in s atunci algoritmul face $|p| + \left[\frac{|s|-|p|}{c}\right]c = m + \left[\frac{n-m}{c}\right]c = \left[\frac{m}{c}\right]c + \left[\frac{n-m}{c}\right]c = \left(\left[\frac{m}{c}\right] + \left[\frac{n-m}{c}\right]\right)c \leq \left[\frac{m+n-m}{c}\right]c = \left[\frac{n}{c}\right]c$ pași, și asta este unul din worst-cases (saltul se face din c în c caractere pe s de la m mai departe (nu contează că este de la m în colo deoarece și pană la m pot fi privite salturi de lungime c deoarece $\left\{\frac{m}{c}\right\} = 0$), deci $\left[\frac{n}{c}\right]c$ caractere vor fi examinate, rezulta că ultimele $n - \left[\frac{n}{c}\right]c$ caractere sunt skip-uite) . În plus $\left[\frac{n}{c}\right]c = n - (n \bmod c)$. Cum acest număr de query-uri este realizabil în worst-case atunci $D_p(n) \leq n - (n \bmod c)$ ($\leq$ deoarece dacă p se află în s mai la început acesta va fi găsit și algoritmul se va opri).

Deci s-a stabilit o margine superioară pentru numărul de caractere interogate pe string-uri de lungime n pe worst-case de cel mai performant algoritm $D_p(n) \leq n - (n \, mod \, c)$, unde algoritm $c = \gcd\big(\boldsymbol{Period}(p)\big)$.

ESENȚA LUCRĂRII

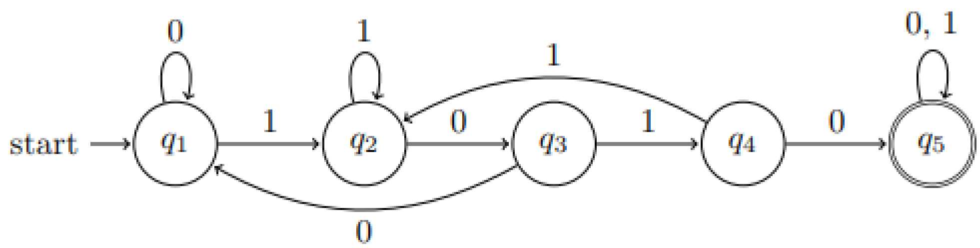
Teorema 1: Fie un pattern $p \in \Sigma$ cu $|\Sigma| = \sigma$, $|p| = m$ și $c = \gcd\big(\boldsymbol{Period}(p)\big)$ atunci $D_p(n) = n - (n \, mod \, c)$ exceptând o fracțiune $O(m^5 \sigma^{\frac{-m}{2}})$ de pattern-uri.

Se poate observa că $\lim\limits_{m \to \infty} m^5 \sigma^{\frac{-m}{2}} = \lim\limits_{m \to \infty} \frac{m^5}{\sigma^{\frac{m}{2}}} = \begin{cases} 0, daca \ \sigma > 1 \\ \infty, daca \ \sigma = 1 \end{cases}$.

Anterior s-a stabilit că este decidabil dacă $p \in s$ după maxim $n - (n \, mod \, c)$ interogari (Lema 1). Ei bine această teorema (Teorema 1) afirmă că sunt necesare numărul maxim de query-uri pentru a decide dacă $p \in s$ exceptând o serie de $O(m^5 \sigma^{\frac{-m}{2}})$ de pattern-uri.

EVAZIVITATEA PATTERN-URILOR PESTE $\boldsymbol{\Sigma = \{0, 1\}}$

Folosind algoritmul KMP pentru un pattern p peste $\Sigma = \{0, 1\}$ se poate construi un Automat Finit Determinist, cu $|p| + 1$ stări (notate q_i $i \in \overline{1, |p| + 1}$ în care q_1 este starea inițială și $q_{|p|+1}$ este starea finală, care să accepte string-uri s de orice lungime doar dacă $p \in s$. Spre exemplu pentru $p = 1010$ DFA-ul este următorul:



Fie $KMP_p(s) = \ i \ dacă \ q_i \ este \ ultima \ stare \ a \ automatului \ după \ parsarea \ lui \ s$ și $U_p(n, i) = \{s \mid |s| = n, \ KMP_p(s) = i\}$. Mai mult $KMP_p(s) \in \overline{1, |p| + 1}$. Fie $g_p(n, i) = \sum_{s \in U_p(n,i)} x^{\|s\|_1} \in \mathbb{R}[X]$ cu $\deg\Big(g_p(n, i)\Big) < n + 1$. Conform Rivest si Vuillemin avem următoarea lemă (**Lema R-V**), cum că dacă $\exists l \in \overline{1, n}$ astfel încât $D_p(n) \leq n - l$ atunci $g_p(n, |p| + 1) = 0 \, mod (x + 1)^l$. Iar pentru aceasta lemă se deduce următorul corolar: dacă $\exists N_0 \in \mathbb{N}$ astfel încât $g_p(n, |p| + 1) \ \neq 0 \, mod \ (x + 1) \ \forall n > N_0$ atunci $D_p(n) = n$ și implici p este evaziv.

Cum $g_p(n, i) \in \mathbb{R}[X]$ are $\deg\Big(g_p(n, i)\Big) < n + 1$ si $g_p(n + 1, j) \in \mathbb{R}[X]$ are $\deg\Big(g_p(n + 1, j)\Big) < n + 2, \forall i, j \ \Rightarrow \ g_p(n + 1, |p| + 1) = (x + 1) g_p(n, |p| + 1) + r \cdot g_p(n, |p|), r \in \{0, \pm 1\}$ (intuitiv și în conformitate cu lema R-V: sumă după numărul de 1 din toate string-urile acceptate de lungime $n + 1$ este multiplu de $(x + 1)$ (lema R-V) la o putere cel puțin 1, înmulțit cu numărul de 1 din toate string-urile de lungime n acceptate plus suma tuturor caracterelor 1 din toate string-urile de lungime n la care automatul se oprește în penultima stare). De aici se deduce ca $g_p(n + 1, |p| + 1) \, mod \ (x + 1) \ = \big[(x + 1) g_p(n, |p| + 1) + r \cdot g_p(n, |p|)\big] \, mod (x + 1)$

$\Leftrightarrow g_p(n+1,|p|+1)\ mod\ (x+1) = r \cdot g_p(n,|p|)\ mod(x+1)$ Dar cum $r \in \{0,\pm 1\} \Rightarrow g_p(n+1,|p|+1)\ mod\ (x+1) = g_p(n,|p|)\ mod(x+1) \Leftrightarrow g_p(n+1,|p|+1) \equiv g_p(n,|p|)\ mod(x+1)$.
Și de aici următoarea lemă:

Lema 4: $g_p(n+1,|p|+1) \equiv 0\ mod(x+1) \Leftrightarrow g_p(n,|p|) \equiv 0\ mod(x+1)$. În plus $\exists N_0 \in \mathbb{N}$ astfel încât $g_p(n,|p|+1) \neq 0\ mod\ (x+1)\ \forall n > N_0$ atunci $D_p(n) = n$ și implici p este evaziv.

Pentru a defini $g_p(n+1,i)$ în termeni de $g_p(n,j)_{j\in\overline{1,|p|}}\ \forall i \in \overline{1,|p|}$ se face următoarea construcție:

$$\begin{pmatrix} g_p(n+1,1) \\ g_p(n+1,2) \\ g_p(n+1,3) \\ \dots \\ g_p(n+1,|p|) \end{pmatrix} = T_p \cdot \begin{pmatrix} g_p(n,1) \\ g_p(n,2) \\ g_p(n,3) \\ \dots \\ g_p(n,|p|) \end{pmatrix}, T_p \in \mathcal{M}(\mathbb{R}[X])$$

Unde T_p se construiește astfel:

- considerăm $T_p^* = \left(t_{i,j}^*\right)_{\substack{i\in\overline{1,|p|},\\ j\in\overline{1,|p|}}}, unde\ {t^*}_{i,j} = \begin{cases} 1, \exists\ in\ \ KMP\ tranzitie\ de\ la\ j\ la\ i \\ 0,\ alfel \end{cases}$
- $T_p = \left(t_{i,j}\right)_{\substack{i\in\overline{1,|p|},\\ j\in\overline{1,|p|}}}, unde\ t_{i,j} = \begin{cases} {t^*}_{i,j}, i\ impar \\ x \cdot {t^*}_{i,j},\ \ i\ par \end{cases}$
- $\overline{T_p} = \left(\bar{t}_{i,j}\right)_{\substack{i\in\overline{1,|p|},\\ j\in\overline{1,|p|}}}, unde\ \bar{t}_{i,j} = t_{i,j}(-1)$ (deci teste matricea de polinoame T_p evaluata în $x = -1$ deoarece ne interesează rezultatele $mod(x+1)$ iar $x+1$ se anulează în $x = -1$)
- $\overline{g_p}(n,i) = g_p(n,i)(-1)$

  Deci: $\left(\overline{g_p}(n+1,i)\right)^T_{i\in\overline{1,|p|}} = \overline{T_p} \cdot \left(\overline{g_p}(n,i)\right)^T_{i\in\overline{1,|p|}}$

Fie polinomul caracteristic asociat lui p cu $|p| = m$, polinomul caracteristic al matricei $\overline{T_p}$: $P(\lambda) = \lambda^m + c_{m-1}\lambda^{m-1} + \dots + c_0$. în plus se dovedește că avem următoarea recurență (folosind Hamilton-Cayley):

$$\overline{g_p}(n+m,m) + c_{m-1}\overline{g_p}(n+m-1,m) + \dots + c_0\overline{g_p}(n,m) = 0$$

În plus conform V. Halava, T. Harju, M. Hirvensalo, si J. Karhumaki avem că pentru un șir recurent ${\{u_n\}^\infty}_{n=1}$ unde $u_n = \sum_{i=0}^{m-1} a_i u_{n-m+i}, a_i \in \mathbb{N}\ \forall i \in \overline{0,m-1}$ și un polinom $p(\lambda) = \lambda^m + a_{m-1}\lambda^{m-1} + \dots + a_0$ care are descompunerea $p(\lambda) = \prod_{i=1}^{r}(\lambda - \lambda_i)^{m_i}$ unde $\lambda_i \epsilon \mathbb{C}, i\epsilon\overline{1,r}$ sunt rădăcini distincte, cu $|\lambda_1| \geq |\lambda_2| \geq \dots \geq |\lambda_r|$ și una din următoarele condiții se întâmplă:

- $|\lambda_1| > |\lambda_2|$ sau
- $|\lambda_1| = |\lambda_2| > |\lambda_3|\ cu\ \lambda_1 = \overline{\lambda_3}$ sau
- $|\lambda_1| = |\lambda_2| = |\lambda_3| > |\lambda_4|\ cu\ \lambda_2 \in \mathbb{R}, \lambda_2 = \overline{\lambda_3}$

atunci $\exists N_0 \in \mathbb{N}$ astfel încât $u_n \neq 0\ \forall\ n > N_0$. În plus este decidabil algoritmic că există o constantă $N_0 \in \mathbb{N}$ astfel încât $u_n \neq 0\ \forall\ n > N_0$, deci tot ceea ce trebuie făcut este să se determine existența zerourilor în șirul $\{u_n\}$ cu $n \leq N_0$.

În plus se demonstrează că $c_{m-k} = \begin{cases} (-1)^{\|p[1...k]\|_1}, k \in Period(p) \\ 0, alfel \end{cases}$. Ceea ce este computațional mai bine decât varianta anterioara în care trebuia calculata matricea de tranziție $\overline{T_p}$.

CONCLUZII

În acest paper se stabilește o margine exacta pentru a decide dacă un pattern $p \in s$ după maxim de interogări exceptând o serie de neglijabilă de pattern-uri. Acest lucru este realizat după o analiză a proprietăților structurale a pattern-urilor în funcție de periodicitatea caracterelor ce apar în pattern-uri.

Apoi se conturează a abordare algoritmică pentru a decide dacă un pattern peste alfabetul $\Sigma = \{0, 1\}$ este evaziv. Acest lucru este realizabil prin două metode algebrice: prin comparație cu problema Skolem din matematică și prin analiza polinomului caracteristic asociat unui pattern.