

SeqBDD における最長共通部分列・部分文字列アルゴリズム

Algorithms for Computing Longest Common Subsequence and Substring on SeqBDD

伊藤 洋平†
Yohei Ito

青木 洋士‡
Hiroshi Aoki

山下 茂†
Shigeru Yamashita

1. はじめに

近年, Web アクセス情報や DNA 配列などの大規模データから, 有用な情報を取り出すデータマイニングの研究が盛んになっている [1][2]. 特に SeqBDD と呼ばれるデータ構造は, データマイニング分野に応用され, 画期的な有効性が示されている [3][4][5].

最長共通部分列 (Longest Common Subsequence: LCSseq) 問題や最長共通部分文字列 (Longest Common Substring: LCSstr) 問題は, 複数のテキストデータに一致して現れる最長の部分列や部分文字列を調べる問題である. これらを用いることで, 複数のテキストの類似性や類似した文字列を抽出することができる. これらの問題は, 複数の DNA 配列の類似性を測るシーケンスアライメントの問題などに利用される [6]. LCSseq 問題や LCSstr 問題を解くアルゴリズムは, 本来 2 つの文字列を入力とするものであるが, これらの問題を拡張し, 近年 2 つの文字列集合を入力とした場合のアルゴリズムの研究がおこなわれている [7]. しかし, 既存研究のアルゴリズムでは, 入力データが大きくなると, データがメモリに収まらなくなることがある.

提案手法では, 必要な演算だけをキャッシュに残すことによって, メモリ使用量の削減を行う. 実験の結果, メモリ使用量が削減され, 大規模な入力に対しても結果を出力することができることを確認した.

本稿は, 全 5 章で構成されている. まず, 第 2 章で言葉の定義や, データ構造についての説明を述べる. 続いて, 第 3 章で SeqBDD において, LCSseq 問題と LCSstr 問題をメモリ使用量を削減して解くための手法を述べる. 第 4 章で, 提案手法の評価方法と評価結果について述べる. 最後に, 第 5 章でまとめを述べる.

2. 準備

2.1 最長共通部分列・部分文字列問題

最長共通部分列問題とは, 二つの文字列が与えられたとき, それらに共通して現れる最長の部分列の長さを求める問題である. 部分列とは, 文字列中に現れる, 非連続な列のことを意味する. 図 1 は, 最長共通部分列問題の例である. 二つの文字列に共通して現れる最長の部分列は, 図の下線が引かれた列になるため, この列の長さである 8 が解となる.

ABRACADABRA
ECADADABRBCRDARA → 8

図 1: 最長共通部分列の例

ABRACADABRA
ECADADABRBCRDARA → 5

図 2: 最長共通部分文字列の例

最長共通部分文字列問題は, 二つの文字列に共通して現れる, 最長の部分文字列の長さを求める問題である. 部分文字列とは, 文字列中に現れる, 連続な列のことを意味する. 図 2 は, 最長共通部分文字列問題の例である. 二つの文字列に共通して現れる最長の部分文字列は, 図の下線が引かれた列になるため, この列の長さである 5 が解となる.

2.2 文字列集合における最長共通部分列・部分文字列問題

文字列集合における最長共通部分列・部分文字列問題は, 2 つの文字列集合 X, Y を入力とし, X の任意の文字列と Y の任意の文字列との組み合わせの内, 最も長い共通部分列・部分文字列の長さを求める問題である.

例えば, $X = \{ab, acd\}$, $Y = \{a, abcd\}$ が入力として与えられたとする. この場合, ab と a , ab と $abcd$, acd と a , acd と $abcd$ の全ての組み合わせの中から, 最も長い共通部分列・部分文字列の長さを求める. この中で最長となる共通部分列は, acd と $abcd$ の組み合わせで acd という共通部分列である. したがって, 最長共通部分列問題の解は 3 となる. また, 最長共通部分文字列は, ab と $abcd$ の組み合わせで, ab という共通部分文字列である. したがって最長共通部分文字列の解は 2 となる.

2.3 非線形テキスト

非線形テキスト (Hypertext) とは, 文字列集合を扱うことができるグラフ構造であり, 1 つのノードが 1 つの文字, エッジが文字と文字のつながりを表す.

非線形テキストによって, 文字列集合における LCSseq 問題や LCSstr 問題を解くための手法が確立されている [7]. この手法では, 動的計画法を用いることにより, 2 つの文字列の集合に対して $O(\|F\| \|G\|)$ 時間で LCSseq 問題や LCSstr 問題を解くことができる. ここで, $\|F\|, \|G\|$ は, 2 つの非線形テキストのエッジの数である.

しかし, この手法では, $\|F\|, \|G\|$ を 2 つの非線形テキストのノード数としたとき, メモリを $O(\|F\| \|G\|)$ 空間使用するため, ノード数が多くなる大規模な文字列集合を入力とした場合, データがメモリに収まらない可能性がある.

† 立命館大学大学院情報理工学研究科, Graduate School of Information Science and Engineering, Ritsumeikan University

‡ 立命館大学大学院理工学研究科, Graduate School of Science and Engineering, Ritsumeikan University

2.4 SeqBDD

SeqBDD (Sequence Binary Decision Diagram) は、文字列集合を効率よく扱うことができるデータ構造である。本稿では、SeqBDD は、閉路の無い有向グラフであり、1文字を含む円を節点、1を四角で囲んだものを終端節点、実線の矢印を1枝、破線の矢印を0枝、として図3のように表記する。SeqBDDの各パスは、一つの文字列を表す。この文字列は、最上位の節点から終端節点までのパスの内、1枝をもつ節点の文字を連結したものになる。SeqBDDは、文字列集合の共通な接頭辞と接尾辞を共有することでコンパクトに表現される。さらに、演算キャッシュの機能により、効率よく文字列集合同士の集合演算が行われる[8]。

SeqBDDの例を図3に示す。図3(a)のSeqBDDで表現である P のルート節点は、 $\{ab, abc, ac, bc\}$ の文字列集合を表す。また、 P の部分グラフもSeqBDDを表すため、それぞれの節点が文字列集合を表す。図3(b)のブロック矢印のパスは、文字列 ac を表す。このパスは、 a の1枝、 b の0枝、 c の1枝と辿り、終端節点に到達する。途中で辿った1枝を持つ節点の文字である a と c をつなぎ合わせることで、 ac という文字列を得ることができる。

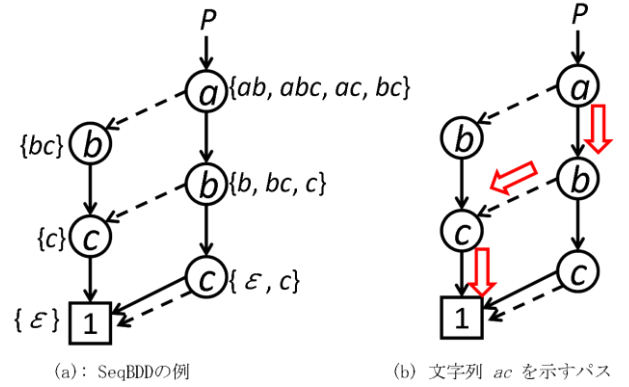


図3: SeqBDDの例

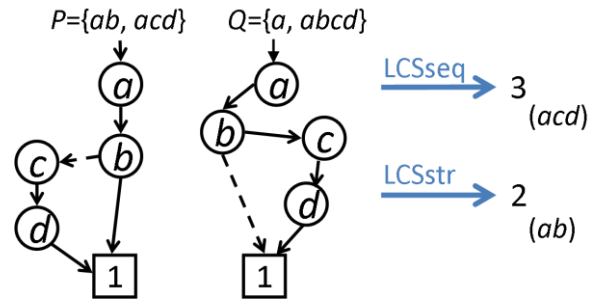


図4: SeqBDDにおける最長共通部分列・部分文字列問題の例

3. SeqBDDにおけるLCSseq・LCSstr問題

3.1 問題定義

SeqBDDにおける最長共通部分列・部分文字列問題は、SeqBDDである P と Q を入力とし、文字列集合における最長共通部分列問題を解くものとする。

この問題の例を図4に示す。図4の例は、2.2節の例と同じものである。よって、LCSseqの長さは3、LCSstrの長さは2となる。

3.2 単純な演算キャッシュを用いた再帰探索

提案手法のSeqBDDにおける最長共通部分列問題では、再帰関数を用いて処理を行う。再帰関数では、二つのSeqBDDをそれぞれ1枝と0枝に再帰的に展開して、LCSseqやLCSstrの長さを求める。また、再帰探索においては、何回も同じ計算を行う可能性があるため、演算キャッシュを用いて、同じ計算を何回も行わないように工夫する。

二つのSeqBDDにおける最長共通部分列を求める疑似コードを図5に示す。この関数は、SeqBDDを再帰的に展開して、二つのSeqBDDのLCSseqの長さを戻り値として返す。 P か Q が終端節点であるならば、LCSseqの長さは0になるので、0を返す(1-2行目)。また、演算キャッシュに P と Q のペアの演算結果が保存されているならば、その値を戻り値として返す(3-4行目)。次に、一方のSeqBDDは節点を1つだけ読み進め、もう一方のSeqBDDは、節点を読み進めないようなSeqBDDの組み合わせについて、最長共通部分列を求める(8-12行目)。つまり、 P_0 と Q 、 P_1 と Q 、 P

```

Procedure: LCSseq
Input:  $P$ : SeqBDD,  $Q$ : SeqBDD

1: if  $P$  = 終端節点 または、 $Q$  = 終端節点 then
2:   return 0
3: else if  $P$  と  $Q$  のペアに対するキャッシュがある then
4:   return  $P$  と  $Q$  のペアに対するキャッシュデータ
5: end if
6:  $c_p \leftarrow P$  の先頭節点の文字
7:  $c_q \leftarrow Q$  の先頭節点の文字
8:  $P_1 \leftarrow P$  の1枝側の SeqBDD
9:  $P_0 \leftarrow P$  の0枝側の SeqBDD
10:  $Q_1 \leftarrow Q$  の1枝側の SeqBDD
11:  $Q_0 \leftarrow Q$  の0枝側の SeqBDD
12:  $maxLen = \max(\text{LCSseq}(P_0, Q), \text{LCSseq}(P, Q_0),$ 
     $\text{LCSseq}(P_1, Q), \text{LCSseq}(P, Q_1));$ 
13: if  $c_p = c_q$  then
14:    $maxLen = \max(maxLen, \text{LCSseq}(P_1, Q_1) + 1)$ 
15: end if
16:  $P$  と  $Q$  のペアのキャッシュに  $maxLen$  を保存する
17: return  $maxLen$ 

```

図5: SeqBDDで最長共通部分列を求めるLCSseq

と Q_0 、 P と Q_1 の4つの組み合わせについて再帰呼び出しを行う。もし、二つのSeqBDDそれぞれの先頭の節点の文字が一致する場合は、この文字を先頭にもつ1枝側のSeqBDDのLCSseqの長さに1を加算した値が解の候補とな

Procedure: LCSstr
Input: P : SeqBDD, Q : SeqBDD
Return: res : P と Q の先頭一致文字数
 ans : P と Q の最長共通部分文字列

```

1: if  $P$  = 終端節点 または,  $Q$  = 終端節点 then
2:   return (0, 0)
3: else if  $P$  と  $Q$  のペアに対するキャッシュがある then
4:   return  $P$  と  $Q$  のペアに対するキャッシュデータ
5: end if
6:  $c_p \leftarrow P$  の先頭節点の文字
7:  $c_q \leftarrow Q$  の先頭節点の文字
8:  $P_1 \leftarrow P$  の 1-枝側の SeqBDD
9:  $P_0 \leftarrow P$  の 0-枝側の SeqBDD
10:  $Q_1 \leftarrow Q$  の 1-枝側の SeqBDD
11:  $Q_0 \leftarrow Q$  の 0-枝側の SeqBDD
12:  $res \leftarrow 0$ ;  $ans \leftarrow 0$ 
13: ( $res\_tmp$ ,  $ans\_tmp$ )  $\leftarrow$  LCSstr( $P_0$ ,  $Q$ )
14:  $res \leftarrow \max(res, res\_tmp)$ ;  $ans \leftarrow \max(ans, ans\_tmp)$ 
15: ( $res\_tmp$ ,  $ans\_tmp$ )  $\leftarrow$  LCSstr( $P_1$ ,  $Q$ )
16:  $res \leftarrow \max(res, res\_tmp)$ ;  $ans \leftarrow \max(ans, ans\_tmp)$ 
17: ( $res\_tmp$ ,  $ans\_tmp$ )  $\leftarrow$  LCSstr( $P$ ,  $Q_0$ )
18:  $ans \leftarrow \max(ans, ans\_tmp)$ 
19: ( $res\_tmp$ ,  $ans\_tmp$ )  $\leftarrow$  LCSstr( $P$ ,  $Q_1$ )
20:  $ans \leftarrow \max(ans, ans\_tmp)$ 
21: if  $c_p = c_q$  then
22:   ( $res\_tmp$ ,  $ans\_tmp$ )  $\leftarrow$  LCSseq( $P_1$ ,  $Q_1$ )
23:    $res \leftarrow \max(res, res\_tmp + 1)$ 
24:    $ans \leftarrow \max(ans, ans\_tmp, res)$ 
24: end if
25:  $P$  と  $Q$  のペアのキャッシュに( $res$ ,  $ans$ )を保存する
26: return ( $res$ ,  $ans$ )

```

図 6: SeqBDD で最長共通部分文字列を求める LCSstr

る。そのため、 P_1 と Q_1 のペアに対して再帰関数を呼び出す (13-14 行目)。これらの再帰呼び出しの結果の内、最大の値を戻り値とすることで、2 つの SeqBDD の LCSseq の長さを求めることができる。最後に、演算キャッシュに P と Q の演算結果として $maxLen$ を保存しておく (16 行目)。これにより、再び同じ再帰演算が呼び出されたとき、計算を行わずに、演算キャッシュの演算結果を利用することで演算を高速化する (3-4 行目)。

これとほぼ同じ再帰の呼び出し方で、図 6 のように LCSstr 問題を解くことができる。また、演算に必要なキャッシュ情報は全て保持し続けているため、同じ再帰演算を 2 回以上呼び出すことがない。そのため、 $\Theta(|P||Q|)$ の計算時間で動作する。なお、 $|P|$ と $|Q|$ は、それぞれ P と Q の節点数を表す。

LCSseq 問題や LCSstr 問題を高速に解くために、演算キャッシュを用いて同じ再帰演算を再び行わないようにしている。この演算キャッシュを実現する単純な手法として、図 7 のような二次元配列を用いたキャッシュ手法が考えられる。この配列は、各行が P の 1 つの節点に対応し、各列が Q の 1 つの節点に対応する。 P 上の節点を p_i ($0 \leq i < |P|$)、

	[0]	[1]	...	[Q -1]	
[0]	0	1	...	1	∞ : 未確定の値 0以上: 計算済みの結果
[1]	1	2	...	∞	
...	...	...	...	...	
[P -1]	∞	∞	...	∞	

図 7: 単純な演算キャッシュを表現する二次元配列

Q 上の節点を q_j ($0 \leq j < |Q|$) とする。このとき、行 i 列 j に格納されている値は、 p_i を先頭に持つ SeqBDD と q_j を先頭にもつ SeqBDD の再帰演算の結果を表す。

この二次元配列の演算キャッシュを用いると、演算キャッシュ情報の取得や書込を $O(1)$ で行うことができるため、高速に動作する。しかし、この演算キャッシュの配列は、 $O(|P||Q|)$ のメモリ量を必要とするため、入力 of SeqBDD の節点数が多くなると、データがメモリに収まらなくなる。

このメモリ使用量は、非線形テキストにおける計算のメモリ使用量とほぼ同じであるため、今回はこの単純な演算キャッシュ手法と 4.5-6 節で示す提案手法とで比較を行う。

3.3 再帰の呼び出し回数の検証

単純な演算キャッシュを用いた手法では、二次元配列を静的に確保するため、演算が終了するまで、 $O(|P||Q|)$ の大きさの演算キャッシュ情報をメモリ上に保持し続けていた。しかし、ある節点のペアの再帰演算について、それ以降、同じ節点のペアで演算が呼び出されないならば、全ての再帰処理が終了していなかったとしても、その演算に対応するキャッシュ情報は削除することができる。したがって、ある節点のペアの演算回数を事前に知ることができれば、最後の演算が呼び出されるタイミングが分かり、全体の処理の終了を待つことなく、演算キャッシュ情報を削除することができる。

本手法では、節点 p と q をそれぞれ先頭とする SeqBDD の再帰演算の呼び出し回数を考慮することによって、不要になったキャッシュ情報から順に削除していく。 p と q をそれぞれ先頭の節点とする SeqBDD に対する再帰の呼び出し回数は、 p と q の節点につながる枝の情報と節点の文字情報から、求めることができる。

再帰の全ての呼び出しパターンを示したものが図 8 である。図 8 の (a)-(c) では、それぞれの左側の図が SeqBDD である P の部分グラフ、右側の図が SeqBDD である Q の部分グラフを表す。ここでは、 P の中のある節点 p と Q の中のある節点 q のペアに対して、その再帰が何回呼び出されるかということを考える。

図 8 (a) は、図 5 の 12 行目の再帰によって呼び出されるパターンを表している。この再帰は、 Q の SeqBDD の節点を固定して、 P の SeqBDD の 1-枝か 0-枝を辿る再帰のパターンである。この場合の再帰呼び出し回数は、 p の入次数に等しくなる。例えば、図 8 (a) の場合、太い矢印のように、 p と q のペアの再帰が、 p の入次数である 4 回呼び出される。

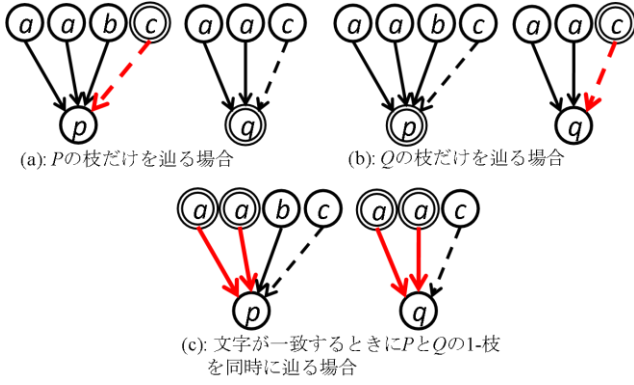


図 8: p と q のペアに対する再帰の呼び出しパターン

図 8 (b)も、図 5 の 12 行目の再帰によって呼び出されるパターンを表している。この再帰は、 P の SeqBDD の節点を固定して、 Q の SeqBDD の 1-枝か 0-枝を辿る再帰のパターンである。この場合、 q の入次数の数だけ再帰が呼び出される。例えば、図 7 (b)の場合、 p と q のペアの再帰が q の入次数である 3 回、呼び出される。

図 8 (c)は、図 5 の 14 行目の再帰によって呼び出されるパターンを表している。これは、 P の節点の文字と Q の節点の文字が一致する場合に、 P と Q の 1-枝を同時に辿る再帰のパターンである。この場合、節点 p と節点 q に 1-枝がつながる節点の文字を見て、再帰の呼び出し回数を求める。回数を求めるときは、節点の文字の種類毎に計算を行う。1-枝が節点 x を指し文字 c を持つ節点の数を $\text{indegree1}(x, c)$ とするとき、このパターンに対する再帰の呼び出し回数を式 (1)に示す。

$$\sum_{c \in \text{全ての文字}} (\text{indegree1}(p, c) \times \text{indegree1}(q, c)) \quad (1)$$

例えば、図 8 (c)の例では、式 (1)より、4 回再帰が呼び出される。これは、文字 a を持つ節点から節点 p と q に伸びる 1-枝の数をかけあわせた値である。

以上のことから、 p と q に対する再帰の呼び出し回数は、式 (2)で求めることができる。この式において、 $\text{indegree}(x)$ は節点 x の入次数を表す。

$$\begin{aligned} & \text{indegree}(p) + \text{indegree}(q) \\ & + \sum_{c \in \text{全ての文字}} (\text{indegree1}(p, c) \times \text{indegree1}(q, c)) \quad (2) \end{aligned}$$

3.4 メモリ使用量削減のための演算キャッシュ手法

提案手法では、図 9 に示すようなデータ構造を用いる。この構造は、ハッシュテーブルと、ハッシュテーブルからチェーンする演算キャッシュ情報によって構成される。チェーンする演算キャッシュ情報には、 p , q , r , rem の 4 つの数値を持たせている。 p は P の節点、 q はもう一つの Q の節点、 r は p と q をそれぞれ先頭にもつ SeqBDD の再帰演算の結果、 rem は、 p と q の節点のペアに対する再帰の残

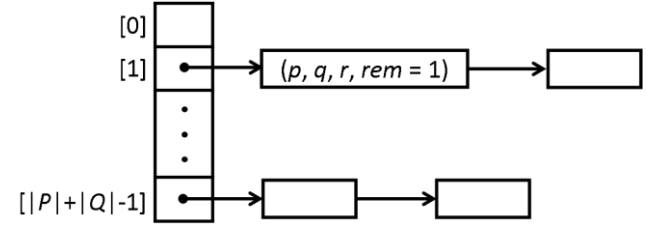


図 9: メモリ使用量削減のための演算キャッシュモデル

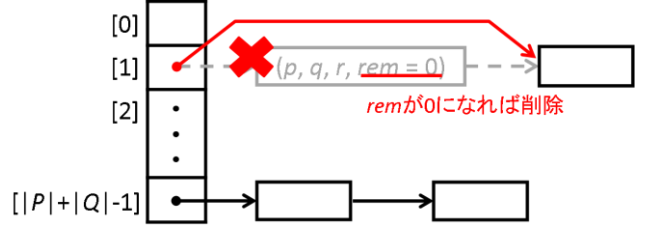


図 10: 演算キャッシュからのデータの削除

表 1: 実験環境

OS	Vine Linux 6.0
CPU	Intel® Core™ i5
メモリ	4GB
コンパイラ	gcc/g++ version 4.4.5 -O2 オプション

り呼び出し回数を表す。

演算キャッシュに新しいキャッシュ情報を挿入するときは、どの節点とどの節点の再帰の結果が何であるかという情報と、式 (2)により、何回この節点のペアに対する再帰が行われるかを計算した結果を挿入する。

演算キャッシュがヒットしたときは、そのヒットした情報の rem をデクリメントする。デクリメントして、もし rem が 0 になったのであれば、このキャッシュ情報は、二度と呼び出されないことがわかるため、演算キャッシュからこの情報を削除して、図 10 のように演算キャッシュの修正を行う。

本節で示した演算キャッシュ手法を用いることにより、LCSseq 問題や LCSstr 問題を最良の場合 $O(|P|+|Q|)$ のメモリ使用量で解くことができる。

4. 実験と結果

4.1 実験概要

文字列の個数が 100 個から 1000 個の間、文字の種類数が 26 個、文字列の長さが 50 である 2 つの文字列集合を入力とし、それらを SeqBDD として構築した。この 2 つの SeqBDD を入力として、LCSseq 問題を解くための最大メモリ使用量と実行時間を単純な演算キャッシュ手法と提案した演算キャッシュ手法とで比較を行った。実験は、表 1 のような環境で行った。

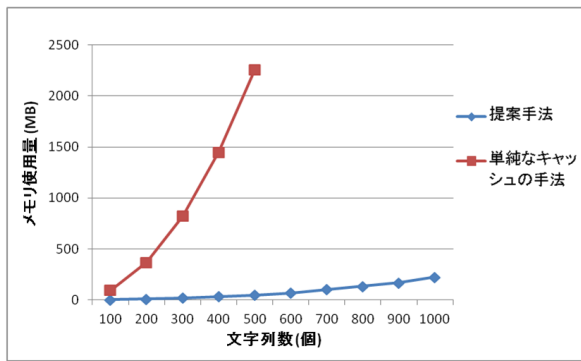


図 11: 文字列数を変化させたときのメモリ使用量比較

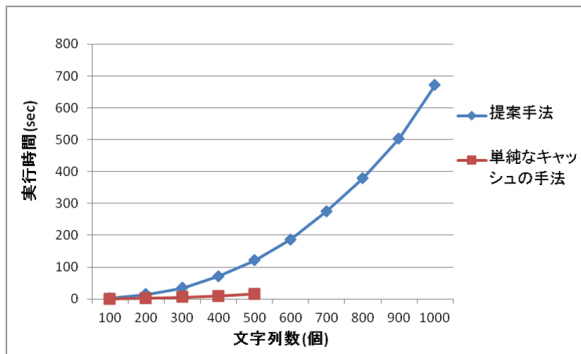


図 12: 文字列数を変化させたときの実行時間比較

4.2 実験結果

実験の結果を図 11, 図 12 に示す. 図 11 は, 入力された文字列の長さに対するメモリ使用量を示している. 図 11 から, 単純な演算キャッシュの手法では, 文字列の個数が増えるにつれ, メモリ使用量が増加していることがわかる. 文字列の個数が 500 個より大きくなると, メモリに収まらずに処理を行えなかった. 一方, 提案手法を用いると, メモリ使用量が大幅に増加することなく, 大規模な文字列集合を入力とした場合でも, 計算を行うことができた.

図 12 は, 入力の文字列の個数に対する実行時間を示している. 図 11 から, 提案手法の実行時間は, 単純な演算キャッシュ手法の実行時間より, 増加量が多いことがわかる. これは, 単純な演算キャッシュ手法では, キャッシュにアクセスするのに $O(1)$ であるのに対し, 提案手法では, キャッシュにアクセスするのにチェーンを辿らなければいけないため遅くなっていると考えられる.

5. まとめと今後の課題

本稿では, 大規模な文字列集合の最長共通部分列・部分文字列問題を解くために, メモリ使用量を削減して問題を解くアルゴリズムを提案した. 提案手法では, SeqBDD を入力として, 再帰関数と演算キャッシュを用いて問題を解く手法を提案した. 演算キャッシュには, 何度も呼び出される演算のキャッシュ情報だけを

保持するようにすることにより, メモリ使用量の削減を行うことができる. ある演算が何度も呼び出されるかを確認するために, 再帰演算の呼び出し回数を求める方法を示した. 実験の結果, 提案手法では, メモリ使用量を削減することがわかった. その一方で, 提案手法は, 単純なキャッシュ手法よりも実行に時間がかかった.

今後は, 結果として最長共通部分列・部分文字列の長さを出力するだけでなく, 最長共通部分列・部分文字列を表す文字列集合を効率よく出力できるような手法を提案する予定である.

参考文献

- [1] D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. MIT Press, Cambridge, MA, 2001.
- [2] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2006.
- [3] E. Loekito, J Bailey, and J. Pei. A binary decision diagram based approach for mining frequent subsequences. *Knowledge and Information Systems: An International Journal*, 24(2):235–268, August 2010.
- [4] 伝住周平, 有村博紀, 湊真一. 系列二分決定グラフを用いた部分文字列索引の構築. 第2 回データ工学と情報マネジメントに関するフォーラム (DEIM 2010), (E3-4), February 2010.
- [5] Hiroshi Aoki, Shigeru Yamashita, and Shin ichi Minato. An efficient algorithm for constructing a sequence binary decision diagram representing a set of reversed sequences. *Granular Computing (GrC), 2011 IEEE International Conference on*, pages 54–59, November 2011.
- [6] David W. Mount. *Bioinformatics: Sequence and Genome Analysis*. Cold Spring Harbor Laboratory, 2004.
- [7] 下平浩二, 稲永俊介, 坂内英夫, 竹田正幸. 非線形テキストにおける最長共通部分文字列・部分列アルゴリズム. 電子情報通信学会技術研究報告. COMP, コンピューテーション, 111(25):9-16, May 2011.
- [8] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35:677–691, 1986.