7J-06

正規表現に対する最短文字列検索アルゴリズムの並列化

大邊　陽介†　山本　博章†　藤原　洋志†

信州大学工学部†

1. はじめに

正規表現（RE）に対する、文字列検索アルゴリズムは、コンピュータサイエンスにおいて重要な役割を果たしており、広く研究されている。RE 最短部分文字列検索問題は、r に一致する T のすべての最短部分文字列を検索することであり、ここで、r はアルファベット Σ 上の長さ m の RE であり、T は長さ n のテキストである。Clarke および Cormack によって提案された、Thompson オートマトンを使用する伝統的なアルゴリズム [1] は、$O(m^2n)$ 時間と $O(m)$ 空間で、RE 最短部分文字列検索問題を解く。この問題に対して、筆者らは、$O(mn)$ 時間、$O(m)$ 空間のアルゴリズムを開発した。提案されたアルゴリズム [2] は、ε 遷移を有する NFA である Thompson オートマトン [3] に基づいている。本論文では、このアルゴリズムを並列化に向け拡張し、実験的に評価する。

2. 正規表現と RE 最短部分文字列検索問題

2.1 正規表現と演算記号

ここで、RE の定義を与える。

定義 1　Σ をアルファベットとする。Σ 上の RE は以下のように定義される。

(1) ϕ, ε(空の文字列)、および a($\in \Sigma$) は、それぞれ空集合、集合 $\{\varepsilon\}$、および集合 {a} を表す RE である。

(2) r_1 と r_2 をそれぞれ集合 R_1 と R_2 を表す RE とする。その場合、$r_1 + r_2, r_1r_2, r^*$ も、それぞれ集合 $R_1 \cap R_2$ (結合)、R_1R_2 (連結)、および R_1^*(スター閉包) を表す RE である。

通常、RE 内の不要な括弧は、「スター閉包は、連結および結合よりも優先度が高く、連結は結合よりも優先度が高い」という優先規則に従って削除される。ここで、L(r) は RE r によって表される言語を表すとする。RE の長さ m は r に出現するアルファベット記号と演算子 (結合、連結、およびスター閉包) の数を表す。$T = a_1 \cdots a_n$ を Σ 上の文字列とする。次に、任意の $1 \leq i \leq j \leq n$ に対して、T の部分文字列 $T[i:j]$ を $T[i:j] = a_i \cdots a_j$ として定義する。空の文字列 ε も T の部分文字列であることに注意する。部分文字列 $T[i:j]$ が T と等しくない場合、$T[i:j]$ は T の *proper subtring* と呼ばれる。

定義 2　r を RE とする。次に、文字列の集合 $P(r)$ を次のように定義する。

$P(r) = \{$x | x ∈ L(r) and for any *proper substring* y of x, y $\notin \mathbf{L(r)}\}$

集合 $\mathrm{Match}(r,T) = \{(i,j) | i \leq j\ , T[i:j] \in P(r)\}$ を定義する。 定義 2 から、$\varepsilon \in L(r)$ であれば、$P(r) = \varepsilon$ となる。よって $\mathrm{Match}(r,T) = \emptyset$ である。したがって、本論文では $\varepsilon \notin L(r)$ となるような RE r のみを考える。

2.2 RE 最短部分文字列検索問題

定義 3　任意の RE r と任意のテキスト T に対して、RE 最短部分文字列検索問題は $\mathrm{Match}(r,T)$ を見つけることである。

次の例を考える。$r = ab(a+b)^*ba$ を $\{a,b\}$ 上の RE とし、$T = aababaaaabaaaba$ とする。次に、r に一致するすべての部分文字列は、$T[2:6] = ababa$、$T[2:11] = ababaaaaba$、$T[2:15] = ababaaaabaaaba$、$T[4:11] = abaaaaba$、$T[4:15] = abaaaabaaaba$、および $T[9:15] = abaaaba$ で構成される。したがって、$\mathrm{Match}(r,T) = \{(2,6),(4,11),(9,15)\}$ である。

3. Thompson オートマトン

Thompson オートマトン (T-NFA) は、RE の定義に従って、再帰的に構築され、構築アルゴリズムは広く知られている。$M = (Q, \Sigma, \delta, q_0, q_f)$ を長さ m の RE r に対する T-NFA とする。ここで、Q は状態の集合であり、Σ はアルファベットであり、δ は $Q \times (\Sigma \cup \{\varepsilon\})$ から 2^Q への遷移関数であり、q_0 は初期状態、q_f は最終状態である。したがって、T-NFA は 1 つの初期状態と 1 つの最終状態を持つ。

4. 並列化アルゴリズム

4.1 用語

T-NFA M の遷移のシーケンスを *path* と呼ぶ。特に、ε 遷移のみからなるシーケンスを $\varepsilon - path$ と呼ぶ。また、任意の状態 $q \in Q$ に対して、q が 2 つの入力遷移を有する場合、q は *junction state* と呼ばれる。加えて、任意の状態 $q \in Q$ に対して、q のすべての入力遷移は空の文字列 ε または Σ 内のアルファベット記号のいずれかを有する。すべての入力遷移が ε を持つ場合、状態 q を $\varepsilon - state$ と呼び、すべての入力遷移がアルファベット記号を持つ場合、状態 q を symbol state ($s - state$) と呼ぶ。

4.2 概要

Algorithm1 では、配列 Start を使用して、T-NFA M が状態 q_0 から状態 q に到達できる最短の部分文字列の開始位置を格納する。入力としてスレッドの番号 *name* を入力として与えており 7〜21 行目の for ループにおいて開始位置を指定する。この開始位置はスレッドの生成数 k に対して、テキストも等しく分割し、各スレッドに対して、分割された各テキストを割り当てると考えたときの、各テキストの開始位置を意味する。7〜21 行目は指定された開始位置から文末まで検索するが、8 行目の UpdateStart の返り値によって、9 行目の判定でそのスレッドの検索を打ち切ることができる (後述)。Algorithm2 では、配列 Next が配列 Start を計算するための一時配列として使用されている。20〜21 行目では 19 行

On a Parallel Shortest String Search Algorithm for Regular Expressions

† Shinshu University

目で Next[q] が更新されたときに、更新された値がこのスレッドの範囲に対応するテキストの部分文字列の開始位置であるなら、変数 $flag$ を ON にする。即ち、配列 Next がこのスレッドの範囲に対応するテキストの部分文字列の開始位置を１つも保持していないならば、$flag$ は OFF のままで、Algorithm1 の 8 行目に返され、9 行目の条件に一致し、このスレッドの検索を打ち切る。

Algorithm 1 REShortSearch($r, T, name$).

Input: an RE r , a string $T = a_1 \cdots a_n$, where $a_i \in \Sigma$ and a thread number $name$.
Output: Match(r, T).

```
 1: Generate a T-NFA M = (Q, Σ, δ, q_0, q_f) from r .
 2: Sort Q in topological order.
 3: for all q ∈ Q do
 4:     Start[q] ← 0
 5: end for
 6: Compute Q_f = { q | q is an s-state such that there is an ε
    -path from q to q_f }
 7: for i = ⌊n × name/k⌋ + 1 to n do
 8:     flag = UpdateStart(M, Start, i, name)
 9:     if flag == OFF then
10:         break
11:     end if
12:     pos = max_{q∈Q_f} Start[q]
13:     if pos > 0 then
14:         Output(pos,i)
15:         for all q ∈ Q do
16:             if Start[q] ≤ pos then
17:                 Start[q] ← 0
18:             end if
19:         end for
20:     end if
21: end for
```

Algorithm 2 Procedure UpdateStart(M,Start,i,$name$).

Input: T-NFA M , Start , input position i and a thread number $name$.

```
 1: loop ← 2
 2: for all q ∈ Q do
 3:     Next[q] ← 0
 4: end for
 5: Start[q_0] ← i
 6: while loop ≠ 0 do
 7:     for all ε -states q ∈ Q − {q_o} in topological order do
 8:         if q is a junction state with q ∈ δ(p_1, ε) and q ∈
            δ(p_2, ε) then
 9:             Start[q] ← max{ Start[p_1],Start[p_2] }
10:         else
11:             Start[q] ← Start[p] , where q ∈ δ(p, ε)
12:         end if
13:     end for
14:     loop ← loop − 1
15: end while
16: flag ← OFF
17: for all s-states q with a transition δ(p, a_i) = {q} by a_i do
18:     if Start[p] > Next[q] then
19:         Next[q] ← Start[p]
20:         if Next[q] < ⌊n × (name + 1)/k⌋ + 1 then
21:             flag ← ON
22:         end if
23:     end if
24: end for
25: for all q ∈ Q do
26:     Start[q] ← Next[q]
27: end for
28: return flag
```

5. 並列化の概要

並列化にあたり Java のインタフェース ExecutorService を用いて実装した。スレッド数は k 個のスレッドを生成し、並列処理を行う。生成したスレッドは thread number $name$ を 0,1,$\cdots$,k-1 とし、各スレッドで Algorithm1 を実行する。アルゴリズムは最良で $O(mn/k)$ 時間で動作する。

6. 実験

6.1 実験環境

実験は以下の環境を用いて行った。スレッド数はコアの数に合わせて、$k = 4$ とした。

OS	Windows10 Pro
CPU	Intel Corei7 3.40GHz
メモリ	16GB
言語	Java

6.2 実験内容

A,C,G,T の 4 文字から構成される 100MB のテキスト T から Match(r, T) を検索する。その際の並列化前と並列化後の検索時間を比較する。

6.3 実験結果

表 1 検索時間

正規表現 r	並列化前 (ms)	並列化後 (ms)
$A(C+G)T$	25264	10150
$A(C+G)^*T$	33092	14213
$A(A+C+G+T)T$	46056	17753
$AC(A+C+G+T)^*GT$	65860	24298

実験の結果から見てわかるように並列化前の処理時間よりも並列化後の処理時間の方が早くなっていることがわかる。また、並列化前と並列化後で各正規表現 r における、Match(r, T) の総数は変化してない。

7. おわりに

本論文では、RE 最短部分文字列検索問題に対して、ε 遷移を有する NFA である Thompson オートマトンを用いた、$O(mn)$ 時間、$O(m)$ 空間のアルゴリズム [2] を並列化に向け拡張し、実験的に評価した。今後の課題として、正規表現に共通集合演算を追加してオートマトンを拡張することが挙げられる。

参考文献

[1] Charles L.A. Clarke, Gordon V. Cormack, On the use of regular expressions for searching text, ACM Trans. Program. Lang. Syst. (TOPLAS)19 (3) (1997) 413–426.

[2] Hiroaki Yamamoto, A faster algorithm for finding shortest substring matches of a regular expression,143:56-60 2018(Dec.)

[3] K. Thompson, Regular expression search algorithm, Commun. ACM11 (6) (1968) 419–422.