社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

FPGA 上に実現した二つの近似文字列マッチングアルゴリズムの比較

清水　敬介† 　中原　啓貴† 　笹尾　勤† 　松浦　宗寛†

† 九州工業大学 情報工学部 〒 820–8502 福岡県飯塚市大字川津 680–4

あらまし　テキスト中に現れるパターンの出現位置を求める問題を厳密文字列マッチングという. 一方で, パターンを編集したものをテキスト中に求める問題を近似文字列マッチングという. 本論文では, 3 つの編集 (削除, 挿入, 置換) を考える. 編集の度合いを定量的に表したものを編集距離という. 近似文字列マッチングの多くのアルゴリズムは動的計画法に基づく. 本論文では, 近似文字列マッチングを動的計画法で求める Naive 法と LL 法について述べ, 最小編集距離を計算する回路の面積を見積もる. Altera 社 FPGA 上に二つの実現法を実装し, 面積と動作周波数を求めた. 面積見積り値が実装結果と一致してることを示す.

A Comparison of Two Approximate String Matching Algorithms Implemented on an FPGA

Keisuke SHIMIZU†, Hiroki NAKAHARA†, Tsutomu SASAO†, and Munehiro MATSUURA†

† Department of Computer Science and Electronics, Kyushu Institute of Technology
680–4, Kawazu, Iizuka, Fukuoka, 820–8502 Japan

Abstract　An approximate string matching finds the most similar pattern in the text. A dynamic programming is used for the approximate string matching. This paper considers two algorithms: Naive method and LL method. Also, it derives area complexities with respect to the pattern length. Our implementations on an Altera's FPGA agree with the derived area complexities.

1. はじめに

テキスト中に現れるパターンの出現位置を求める問題を厳密文字列マッチングという. 一方で, パターンを編集したものをテキスト中に求める問題を近似文字列マッチングという. 本論文では, 3 つの編集 (削除, 挿入, 置換) を考える. 編集の度合いを定量的に表したものを編集距離という. 近似文字列マッチングには様々な応用がある. 例えば, タイプミスやスペルミスを許容するテキスト検索, DNA 配列や RNA 配列間の類似度解析 (アライメント), 音声認識, 及びデータマイニング等が挙げられる.

近似文字列マッチングでは, パターンに削除・置換・挿入を行いながら, テキストと比較を行う手法が一般的である. 多くのアルゴリズムは動的計画法に基づく. このうち, **Smith-Waterman (SW) アルゴリズム**は最も有名である [9]. SW アルゴリズムはバイオ・インフォマティクスにおいて, DNA 配列や RNA 配列間の類似性を評価するために用いられている. ただし, ソフトウェアで実行すると計算時間がかかり, 実用的ではない.

近年, ハードウェアを用いて近似文字列マッチングを高速化する手法やその改良アルゴリズムが提案されている [7]. 改良アルゴリズムとしては Lipton らが, 提案したアルゴリズムがある [6]. 一方, バイオ・インフォマティクス用ハードウェアの実装事例は数多く報告されている. Hoang らは FPGA を用いた近似マッチングマシン 'SPLASH' [4], 及び 'SPLASH 2' [5] を実現した. Yamaguchi らは近似マッチングハードウェアを繰返し書き換え, 長いパターンをマッチングする手法を提案した [12]. Storaasli らは, 150 個の FPGA (Xilinx 社 Virtex-4) をスーパーコンピュータ (Cray XD1) に実装したシステムを用いて, 近似マッチングシステムを実現した. また, 近似マッチングのプロファイル解析を行った [10]. Buyukkurt らは高位合成ツールを用いた SW アルゴリズムの実装事例の報告を行った [2]. また, GPU を用いたバイオ・インフォマティクス用近似マッチングシステムも報告されている [15].

本論文では近似文字列マッチングを直接実現する手法 (**Naive 法**) とそれを改良した Lipton らが提案した手法 (**Lipton-Lopresti 法 (LL 法)**) [6] の最小編集距離を計算する回路の面積を見積もる. また, これらの手法を FPGA 上に実装し, 理論的解析の検証と, 実行速度の実験的な比較を行う.

第 2 章では近似文字列マッチングの定義を行う. 第 3 章では近似文字列マッチングを動的計画法で解く方法について述べる. 第 4 章では近似文字列マッチングの実現法について述べる. 第 5 章では近似文字列マッチングの実現法の比較を行いう. 第 6 章で本論文のまとめを行う.

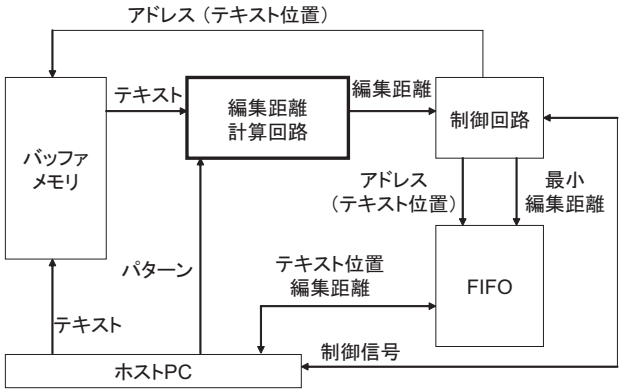

図 1　近似文字列マッチングを行うシステム.

2. 近似文字列マッチング

テキスト中に現れる**パターン**の出現位置を求める問題を厳密文字列マッチングといい, パターンを編集したものをテキスト中に求める問題を**近似文字列マッチング**という. 二つの文字列間の類似度を定量的に表したものを**編集距離** (edit distance) という [11]. 編集距離とは以下に示す**基本編集演算**に割当てた**編集スコア**の和と定義する.

（1） 挿入 (Insert)
（2） 削除 (Delete)
（3） 置換 (Substitute)

［例 2.1］ テキストを ACG とし, パターンを TGG とし, 編集距離を求める.

（1） テキスト'ACG' から'A' を削除し, 'CG' を得る.
（2） 'CG' から'C' を削除し, 'G' を得る.
（3） 'G' に'G' を挿入し, 'GG' を得る.
（4） 'GG' に'T' を挿入し, 'TGG' を得る. パターンと一致したので終了. (例終)

本論文では, 挿入と削除の編集スコアを 1 とする. 置換は, 挿入と削除を行うことであるから, 編集スコアを 2 とする.

［例 2.2］ 例 2.1 に示した'ACG' と'TGG' 編集距離は 4 である. (例終)

図 1 に近似文字列マッチングを行うシステムを示す. ホスト PC から, テキストとパターンを送る. バッファメモリからテキストを読み出し, 編集距離計算回路でテキストの一部とパターンの編集距離を計算する(注1). 制御回路は, 編集距離が最小の場合, FIFO に最小編集距離とテキスト位置を表すアドレスを格納する. テキストを 1 文字ずつずらしながら, これらの操作を行う. 全てのテキストのマッチングが終ったらホスト PC は最小編集距離とテキスト位置を FIFO から読み出し, 必要ならば編集パターンを求める.

後述するように, 近似文字列マッチングでは編集距離を求める計算が最も時間を要する. 本論文では, 高速で最小な編集距離を計算する回路を考える.

3. 編集距離の計算

2 つの文字列間の編集距離は動的計画法を用いて計算できる. Needleman-Wunsch (NW) アルゴリズム [8] はテキスト全体とパターンの編集距離の最小値を求める. Smith-Waterman (SW) アルゴリズムはテキストの一部とパターンの編集距離の最小値を求める. ここでは動的計画法を用いた二つの文字列間の編集距離の最小値を求めるアルゴリズムについて述べる.

(注1)：編集距離計算回路ではテキストの一部とパターンのローカルな最小編集距離を求める.

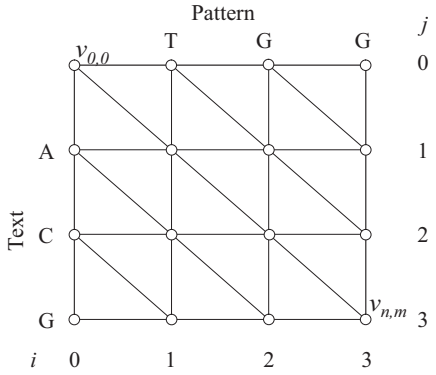

図 2　近似文字列マッチング・グラフの例.

パターンを $P=(p_1,\ldots,p_n)$, テキストを $T=(t_1,\ldots,t_m)$ とし, これらを各行各列にラベル付けした $(n+1)\times(m+1)$ 個の頂点を持つ**近似文字列マッチング・グラフ**を考える. 本論文では, 座標 (i,j) には頂点 $v_{i,j}$ が配置され, 左上の頂点を原点 $(0,0)$ とし, 右下の頂点 (n,m) に向かって座標 (i,j) が増加するものとする. また, $0\leqq i\leqq n-1, 0\leqq j\leqq m-1$ に対して, $v_{i,j}$ と $v_{i+1,j}$, $v_{i,j}$ と $v_{i,j+1}$ 間を結ぶ縦横方向の枝が存在し, $v_{i,j}$ と $v_{i+1,j+1}$ 間を結ぶ対角線方向の枝が存在する.

［例 3.3］ 図 2 にテキスト ACG, パターン TGG に対する近似文字列マッチング・グラフの例を示す. (例終)

削除の編集スコアを s_{del}, 挿入の編集スコアを s_{ins}, 置換の編集スコアを s_{sub} とする. 本論文では, $s_{del}=1$, $s_{ins}=1$, $s_{sub}=2$ とする. 各頂点 $v_{i,j}$ にサブパターン $P'=(p_1,p_2,\ldots,p_i)$ とサブテキスト $T'=(t_1,t_2,\ldots,t_j)$ に対する編集距離を記憶させる. 各頂点が持つ編集距離のことを**頂点スコア**と定義する. 各頂点 $v_{i,j}$ に対する編集距離の最小値は以下に示す再帰式により計算できる.

$$v_{i,j}=min\begin{cases} v_{i-1,j-1}+\begin{cases} p_i=t_j\text{のとき} & 0\\ p_i\neq t_j\text{のとき} & s_{sub}\end{cases}\\ v_{i-1,j}+s_{ins}\\ v_{i,j-1}+s_{del}\end{cases}\qquad(1)$$

再帰式 (1) を頂点 $v_{0,0}$ から頂点 $v_{n,m}$ へと再帰的に適用すると, 最小な編集距離を計算できる. 以下に, 最小編集距離を計算するアルゴリズムを示す.

［アルゴリズム 3.1］ テキスト T とパターン P が与えられ, それぞれの長さを m,n とする.

1: $v_{i,0}\leftarrow i,(i=0,1,\ldots,n)$, $v_{0,j}\leftarrow j,(j=0,1,\ldots,m)$ とする.
2: **for** $j\leftarrow 1$ **until** $j\leqq m+n-1$ **begin**
3: **for** $i\leftarrow 1$ **until** $i\leqq n$ **begin**
4: **if** $0<j-i+1\leqq m$ ならば,
式 (1) を用いて $v_{i,j-i+1}$ を求める.
5: $i\leftarrow i+1$ とする.
6: **end**
7: $j\leftarrow j+1$ とする.
8: **end**
9: $v_{n,m}$ を編集距離とし, 停止.

パターン長 n はテキスト長 m よりもはるかに短いと仮定する. 例えば, バイオインフォマティクスのアライメントでは, $n=10^3$ 程度に対して, $m=10^9$ 程度である. ソフトウェア上

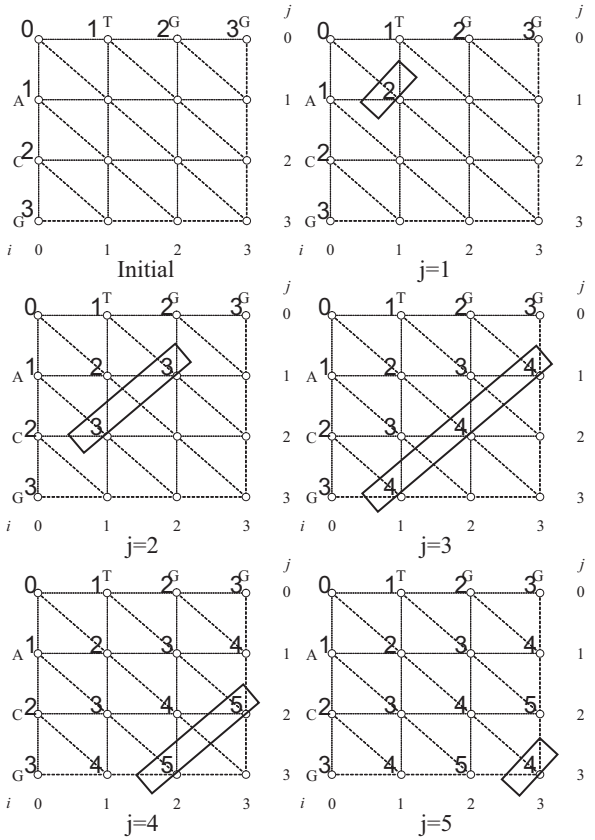

図 3　アルゴリズム 3.1 の例.

での編集距離の計算時間複雑度は $O(mn)$ である. 本論文ではハードウェア (並列プロセッサ) 上でアルゴリズム 3.1 を実現する.

［例 3.4］ 図 3 に図 2 に示した近似文字列マッチング・グラフに対してアルゴリズム 3.1 を適用した例を示す. 図 3 において, j はアルゴリズム 3.1 中の j を表す. アルゴリズム 3.1 より, 編集距離 4 を得る. これは例題 2.2 の結果と一致している. (例終)

4. 最小編集距離計算回路の実現法

4.1 Naive 法

再帰式 (1) を直接用いて最小編集距離を計算するアルゴリズム 3.1 を **Naive 法**とする. 例 3.4 より, 図 3 に示した太枠内の頂点スコアはそれぞれ独立に計算できる.

近似文字列マッチング・グラフの各列の計算を行うために**プロセッシング・エレメント (PE)** を用いる [3]. PE はクロック毎に対応する頂点の値を計算し, 出力する (図 4 において太枠で囲まれた範囲). 図 4 において, t は時刻を表す. ここで, 頂点 $v_{i,j}$ を PE_i で演算するために必要なデータの依存関係を考える. 再帰式 (1) より, $v_{i,j}$ を計算するには $v_{i,j-1}$, $v_{i-1,j}$, $v_{i-1,j-1}$ が必要である. ここで, $v_{i,j-1}$ は時刻 $t-1$ において, PE_i が出力する値であるから, PE_i の出力をフィードバックして参照すればよい. $v_{i-1,j}$ は時刻 $t-1$ において, PE_{i-1} が出力する値であるから, PE_{i-1} の出力を受け取ればよい. $v_{i-1,j-1}$ の値は時刻 $t-2$ において, PE_{i-1} が出力する値である. 従って, レジスタを 1 段挟み, 時刻を 1 つ遅らせてから受け取ればよい. 図 4 の下部に示す PE を直列に接続した回路は, 1 クロックで近似文字列マッチング・グラフの対角線 (図 4 において, 太枠で囲まれた範囲) を一度に評価する. つまり, 並列プロセッサを用いるとアルゴリズム 3.1 に示した 3 行目から 6 行目までのループを

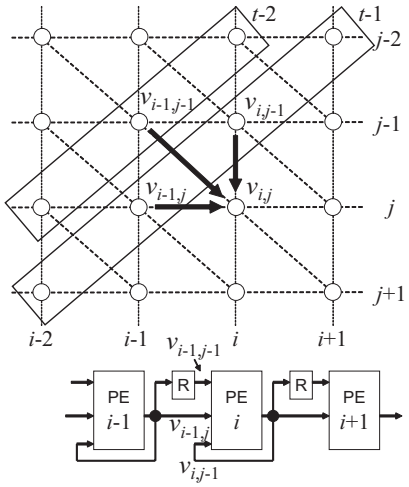

図 4　Navie 法の PE 間のデータ依存.

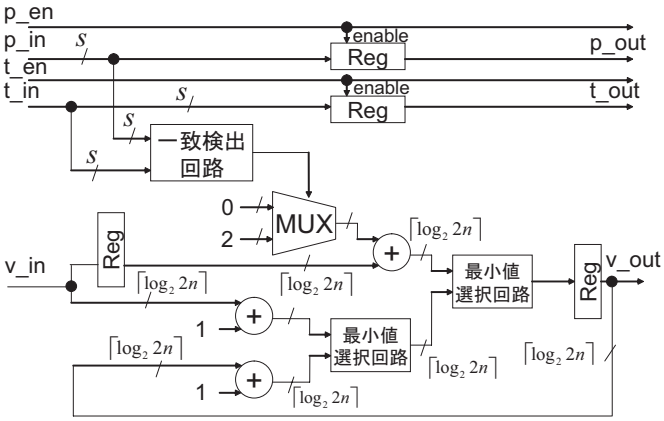

図 5　Naive 法の PE の構造.

並列に計算できる. 従って, この回路を用いた場合, アルゴリズム 3.1 の計算時間複雑度は $O(m)$ となる.

図 5 に Naive 法の PE の構造を示す. 図 5 において, s はテキストとパターン 1 文字のビット数を表し, n はパターン長を表す. 再帰式 (1) を直接実行するため, 図 4 の下部に示した回路にテキスト (t_in) とパターン (p_in) を入力し, 一致検出回路を通して, 置換の編集スコアを加算するか選択する. 同時に, 各頂点のスコアに対してそれぞれの編集スコアを加算し, 最小値選択回路を用いて最小編集スコアを求める.

Naive 法の PE の回路面積を見積り, パターン長 n に対する面積を見積もる. Naive 法の PE では, 一致検出回路, 定数加算器, 及び最小値選択回路が大部分を占める. 提案手法を FPGA 上に実現するので, FPGA の構成要素である Look-Up Table (LUT) 数で面積を見積もる. 本論文では, k を LUT の入力数, c を LUT の段数, s をテキストとパターン 1 文字のビット数とし, LUT の出力数は 1 とする.

図 6 に一致検出回路の構成を示す. 2 つの文字を上位 $2i$ ビット $(1 \leqq i)$ ずつ同時に比較する. 一致又は不一致の情報を次段の LUT に伝播していけばよい. LUT 数を c_{equal} とすると, 関係

$$\begin{cases} k+(k-2)(c_{equal}-1) \leqq 2s, & k \text{ が偶数のとき} \\ k-1+(k-1)(c_{equal}-1) \leqq 2s, & k \text{ が奇数のとき} \end{cases} \quad (2)$$

が成立する. 一致検出回路に必要な LUT 数を c_{equal} とすると, 関係

$$c_{equal} \leqq \begin{cases} \frac{2s-2}{k-2}, & k \text{ が偶数のとき} \\ \frac{2s}{k-1}, & k \text{ が奇数のとき} \end{cases} \quad (3)$$

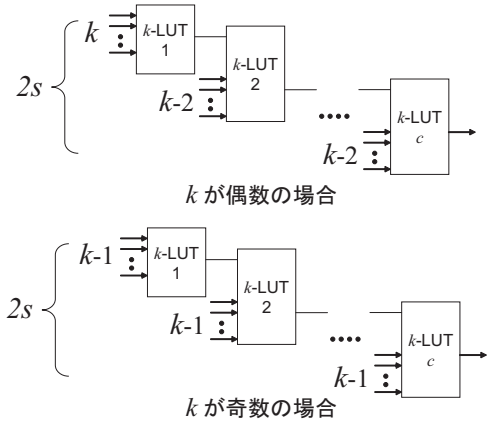

図 6　一致検出回路.

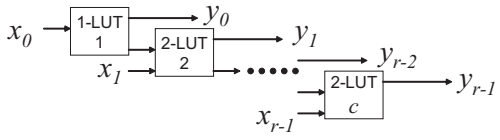

図 7　定数加算器.

を得る.

図 7 に定数加算器の構成を示す. 各ビット毎に加算を行い, 加算結果を出力する 2 入力 LUT と桁上がりを出力する 2 入力 LUT が必要である. 最終段のみ桁上がりが不要であるから, 入力ビット数を r とすると, 定数加算器に必要な LUT 数 c_{adr} は

$$c_{adr} = 2r - 1 \tag{4}$$

となる.

図 8 に, 2 つの入力 $X = (x_0, x_1, \ldots, x_{r-1})$, $Y = (y_0, y_1, \ldots, y_{r-1})$ を比較し, 最小となる入力を出力する最小値選択回路と LUT を用いた MUX の構成を示す. 最小値選択回路は, 最小となる入力を決定するための比較器 (Comparator) と入力を選択する選択回路 (MUX) で構成できる. 比較器は図 6 に示した一致回路の LUT の中身を書換えれば実現できる. また, 最小となる値 $z_i (i = 0, 1, \ldots, r-1)$ を選択する回路は図 8 の右に示すように, 最小となる入力を各ビット毎に選択する r 個の 2 入力 LUT で実現できる. 選択回路に必要な LUT 数を c_{MUX} とすると $c_{MUX} = r$ である. 最小値選択回路に必要な LUT 数を c_{min} とするとき, 関係

$$c_{min} = c_{equal} + c_{MUX} \tag{5}$$

を得る.

図 5 より, 一致検出回路は 1 個, 定数加算器は 3 個, 最小値選択回路は 2 個である. 従って, 式 (3), (4), 及び (5) より, Naive 法の PE を実現するために必要な LUT の総数 c_{Naive} は

$$c_{Naive} = c_{equal} + 3 \times c_{adr} + 2 \times c_{min} \tag{6}$$

と見積もれる.

4.2 LL　　法

再帰式 (1) において, 削除の編集スコアを $s_{del} = 1$, 挿入の編集スコアを $s_{ins} = 1$, 置換の編集スコアを $s_{sub} = 2$ とするとき, Lipton らは近似文字列マッチング・グラフの頂点スコアに関する有用な性質を示した [6].

[定理 4.1]　近似文字列マッチング・グラフのある頂点における上下左右の頂点スコアの差分は常に 1 となり, 右下, 及び左上の頂点スコアの差分は常に 0 又は 2 のどちらかである.

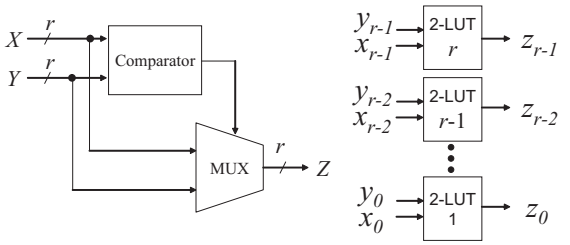

図 8　最小値選択回路と LUT を用いた選択回路の構成.

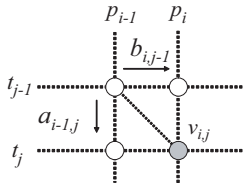

図 9　頂点間の差分を表す変数.

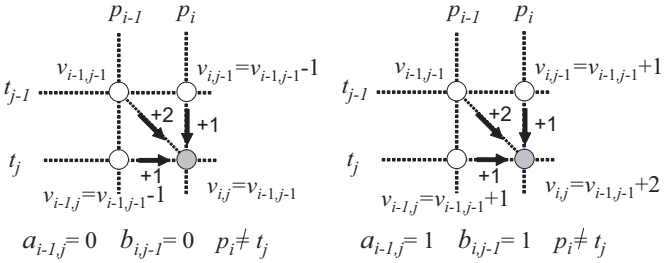

図 10　頂点スコアの関係.

Lipton らは定理 4.1 を用いて, 再帰式 (1) を再定義し, 図 5 に示した PE を簡単にする手法を提案した. 以降, Lipton らの手法を **Lipton-Lopresti 法 (LL 法)** とする.

縦方向の頂点間, すなわち $v_{i,j}$, $v_{i,j-1}$ 間の頂点スコアの大小関係を表す変数を $a_{i,j} \in \{0, 1\}$ とし, 横方向の頂点間, すなわち $v_{i,j}$, $v_{i-1,j}$ 間の頂点スコアの大小関係を表す変数を $b_{i,j} \in \{0, 1\}$ とする (図 9). 定理 4.1 より, ある頂点と上下左右の頂点間の頂点スコアの差分は常に 1 となるので, $a_{i,j}$, $b_{i,j}$ をそれぞれ

$$a_{i,j} = \begin{cases} v_{i,j} = v_{i,j-1} - 1 \text{ のとき } 0 \\ v_{i,j} = v_{i,j-1} + 1 \text{ のとき } 1 \end{cases}$$

$$b_{i,j} = \begin{cases} v_{i,j} = v_{i-1,j} - 1 \text{ のとき } 0 \\ v_{i,j} = v_{i-1,j} + 1 \text{ のとき } 1 \end{cases} \tag{7}$$

と定義する. a は一つ上の頂点スコアよりも小さい場合は 0, 大きい場合は 1 になり, b は一つ左の頂点スコアよりも小さい場合は 0, 大きい場合は 1 になるとも言える.

$a_{i-1,j}$, $b_{i,j-1}$, p_i, t_j を用いて $v_{i-1,j-1}$ から $v_{i,j}$ を求める. 近似文字列マッチング・グラフの頂点 $v_{i,j}$ に関する再帰式 (1) より, $p_i = t_j$ のとき, $v_{i,j} = v_{i-1,j-1}$ となるのは明らか. よって, $p_i \neq t_j$ の場合を考える. $a_{i-1,j} = b_{i,j-1} = 0$ のとき (図 10 左),

$$v_{i-1,j} = v_{i,j-1} = v_{i-1,j-1} - 1$$

であるから, 再帰式 (1) より, $p_i \neq t_j$ のとき $v_{i,j}$ は各頂点スコアに編集スコアを加算し, その中から最小となる値を求めればよい. ここで, 編集スコアは $s_{del} = 1$, $s_{ins} = 1$ であるから

$$\begin{aligned} v_{i-1,j} + s_{del} &= v_{i-1,j-1} - 1 + s_{del} \\ &= v_{i-1,j-1} \end{aligned}$$

であり,

$$
\begin{aligned}
v_{i,j-1} + s_{ins} &= v_{i-1,j-1} - 1 + s_{ins} \\
&= v_{i-1,j-1}
\end{aligned}
$$

であるから, $v_{i,j} = v_{i-1,j-1}$ となる. $a_{i-1,j} = 0$ または $b_{i,j-1} = 0$ のとき, 同様に $v_{i,j} = v_{i-1,j-1}$ となる. $a_{i-1,j} = b_{i,j-1} = 1$ のとき (図 10 右),

$$
v_{i-1,j} = v_{i,j-1} = v_{i-1,j-1} + 1
$$

であり, 編集スコアは $s_{del} = 1$, $s_{ins} = 1$, $s_{sub} = 2$ であるから

$$
\begin{aligned}
v_{i-1,j} + s_{del} &= v_{i-1,j-1} + 1 + s_{del} \\
&= v_{i-1,j-1} + 2
\end{aligned}
$$

となり,

$$
\begin{aligned}
v_{i,j-1} + s_{ins} &= v_{i-1,j-1} + 1 + s_{ins} \\
&= v_{i-1,j-1} + 2
\end{aligned}
$$

となる. 一方,

$$
v_{i-1,j-1} + s_{sub} = v_{i-1,j-1} + 2
$$

であるから $v_{i,j} = v_{i-1,j-1} + 2$ となる. 以上の議論より変数 a, b を用いると, 近似文字列マッチング・グラフの頂点に関する再帰式 (1) は

$$
v_{i,j} = \begin{cases} p_i \neq t_j \text{のとき}, \begin{cases} a_{i-1,j} = b_{i,j-1} = 1 \text{ ならば } v_{i-1,j-1} + 2 \\ \text{上記以外 } v_{i-1,j-1} \end{cases} \\ p_i = t_j \text{のとき}, v_{i-1,j-1} \end{cases} \tag{8}
$$

と定義できる.

頂点スコアを再帰的に計算するために, $a_{i,j}$, $b_{i,j}$ を求める. 変数 $d \in \{0, 1\}$ を導入し, 再帰式 (8) の結果に応じて

$$
d = \begin{cases} v_{i,j} = v_{i-1,j-1} \text{のとき } 0 \\ v_{i,j} = v_{i-1,j-1} + 2 \text{ のとき } 1 \end{cases} \tag{9}
$$

とする. ここで, $d = 1$ となるのは図 10 右に示した場合, すなわち $a_{i-1,j} = b_{i,j-1} = 1$ となる場合のみである. 図 10 右に示すように, 明らかに $v_{i,j} = v_{i-1,j} + 1 = v_{i,j-1} + 1$ であるから, a, b の定義式 (7) より, $a_{i,j} = b_{i,j} = 1$ となる. 一方, $d = 0$ のとき, $v_{i-1,j-1} = v_{i,j}$ である. $a_{i,j}$ は $b_{i,j-1} = 1$ のとき (図 11 左), $v_{i,j-1} = v_{i-1,j-1} + 1$ であるから, 条件 $v_{i-1,j-1} = v_{i,j}$ を満たすには

$$
v_{i,j} = v_{i,j-1} - 1 \tag{10}
$$

でなければならない. 式 (10) を条件式 (7) に適用すると $a_{i,j} = 1$ となる. 逆に, $b_{i,j-1} = 0$ のとき (図 11 右) は $a_{i,j} = 0$ となる. 従って, 以上の議論より $a_{i,j}$ は

$$
a_{i,j} = d \vee \bar{b}_{i,j-1} \tag{11}
$$

で得られる. 同様に, $b_{i,j}$ は

$$
b_{i,j} = d \vee \bar{a}_{i-1,j} \tag{12}
$$

で得られる.

式 (8), (9), (11), (12) より, $p_i, t_j, a_{i-1,j}, b_{i,j-1}$ から $a_{i,j}, b_{i,j}$ が求まる. 図 12 に LL 法の PE 間のデータ転送を示す. LL 法では, Naive 法とは異なり, 各列の頂点スコアの差分を計算する PE を直列に接続し, 編集距離を求める. すなわち, 頂点間の

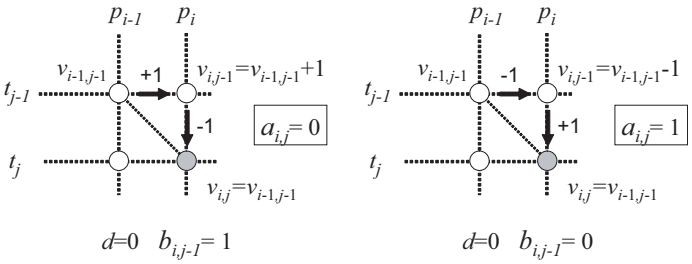

図 11 頂点スコアの差分の関係.

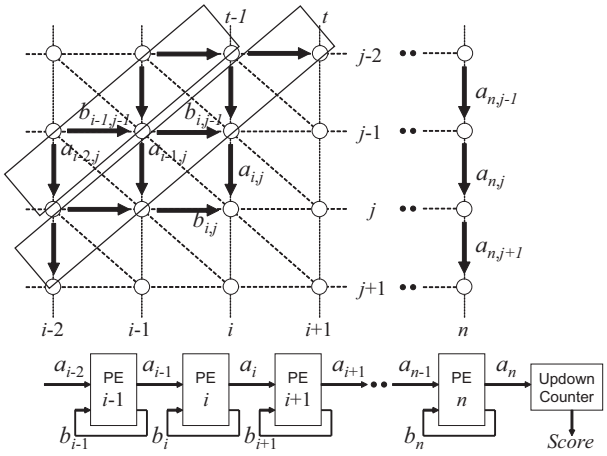

図 12 LL 法の PE 間のデータ転送.

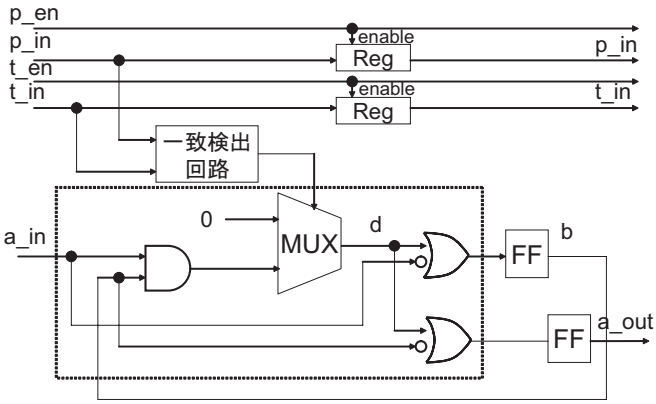

図 13 LL 法の PE の構成.

枝の重みに頂点スコアの差分を割り当て, 枝の重みを再帰的に計算することと等しい. 図 12 に示すように, 頂点間の横方向の差分を表す a を次の PE に転送する. 縦方向の差分を現す b はフィードバック参照する. 最終段の PE の出力は前の時刻の最終段の縦方向の差分しか出力されない. 従って, PE の他に初期値を n とするアップダウンカウンタを用いて編集距離を求める.

図 13 に LL 法の PE の構成を示す. LL 法では a, b, d が全て 1 ビットで表現できる. LL 法の PE は近似文字列マッチング・グラフの頂点間の差分のみを用いるため, PE のサイズはパターン長 n に依存しない. 図 13 より, LL 法の PE は一致検出回路と複数の論理ゲートで実現できる. 一致検出回路は, 式 (3) に示した c_{equal} 個の LUT で実現できる. また, 図 13 の点線で囲まれた部分は 2 個の 3 入力 1 出力 LUT で実現できる. LL 法の PE を実現するために必要な LUT の総数 c_{LL} は

$$
c_{LL} = c_{equal} + 2 \tag{13}
$$

と見積もれる.

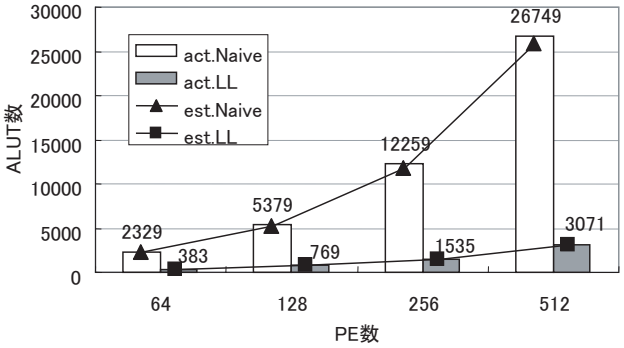

図 14　Stratix III の ALUT 数の比較.

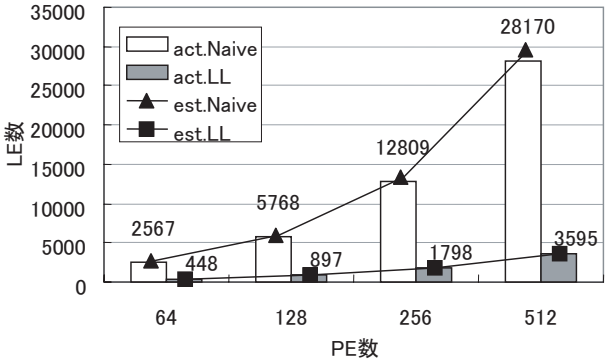

図 15　Cyclone III の LE 数の比較.

5. 近似文字列マッチングアルゴリズムの実装の比較

第 4 章では一般的な k 入力 1 出力 LUT を用いた場合の面積を見積もった. 実際の FPGA の LUT は多機能な付属回路を持つ. 本章では, Altera 社 [1] の 2 種の FPGA に対して面積見積りを行い, 実際に実装した結果と比較を行い, 見積り値を検証する. 次に, 見積り値の比較から, Naive 法と LL 法の違いを考察する.

5.1 Altera 社 Cyclone III の LE を用いた場合

Cyclone III では標準的な 4 入力 1 出力 LUT の他に, 全加算器を構成するために 3 入力 2 出力 LUT を構成できる. 3 入力 2 出力 LUT を用いれば, 式 (4) に示した r ビットの定数加算器を

$$c_{adr'} = r \tag{14}$$

個の LUT で実現できる. 従って, $k = 4$ とすれば Cyclone III を用いた場合の必要な LUT 数を推定できる.

5.2 Altera 社 Stratix III の ALUT を用いた場合

Stratix III は ALUT で構成されており, 様々な LUT を構成できる. 本論文では, 4 入力 2 出力 LUT と 6 入力 1 出力 LUT を構成し, 面積を見積もる. よって, 定数加算器は式 (14) で示した $c_{adr'}$ 個の LUT で実現できる. 従って, $k = 6$ とすれば Stratix III を用いた場合の必要な LUT 数を推定できる.

5.3 実験内容

Naive 法と LL 法を Altera 社 FPGA StratixIII と Cyclone III 上にパターン長 n を 64,128,256,512 と変化させ実装した. そして, 最小編集距離計算回路の面積と動作周波数を測定した. 回路の合成には Quartus II version.9.1 を用いた.

図 14 に Stratix III 上への実装結果を, 図 15 に Cyclone III 上への実装結果を示す. 図 14,15 は, 横軸は PE 数を表し, 縦軸はそれぞれ ALUT 数と LE 数を表し, act.Naive, act.LL は Naive 法と LL 法の実験値を表し, est.Naive, est.LL は Naive 法と LL 法の見積り値を表す. また, 図の値は実験値を表す. 図 14, 15 より, 式 (6), (13) を用いると, ほぼ正確な面積の見積りが可能である.

一方, 実装結果より, Stratix III 上で LL 法は Naive 法よりも動作周波数が 2.5〜3.5 倍高く, Cyclone III 上で LL 法は Naive 法よりも動作周波数が 2.6〜3.0 倍高かった. これは, LL 法の PE の構造は Naive 法の PE の構造よりも, 単純であるためと考えられる.

面積と動作周波数の関係に関する結果から, CUPS (Cell Updates Per Second) に関して LL 法は Naive 法よりも優れている.

6. ま と め

本論文では, 近似文字列マッチングを動的計画法で求める Naive 法と LL 法を紹介した. また, パターン長 n に対する最小編集距離の計算回路の面積を導出した. Altera 社 FPGA 上に二つの実現法を実装し, 面積と動作周波数を求め, 実装での面積が計算上での面積見積りと一致していることを確認した. 動作周波数の実験値と, 本論文で求めた面積より, CUPS (Cell Updates Per Second) に関して LL 法は Naive 法よりも優れていると予測できる.

7. 謝　　辞

本研究は, 一部, 日本学術振興会・科学研究費補助金, および, 文部科学省・知的クラスター創成事業 (第二期) の補助金による. 長崎大学柴田裕一郎先生, 筑波大学山口佳樹先生には貴重な御助言を頂いた.

文　　献

[1] Altera Corp, http://www.altera.com/.
[2] B. Buyukkurt and W.A. Najjar, “Compiler generated systolic arrays for wavefront algorithm acceleration on FPGAs,” *FPL2008*, pp. 655-658, Sept., 2008.
[3] L. J. Guibas, H. T. Kung and C. D. Thompson, “Direct VLSI implementation of combinatorial algorithms,” *Proc. Conf. VLSI: Architecture, Design, Fabrication*, pp. 509-525, 1979.
[4] T. Hoang and D. P. Lopresti, “FPGA Implementation of Systolic Sequence Alignment,” *Int. Workshop on Field Programmable Logic and Applications*, 1992.
[5] Dzung Hoang, “Searching Genetic Databases on Splash 2,” Proc. of IEEE Workshop on FCCM, pp. 185-191, 1993.
[6] R.J. Lipton and D. P. Lopresti, “A systolic array for rapid string comparison,” *Proc. of the Chapel Hill Conf. on VLSI*, pp. 363-376, 1985.
[7] G. Navarro, “A guided tour to approximate string matching,” *ACM Computing Surveys*, Vol.33, No.1, pp. 31-38, March, 2001.
[8] S. B. Needleman and C. D. Wunsch, “A general method applicable to the search for similarities in the Amino-Acid sequence of two Proteins,” *Journal of Molecular Biology*, 48, pp. 443-453, 1970.
[9] T. F. Smith and M. S. Waterman, “Identification of common molecular subsequences,” *Journal of Molecular Biology*, 147(1): pp. 195-197, 1981.
[10] O. Storaasli, Y. Yu, D. Strenski and J. Maltby, “Performance evaluation of FPGA-based Biological applications,” *Proc. Cray users group’07*, 2007.
[11] R.A.Wagner and M.J.Fisher, “The string-to-string correction problem,” *J. Ass. Compt Mach.*, Vol. 1, pp. 168-173, 1974.
[12] Y. Yamaguchi, T. Maruyama and A. Konagaya, “High speed homology search with FPGAs,” *Proc. Pacific Symp. on Biocomputing*, pp. 271-282, 2002.
[13] C.W. Yu, K.H. Kowng, K.H. Lee and P.H.W. Leong, “A Smith-Waterman systolic cell,” *FPL2003*, pp. 375-384, 2003.
[14] 清水敬介、笹尾勤、中原啓貴、“Smith-Waterman アルゴリズムの FPGA 上への実装とその評価に関する研究,” 電子情報通信学会 第２３回多値論理とその応用研究会, MVL10-4, pp.16-23, 2010 年 1 月.
[15] 土肥慶亮, L. Cheng, 濱田剛, 柴田裕一郎, 小栗清, K. Benkrid, “Smith-Waterman アルゴリズムにおける GPU を用いた実装方法の一提案,” 電子情報通信学会 *RECONF* 研究会 *(高知)*, vol. 109, no. 319, CPSY2009-43, pp.1-6, 2009 年 12 月.