数理解析研究所講究録 第2088巻 2018年 27-35

ウォークからの最大次数2の最小グラフ推論

Inferring a minimum graph of bounded degree 2 from a walk

成定 真太郎 Shintaro Narisada
ディプタラマ ヘンリアン Diptarama Hendrian
吉仲 亮 Ryo Yoshinaka
篠原 歩 Ayumi Shinohara
東北大学大学院情報科学研究科
Graduate School of Information Sciences, Tohoku University.

1 導入

辺集合 V，頂点集合 E とラベル $c\colon E \to \Sigma$ からなるラベル付き無向グラフ $G = (V, E, c)$ を考える．G の隣接した頂点をたどる列 $v_0, e_1, v_1, \ldots, v_{n-1}, e_n, v_n$ (ここに $v_i \in V$, $e_i = (v_{i-1}, v_i) \in E$ である) を G 上の**ウォーク**といい，そこで遭遇する辺ラベルの列 $c(e_1), c(e_2), \ldots, c(e_n)$ をそのウォークの**出力**と呼ぶ．**ウォークからの最小グラフ推論問題**とは，入力として与えられた文字列 w に対して，出力が w となるウォークを持つグラフで，頂点数が最小のものを見つける問題である．図 1 に一例を示す．

Aslam ら [1] は，この問題を定式化し，最大次数を 2 に制限したグラフ族に対して，ウォークからの最小グラフ推論問題に対する多項式時間アルゴリズムを提案した（表 1）．最大次数 2 の連結グラフは，閉路グラフ (次数 2 の頂点のみからなる) と線形鎖 (次数 1 の頂点も含む) に大別される．線形鎖において，始点と終点が共に次数 1 の頂点であるウォークを**両端ウォーク**といい，その制限がないものを**一般ウォーク**と言う．Raghavan [4] は，ウォークからの閉路グラフの推論問題と，一般ウォークからの線形鎖の推論問題が，ともに両端ウォークからの線形鎖の推論問題に $O(n)$ 時間で帰着できることを示し，またこの問題に対する $O(n \log n)$ 時間アルゴリズムを与えた．また，任意の定数 $k \geq 3$ に対し，次数 k のグラフを推論する問題が NP 困難であることを証明した．一方，Maruyama ら [3] は，ウォークからの最小木推論問題に対して，$O(n)$ 時間アルゴリズムを提案した．また，複数の部分ウォークからの最小木および最小線形鎖推論問題が NP 完全であることを示した．

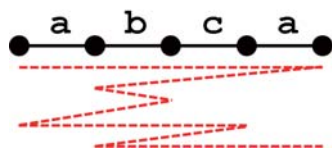

図 1: 出力が abcaacbbbaabccbbca となるウォークをもつ最小の線形鎖

表 1: 次数 2 の最小グラフ推論問題における計算量

手法	次数 2 の連結グラフ		
	線形鎖		閉路グラフ
	両端	一般	
[1]	$O(n^3)$	$O(n^3)$	$O(n^5)$
[4]	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
提案	$O(n)$	$O(n)$	$O(n)$

本論文は，前述の両端ウォークからの最小線形鎖推論問題に対して，Manacher の極大回文を求めるアルゴリズムを応用した $O(n)$ 時間・領域アルゴリ

ズムを提案する．このことにより，ウォークからの最大次数 2 の最小グラフ推論問題がすべて $O(n)$ 時間で解けることが導かれる（表 1 最下段）．

2 準備

大きさ σ のアルファベットを Σ とし，Σ^* の要素を文字列という．文字列 w の長さを $|w|$ と表し，空文字列を ε と書く．空でない文字列の集合を $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ と書く．$1 \leq i \leq |w|$ に対して，$w[i]$ を w の i 番目の文字とする．文字列 $w = xyz$ に対して，x，y，z をそれぞれ w の**接頭辞**，**部分文字列**，**接尾辞**と呼ぶ．$1 \leq i \leq j \leq |w|$ に対して，i で始まり j で終わる w の部分文字列を $w[i..j]$ と書く．文字列 x の逆文字列を $x^R = x[|x|] \cdots x[1]$ と表す．文字列 p が**偶回文**であるとは，文字列 $x \in \Sigma^*$ に対して $p = xx^R$ となるときをいい，この長さ $r = |x|$ を**半径**と呼ぶ．本稿では，偶回文のみを扱うので，以下，偶回文のことを単に回文と呼ぶ．

w を長さ n の任意の文字列とする．w の部分文字列 $w[b..e]$ が回文であるとき，$p = w[b..e]$ を w の**回文部分文字列**と呼び，位置 $i = \frac{b+e}{2}$ を p の**中心**と呼ぶ．特に，$w[b-1] \neq w[e+1]$，$b = 1$，または $e = n$ を満たすとき，p を i を中心とする**極大回文**と呼ぶ．中心 c，半径 r の極大回文を 2 つ組 (c, r) で表す．文字列 w 中の極大回文の集合を $P(w)$ と書く．$P_j(w) = \{(c, r) \in P(w) \mid c < j\}$ と書く．また，$P_{i,j}(w) = \{(c, r) \in P(w) \mid i < c < j\}$ と書く．

文字列 z が **Z 形**であるとは，空でない文字列 $x \in \Sigma^+$ に対して，$z = xx^Rx$ となるときをいう．$z = w[b..e]$ が Z 形であるとき，位置 $c = \frac{2b+e-2}{3}$ を**ピボット**，$r = \frac{e-b+1}{3}$ を**半径**と呼び，2 つ組 $\langle c, r\rangle$ で表す．文字列 w 中の Z 形の集合を $Z(w)$ と書く．

例 1 $w = \mathtt{ab}\underline{\mathtt{abccbaabc}}\mathtt{a}$ において，Z形 abccbaabc はピボット 5，半径 3 である．すなわち，$\langle 5, 3\rangle$ である．

$Z_j(w) = \{\langle c, r\rangle \in Z(w) \mid c < j\}$ とし，$Z_{i,j}(w) = \{\langle c, r\rangle \in Z(w) \mid i < c < j$ または $i < c + r < j\}$ とする．

R を Σ^* 上の二項関係とし，$(x, y) \in R$ を $x\ R\ y$ と書く．R^* を R の**反射推移閉包**とする．文字列 x が**既約**であるとは，$x\ R\ y$ となる文字列 $y \neq x$ が存在しないときをいう．R が**合流性**をもつとは，$\forall w, x, y \in \Sigma^*, wRx \wedge wRy \Rightarrow \exists z \in \Sigma^*, xR^*z \wedge yR^*z$ であるときをいう．合流性をもつ関係 R と文字列 $x \in \Sigma^*$ に対し，xR^*y となる既約な文字列 y を x の**正規形**と呼び，$\hat{x}$ と書く．Σ^* 上の二項関係 $\to$ を次式で定義し，この書き換え処理を，($\to$ の関係の下)Z 形を**縮約する**という：

$$\to = \{(pxx^Rxq, pxq) \mid x \in \Sigma^+, p, q \in \Sigma^*\}$$

定理 1 ([1]) 二項関係 $\to$ は合流性を持つ．

定理 1 より，任意の順番で ($\to$ の関係の下) 文字列 x を有限回数縮約することで，正規形 $\hat{x}$ が求まる．このとき，$Z(\hat{x}) = \emptyset$ である．

定理 2 ([1]) 文字列 w が両端ウォークとなる最小の線形鎖は二項関係 $\to$ の正規形 $\hat{w}$ である．

例 2 $w = \mathtt{cbaaaabccbaabba}$ とすると，最小線形鎖 $\hat{w} = \mathtt{cba}$ は次のように求められる．下線部は Z 形を表す．$\mathtt{cb}\underline{\mathtt{aaa}}\mathtt{abccbaabba} \to \mathtt{cbaabcc}\underline{\mathtt{baabba}} \to \underline{\mathtt{cbaabccba}} \to \mathtt{cba}$．なお $\to$ の合流性により，上記と異なる書き換え順でも最終的には $\hat{w}$ が得られる．

3 両端ウォークからの最小線形鎖推論

定理 2 より，両端ウォークからの最小線形鎖推論は，入力文字列 w に対して，以下の処理を行えばよいことが分かる．

(1) w の部分文字列に Z 形 xx^Rx が見つかった場合，xx^Rx の接尾辞 x^Rx を w から取り除く．

(2) w の部分文字列中に Z 形が 1 つも存在しなくなるまで処理 (1) を繰り返す．

長さ n の入力文字列 w に対して，w 中に出現しない文字 \$ と # を用いて，$w[0] = \$$，$w[n+1] = \#$ と約束する．本問題を解くアイデアとして，Z 形 xx^Rx は，2 つの回文 xx^R と x^Rx がそれぞれ半分ずつ重なったものと考えられることから，Manacher のアルゴリズム [2] を利用して Z 形の検出および縮約を行う．文字列 w のすべての位置 i に対し，i を中心とする極大回文の半径を格納する配列 $Pals$ を考える．

$$Pals[i] = \{r \mid w[i-r+1..i+r] \text{ が } i \text{ を中心とする } w \text{ の極大回文 }\}.$$

すなわち，$P(w) = \{(i, Pals[i]) \mid 1 \le i \le n\}$ となるように $Pals$ を求めたい．$Pals$ は，Manacher のアルゴリズムによって $O(n)$ 時間・領域で求めることができる．以下に Manacher のアルゴリズムを再帰的に記述したアルゴリズム RecManacher を示す．

3.1 RecManacher アルゴリズム

RecManacher アルゴリズムを Algorithm 1 に示す．また，Algorithm 2 – 4 に Algorithm 1 によって呼び出される関数 $manacher(c,i)$，$extension(c,i)$，$transfer(c,i)$ をそれぞれ示す．関数 $extension(c,i)$ は，中心が c である極大回文の半径 r を求める関数である．ただし，事前条件として，中心を c とする半径 $i-c-1$ 以上の回文が存在することが分かっている．また，$transfer(c,i)$ は，$c' = c+1, ..., c+r$ の順に c' を中心とする極大回文の半径を求める関数である．ただし，途中で極大回文 (c', r') が $c+r < c'+r'$ となる場合，c をシフトし $manacher(c', i)$ を再帰的に呼び出す．

本アルゴリズムの大まかな流れを以下に記す．$i-1$ までの $Pals$ が全て求まっていると仮定すると，Algorithm 1 の $RecManacher(w)$ において，文字列 w のある位置 $i-1$ について，Algorithm 2 の $manacher(i-1,i)$ を呼び出す．$manacher(i-1,i)$ は，Algorithm 3 の $extension(i-1,i)$ を呼び出し，$c = i-1$ を中心とする極大回文の半径 r を求める．その後，Algorithm 4 の $transfer(c, c+r+1)$ において，$c' = c+1, ..., c+r$ の順に，極大回文 $Pals[c]$ の左側の情報を用いて $Pals[c']$ を求める．その際，$(c', r') \in P(w)$ に対して，$c'+r' > c+r$ となった場合，すなわち，c' を中心とする極大回文が，c を中心とする極大回文の右端を越える場合，Algorithm 2 の関数 $manacher(,を)$ 引数 $(c', c+r+1)$ によって再帰的に呼び出す．Algorithm 1 において，$manacher(i-1,i)$ の処理が終了して返り値 i_2 が戻されたとき，すなわち，$i_2 = manacher(i-1,i)$ となるときを考える．関係 コ を コ $= \{((c,r),(c',r')) \mid (c,r) \in P(w), (c',r') \in P(w), c < c' \le c+r < c'+r'\}$ と定義し，この コ を用いて，$RMostPal_w(c)$ を次に定義する (図 2)．

$$RMostPal_w(c) = \max\{c'+r'+1 \mid (c,r) \in P(w), (c,r) \text{ コ}^* (c',r')\}.$$

Algorithm 1 において，$i_2 = manacher(i-1,i)$ が求まったとき，$i_2 = RMostPal_w(i-1)$ となる．この

Algorithm 1: $RecManacher(w)$

```
1 Pals[0] = 0;
2 i = 2;
3 while w[i-1] ≠ # do
4 |   i = manacher(i-1, i);
5 |   i = i+1;
6 return Pals;
```

Algorithm 2: $manacher(c,i)$

```
1 i = extension(c, i);
2 i = transfer(c, i);
3 return i;
```

Algorithm 3: $extension(c,i)$

```
1 r = i-c-1;  j = c-r;
2 while w[i] = w[j] do
3 |   r = r+1;  j = j-1;  i = i+1;
4 Pals[c] = r;
5 return i;
```

Algorithm 4: *transfer*(c, i)

```
1  r = i − c − 1;  d = 1;
2  while d ≤ r do
3  |  c' = c + d;  r_l = Pals[c − d];
4  |  if r_l < r − d then
5  |  |  Pals[c'] = r_l;
6  |  |  d = d + 1;
7  |  else if r_l > r − d then
8  |  |  Pals[c'] = r − d;
9  |  |  d = d + 1;
10 |  else
11 |  |  if w[i] ≠ w[2 * c' − i + 1] then
12 |  |  |  Pals[c'] = r − d;
13 |  |  |  d = d + 1;
14 |  |  else return manacher(c', i);
15 return i;
```

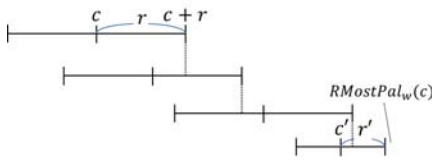

図 2: $RMostPal_w(c)$ と極大回文との関係

とき，0 から $i_2 - 1$ までの極大回文が全て求まっている．次に，$i = i_2 + 1$ に対して，$manacher(i-1, i)$ を行う．このとき，$i - 1$ までの $Pals$ が全て求まっている．これを，$w[i-1] = \#$ となるまで繰り返すことで，最終的に $P(w)$ を得る．RecManacher アルゴリズムは，長さ n の文字列 w に対して $O(n)$ 時間・領域で $Pals$ を計算できる．

3.2 RecManacher を用いた Z 形の検出

観察 1 (極大回文と Z 形との関係) $\langle i, r\rangle$ が w の Z 形である $\iff$ $c_1 = i, c_2 = i + r$ に対する w の極大回文 (c_1, r_1) と (c_2, r_2) について，$c_2 \le c_1 + r_1$ かつ

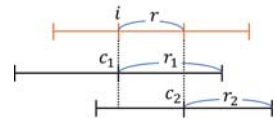

図 3: Z 形 $\langle i, r\rangle$ と 2 つの極大回文 (c_1, r_1)，(c_2, r_2) との関係

$c_2 - r_2 \le c_1$ である．

図 3 に観察 1 を図示する．観察 1 を用いて，Algorithm 1 に基づいて Z 形を検出することを考える．配列 $Pals$ の値を用いて，観察 1 における式 $c_2 \le c_1 + r_1$ かつ $c_2 - r_2 \le c_1$ を表すと，$c_2 \le c_1 + Pals[c_1]$ かつ $c_2 - Pals[c_2] \le c_1$ となる．Algorithm 5 は，Algorithm 3 の **while** ループの内部に Z 形 $\langle c_1, c_2 - c_1\rangle$ を検出し，これを縮約する命令を挿入したものである．Algorithm 5 の 4 行目における式 $Pals[j-1] \ge r$ が真となる場合に，Z 形 $w[j-r..i]$ が検出される．Algorithm 5 の 4 行目における式 $Pals[j-1] \ge r$ は，

Algorithm 5: *zextension*(c, i)

```
1  r = i − c − 1;  j = c − r;
2  while w[i] = w[j] do
3  |  r = r + 1;
4  |  if Pals[j − 1] ≥ r then
5  |  |  delete(j, i);
6  |  |  return j − 1;
7  |  j = j − 1;  i = i + 1;
8  Pals[c] = r;
9  return i;
```

式 $c_2 \le c_1 + Pals[c_1]$ かつ $c_2 - Pals[c_2] \le c_1$ において，$c_1 = j - 1$，$c_2 = c$ とおいた場合と同様である．これにより，w 中の一部の Z 形が検出可能となる．この関係をうまく利用し，すべての Z 形が検出できるようなアルゴリズムを考える．

Z 形を検出した後，Z 形の縮約を行うことを考える．すなわち，$w = w_0 \to w_1 \to \ldots \to w_{m-1} \to w_m = \hat{w}$ とする．Algorithm 5 の 5 行目に，$w_k[j..i]$ を削除す

る関数 $delete(j,i)$ を書く．Algorithm 5 の 4 行目においてZ形である $xx^Rx = w_k[j-r..i]$ が検出され，$delete(j,i)$ において接尾辞 $x^Rx = w[j..i]$ が削除されたとする．その後，文字列 $w_{k+1} = w_k[1..j-1]w_k[i+1...|w_k|]$ において，アルゴリズム中の変数 $i = j-1$ として，RecManacher の処理を再開したい．しかし，縮約後の文字列 w_{k+1} に対する配列 $Pals_{w_{k+1}}$ に関して，次の観察を得る．

観察 2 (Z形の縮約後の文字列の性質) いま，$c = i-r-1$ について，$c-1$ までの $Pals$ が全て求まっていると仮定する．$j = c-r$ において，Z 形：$xx^Rx = w_k[j-r..i]$ が検出され，Z 形の接尾辞 $x^Rx = w_k[j..i]$ が削除されたとすると，削除後の文字列 $w_{k+1} = w_k[1..j-1]w_k[i+1...|w_k|]$ において，$P_{j-1}(w_k) = \{(l, Pals[l]) \mid 0 \leq l < j-1\}$ が成立するとは限らない．

このようになる原因は，Z形 $xx^Rx = w_k[j-r..i]$ の検出による Z 形の接尾辞 $x^Rx = w_k[j..i]$ の削除によって，w_k では $j-1$ で終端する極大回文が，w_{k+1} においてさらに伸長する可能性があるためである．すなわち，w_k における $transfer$ にて既に計算した $j-1$ までの $Pals$ を多重に更新しなくてはならない場合がある．よって，RecManacher のアルゴリズムをそのまま両端ウォークからの最小線形鎖推論に用いると，時間計算量は $O(n^2)$ となる．

3.3 LAmanacher アルゴリズム

両端ウォークからの最小線形鎖推論を $O(n)$ で計算する最適アルゴリズム LAmanacher を示す．本アルゴリズムは，観察 2 で示した，配列 $Pals$ の要素を多重に更新しないように，$Pals$ の要素を計算する前に Z 形を先読みして検出し，縮約を行うアルゴリズムである．LAmanacher は，w 中のZ形の終端位置が小さい順にZ形の縮約を行い，最終的に $\hat{w}$ を得る．アルゴリズム LAmanacher を Algorithm 6 に示す．また，Algorithm 6 によって呼び出される関数 $lamanacher(c,i)$，$laextension(c,i)$，$latransfer(c,i_0,i')$，$skip(c,i)$ を，Algorithm 7 – 10 にそれぞれ示す．関数 $laextension(c,i)$ は，中心が c である極大回文の半径を求める関数である．事前条件として，中心を c とする半径 $i-c-1$ 以上の回文が存在することが分かっている．ただし，ある $r' > 0$ に対して，$\langle c-r', r' \rangle \in Z(w_k)$ が検出される場合は，Z 形の縮約を行う．また，$latransfer(c,i_0,i')$ は，$i_0 \leq c' < i'$ を満たす全ての c' について，c' を中心とする極大回文の半径を求める関数である．これらの関数を再帰的に呼び出すことで，$\hat{w}$ を求める．本アルゴリズムの流れを記す．$Pals$ のすべての要素が -1 となるように初期化する．ある i に対して，$P_{i-1}(w_k) = \{(j, Pals[j]) \mid 0 \leq j < i-1\}$ であり，$Z_{i-1}(w_k) = \emptyset$ であると仮定する．Algorithm 6 の関数 $LAmanacher(w)$ において，文字列 w のある位置 $i-1$ について，Algorithm 7 の関数 $lamanacher(i-1,i)$ を呼び出し，$lamanacher(i-1,i)$

Algorithm 6: $LAmanacher(w)$

```
1 Pals[0] = 0;
2 Let Stack be an empty stack;
3 i = 2;
4 while w_k[i-1] ≠ # do
5 |   i = lamanacher(i-1, i);
6 |   i = i + 1;
7 |   Stack.clear()
8 return ŵ;
```

Algorithm 7: $lamanacher(c,i)$

```
1 i_0 = i;
2 i = laextension(c, i);
3 if i < c then return i;
4 i' = latransfer(c, i_0, i);
5 if c < i' < i then
6 |   Pals[i'] = Pals[2 * c - i'];
7 |   i = i' + 1;
8 |   goto 2;
9 return i';
```

Algorithm 8: *laextension*(c, i)

```
1  r = i − c − 1;  j = c − r;
2  while w_k[i] = w_k[j] do
3  |  r = r + 1;
4  |  if Pals[j − 1] ≥ r then
5  |  |  delete(j, i);
6  |  |  return j − 1;
7  |  j = j − 1;  i = i + 1;
8  |  Pals[i] = Pals[j − 1];
9  Pals[c] = r;
10 return i;
```

Algorithm 9: *latransfer*(c, i_0, i')

```
1  r = i' − c − 1;  d = r;
2  while i_0 ≤ c + d do
3  |  c' = c + d;
4  |  if Pals[c'] < r − d then d = d − 1;
5  |  else if Pals[c'] > r − d then
6  |  |  Pals[c'] = r − d;
7  |  |  d = d − 1;
8  |  else
9  |  |  i_2, p = skip(c', c + r + 1);
10 |  |  r' = i_2 − c' − 1;
11 |  |  if w_k[i_2] ≠ w_k[c' − r'] then
12 |  |  |  Pals[c'] = r';
13 |  |  |  d = d − max{1, r'};
14 |  |  |  if p ≥ 1 then Stack.push(c');
15 |  |  else
16 |  |  |  i'_2 = lamanacher(c', i');
17 |  |  |  if i'_2 > i' then Stack.push(c');
   |  |  |    i' = i'_2;  d = d − max{1, i' − c' − 1};
18 |  |  |  else return i'_2;
19 return i';
```

は Algorithm 8 の関数 *laextension*$(i-1, i)$ を呼び出す．*laextension*$(i-1, i)$ は，$c = i-1$ を中心とする極

Algorithm 10: *skip*(c, i)

```
1  p = 0;
2  while Stack is not empty do
3  |  d = Stack.top() − c;
4  |  if Pals[c − d] = Pals[Stack.top()] then
5  |  |  i = Stack.top() + Pals[Stack.top()] + 1;
6  |  |  Stack.pop();
7  |  |  p = p + 1;
8  |  else
9  |  |  r = min{Pals[c−d], Pals[Stack.top()]};
10 |  |  i = Stack.top() + r + 1;
11 |  |  break;
12 return i, p;
```

大回文 $Pals[c]$ を RecManacher と同様にして求める．また，このとき，8行目の通りに $Pals[i]$ に $Pals[j-1]$ を代入する．これは，*latransfer* 関数で $Pals$ を転写して求める前に，仮の $Pals$ の値を計算する処理であり，**仮転写**と呼ぶ．仮転写された $Pals$ の値は正しくない場合があるが，本アルゴリズムにおいては正当性を示すことができる．4行目の式において，Z 形である $xx^Rx = w_k[j-r..i]$ が検出された場合を考える．ただし，$|x| = r$ とする．このとき，$w_k[j..i]$ を削除し，$w_{k+1} = w_k[1..j-1]w_k[i+1..|w_k|]$ とし，$j-1$ を次の更新開始位置として **return** する．

Z 形の縮約が起こらなかった場合，中心 c の極大回文の半径 r が求まる．Algorithm 7 の *lamanacher*$(i-1, i)$ の子関数である Algorithm 9 の関数 *latransfer*$(c, i_0 = i, i' = c+r+1)$ において，$i_0 \le c' \le c+r$ の範囲で，**降順**に，極大回文 $Pals[c]$ の左腕の情報を用いて $Pals[c']$ を求める．降順に転写を行い，再帰呼び出しを行うことが，実質的に Z 形の**先読み**を行うことになる．c' に対して，i' を次のように定義する．

$$i' = \max(\{j \mid j = RMostPal_{w_k}(c'') \text{ for some } c' < c'' \le c+r\}) \quad (1)$$

2行目の **while** ループ内において，常に i' が求まっ

ている．また，関数 *skip* により，次を満たす i_2 が返却されると仮定する．

$$i_2 = \min(\{c' + r' + 1 \mid (c', r') \in P(w_k)\} \cup \{i'\}) \tag{2}$$

各 c' と $(c', r') \in P(w_k)$ を満たす r' に対して，$c' + r' + 1 \geq i'$ となった場合を考える．このとき，16行目のように，Algorithm 7 の *lamanacher* を引数 (c', i') によって再帰的に呼び出す．

lamanacher(c', i') の返り値 i'_2 が，$i'_2 > i'$ を満たすときを考える．アルゴリズムにおいて，関係 $\to = \{(pxx^Rxq, pxq) \mid x \in \Sigma^+, p, q \in \Sigma^*\}$ によってZ形が縮約され，$w_k \to^* w_{k'}$ となる $w_{k'}$ が得られたとする．このとき，次の **while** ループ変数 $c' - r'$ に対しても，式 1 の i' は成立したままであり，$Z_{c'-r',i'}(w_{k'}) = \emptyset$ かつ $P_{c'-r',i'}(w_{k'}) = \{(j, Pals[j]) \mid c' - r' < j < i'\}$ が言える．また，i'_2 が，$c \leq i'_2 \leq i'$ を満たすときは，子もしくは孫の *lamanacher* において，Z形の縮約が行われた場合である．この場合は，Algorithm 7 の *lamanacher* における 5 行目から 8 行目に記述する処理に従う．

Algorithm 6 の *LAmanacher* において，*lamanacher*$(i-1, i)$ の処理が終了して返り値 i' が戻されたときを考える．このとき，$w_{k'}$ に対して，$i' = RMostPal_{w_{k'}}(i-1)$ もしくは $i' = i$ となる．このような i' に対して，$Z_{i'}(w_{k'}) = \emptyset$ かつ $P_{i'}(w_{k'}) = \{(j, Pals[j]) \mid 0 \leq j < i'\}$ が成り立つ．これを，$w_{k'}[i-1] = \#$ となるまで繰り返すことで，$\hat{w}$ が求まる．LAmanacher に関して，Z形について次の補題が成立する．

補題 1 (極大回文中の Z 形の存在条件 (1))
$(c, r) \in P(w_k)$ とする．$Z_c(w_k) = \emptyset \Rightarrow$ 任意の c_2, r_2 について，$\langle c_2, r_2 \rangle \in Z(w_k)$ ならば $c_2 + 2r_2 > c + r$ である．

補題 1 より，Z形 $\langle c_2, r_2 \rangle$ が存在するならば，必ず $c + r$ より大きい位置で終端するということが分かる．次の補題によって，すべてのZ形がLAmanacher の関数 *laextenstion* において検出されることが導かれる．

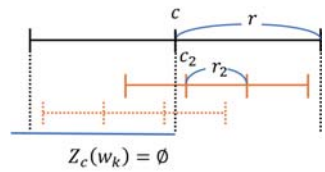

図 4: 補題 1 に矛盾する極大回文と Z 形との関係

補題 2 (極大回文中の Z 形の存在条件 (2))
$(c, r) \in P(w_k)$ とする．$c < c' \leq c + r$ を満たす任意の c' について，$(c', r') \in P(w_k)$ とし，式 *1* における i' を考える．$Z_c(w_k) = \emptyset$ かつ $Z_{c',i'}(w_k) = \emptyset \Rightarrow$ 任意の c_2, r_2 について，$\langle c_2, r_2 \rangle \in Z(w_k)$ ならば $c_2 + 2r_2 \geq i'$ である．

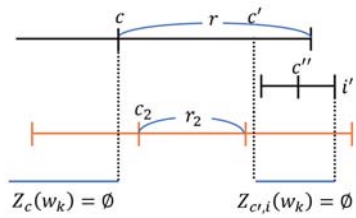

図 5: 補題 2 における，極大回文 (c, r) とループ変数 c', i' および Z 形 $\langle c_2, r_2 \rangle$ との関係

また，仮転写された値に関して次の性質がある．

補題 3 (仮転写された値の性質) $P_c(w_k) = \{(j, Pals[j]) \mid 0 \leq j < c\}$ と仮定する．ある c, i について，$laextension(c, i)$ によって，極大回文 (c, r) が求まり，$0 < d \leq r$ の範囲に対して，中心が $c + d$ の仮転写された値を $\tilde{r}_{c+d}$ と書き，中心が $c + d$ の極大回文の値を r_{c+d} と書く．任意の $c + d$ に対して次が成立する．

(1) $d + \tilde{r}_{c+d} < r \Rightarrow \tilde{r}_{c+d} = r_{c+d}$,

(2) $\tilde{r}_{c+d} \neq r_{c+d} \Rightarrow d + \tilde{r}_{c+d} \geq r$ かつ $d + r_{c+d} \geq r$.

補題 3 を用いて，LAmanacher の仮転写の正当性が次の補題で示される．

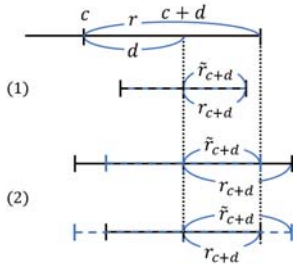

図 6: 補題 3 における，仮転写された値 $\tilde{r}_{c+d}$ と正しい極大回文の値 r_{c+d} との関係

補題 4 (仮転写の正当性) $P_c(w_k) = \{(j, Pals[j]) \mid 0 \leq j < c\}$ かつ $Z_c(w_m) = \emptyset$ と仮定する．このとき，$lamanacher(c, c+1)$ に対して $laextension(c, c+1)$ が呼ばれ，極大回文 (c, r) が求まり，$c+1, ..., c+r$ の範囲に対して，$Pals$ が仮転写されたとする．$latransfer(c, c+1, c+r+1)$ において，$c < c' < c+r+1$ の範囲の c' を考え，c' に対して式 *1* における i' を考える．また，関数 $skip$ により，式 *2* における i_2 が返却されると仮定する．すると，次が成り立つ．

- $lamanacher(c, c+1)$ が $lamanacher(c', i')$ を再帰呼び出しした際，$lamanacher(c', i')$ が参照する任意の仮転写された値は，参照した時点で正しい．
- $lamanacher(c, c+1)$ の終了時，$i' = RMostPal_{w'_k}(c)$ に対して，$P_{i'}(w_k) = \{(j, Pals[j]) \mid 0 \leq j < i'\}$ が成立する．

さらに，$LAmanacher$ における Z 形の縮約の正当性を以下の補題によって示すことができる．

補題 5 (Z 形の縮約の正当性) $(c, r) \in P(w_k)$ とし，$Z_{c-r,c}(w_k) = \emptyset$ とする．ある c, i に対して $laextension(c, i)$ が呼ばれ，極大回文 $Pals[c] = r$ が求まったとする．$i < c' \leq c+r$ の範囲の任意の c' に対して，式 *1* における i' を考えると，次が成り立つ．

- ある r' について，$\langle c'-r', r' \rangle \in Z(w_k) \iff laextension(c', i')$ において $w_k[c'-r'..c'+r']$ が削除される．その後，*(1)*$c'-r' = c$ なら，ある $c_2 \leq c$ に対して，$laextension(c_2, c+1)$ がやり直される．また，*(2)*$c'-r' > c$ なら，$laextension(c, c'-r'+1)$ まで戻り，正当性を保ったままアルゴリズムが再開される．

また，関数 $skip$ に関して次の正当性が示される．

補題 6 ($skip$ の正当性) $(c, r) \in P(w_k)$ とし，$Z_{c-r,c}(w_k) = \emptyset$ とする．$c < c' \leq c+r$ の範囲の任意の c' に対して，式 *1* における i' を考える．いま，$i' \neq c+r+1$ であるとき，c_q および r_q を用いて $i' = c_q + r_q + 1$ と書き，$1 \leq l \leq q-1$ に対して c_l, r_l を以下に再帰的に定義する．

$$(c_l, r_l) \in P(w_k),$$
$$c_l < c_{l+1} \leq c_l + r_l < c_{l+1} + r_{l+1}, c_1 = c'$$

このような列 $c_1, c_2, ..., c_q$ に対して，グローバル

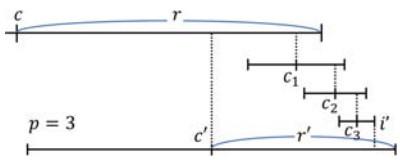

図 7 補題 6 における，極大回文と関数 $skip$ の返り値 $\imath_2, p$ との関係の例　図の関係においては，$\imath_2 = i'$ および $p = 3$ を返し，このとき，Stack は空である．

な $Stack$ に底から順に $c_q, c_{q-1}, \ , c_1$ が格納されていると仮定する．このとき，文字列 w_k に対する $skip(c', c+r+1)$ は，式 *2* における $\imath_2$ および次を満たす p を $O(p)$ 時間で返す．

p は $skip(c', c+r+1)$ において pop された回数．

ただし p は $c_p + r_p < c' + r'$ を満たす最大の数

関数 $skip$ と Stack により，先読みによって $Pals$ を計算した範囲を多重に計算することがなくなる．

以上の補題によって，$LAmanacher$ の正当性　計算量に関して次の定理を得る．

定理 3 ($LAmanacher$ の計算量) $LAmanacher(w)$ は $\hat{w}$ を $O(n)$ 時間　領域で計算する．

参考文献

[1] J. A. Aslam and R. L. Rivest. Inferring graphs from walks. In *Colt 1990*, pp. 359–370, 1990.

[2] G. K. Manacher. A new linear-time on-line algorithm for finding the smallest initial palindrome of a string. *J. ACM*, 22(3):346–351, 1975.

[3] O. Maruyama and S. Miyano. Inferring a tree from walks. *Theoretical Computer Science*, 161(1):289–300, 1996.

[4] V. Raghavan. Bounded degree graph inference from walks. *Journal of Computer and System Sciences*, 49(1):108–132, 1994.