RC-003

高速なテキスト検索のための近似正規表現マッチングアルゴリズムとその FPGA 実装

An Approximate Regular Expression Matching Algorithm for High-Speed Text Search and Its FPGA Implementation

宇丹 裕一朗 †　若林 真一 †　永山 忍 †
Yuichiro Utan　Shin'ichi Wakabayashi　Shinobu Nagayama

まえがき—テキスト検索とは，与えられた文字列（パターン）に類似する文字列をテキストから探し出すという問題である．主な応用としてはバイオインフォマティクスにおける DNA 配列の解析などがある．本論文では，パターンの記述に正規表現の部分クラスを用い，パターンに類似するすべての部分文字列をテキストから探し出す問題を近似正規表現マッチングと呼ぶ．そして，この問題を 1 次元シストリックアレイを用いて高速に解くことができるハードウェアアルゴリズムを提案し，その FPGA 実装により提案アルゴリズムの有効性を示す．

1　はじめに

ストリングマッチングとは与えられた文字列（パターン）と類似する文字列を入力系列から探索する問題である [1]．主な応用としてはデータベースにおける検索，バイオインフォマティクスにおける DNA 配列の解析などがある．

ストリングマッチングには様々な種類があり，その最も単純なものは，普通の文字列で表されたパターンに完全に一致する部分文字列を入力系列から探索するものである．有名なクヌース-モリス-プラットアルゴリズムはこの種類のストリングマッチングである [6]．正規表現マッチングはより複雑なストリングマッチングであり，パターンに正規表現を用いる [8]．これらは，テキスト内の部分文字列がパターンに正確に一致するイグザクトマッチングに分類される．

他の種類のストリングマッチングとして近似文字列照合が挙げられる [4]．近似文字列照合とは，パターンと類似の文字列を入力系列から検索するものである．この種類のストリングマッチングは，文字列間に類似性の指標を導入する必要がある．主な指標としては，編集距離があげられる [9]．

ストリングマッチングは計算時間を短くするために広く研究されており，ストリングマッチングに関しては，理論的なものから実用的なものまで，数多くの従来研究がある [1]．高速な近似文字列照合を行うためのハードウェアアルゴリズムも，数多く提案されてきた [10]，[11]，[5]．しかしながら，既存のハードウェアのほとんどが，パターンとして正規表現を扱うことができない．一方で，正規表現マッチングに対するハードウェアも，特にネットワーク不正侵入検知システムなどのために数多く提案されているが [2]，[3]，これらは，正規表現を用いたパターンに類似する文字列を探し出すことはできない．私たちの調べた限りでは，正規表現のパターンに類似する文字列を検索可能なハードウェアは，未だ提案されていない．

† 広島市立大学大学院情報科学研究科
Graduate School of Information Sciences, Hiroshima City University

本論文では，パターンに正規表現を使用し，そのパターンに類似する部分文字列を入力系列から探し出すというテキスト検索問題を解くための，効率的なハードウェアアルゴリズムを提案する．より柔軟なテキスト検索が可能となるように，正規表現マッチングと近似文字列照合を組み合わせ [7]，1 次元シストリックアーキテクチャを用いて実現する．提案アルゴリズムを FPGA とソフトウェアの両方で実現し，その 2 つの計算時間を比較することで，提案ハードウェアの有効性を示す．

本論文の構成は以下の通りである．2 章では，近似文字列照合の定式化について述べる．3 章では，近似正規表現マッチングの基本的なハードウェアの構成について述べる．4 章では，3 章に基づいた近似正規表現マッチングについて述べる．5 章では，その FPGA 実装について述べる．最後の 6 章は本論文のまとめとなっている．

2　近似文字列照合

この章では，近似文字列照合問題とそのダイナミックプログラミング (DP) 法に基づいた解法について説明する．

2.1　編集距離 [9]

Σ を任意の文字の集合（アルファベット）とする．A を Σ 上の有限長の文字列とする．$A<i>$ は文字列 A の i 番目の文字である．$A<i:j>$ は A の i 番目から j 番目の文字列である $(i \leq j)$．$|A|$ は文字列 A の長さと定義する．

文字列に対して長さ 1 以下の文字のペア $(a,b) \neq (\varepsilon,\varepsilon)$ を考え，a を b に変換する操作を編集操作と呼ぶ．ε は空語 (長さ 0 の文字) を表し，編集操作 (a,b) を $a \rightarrow b$ と書く．以降，編集操作 $a \rightarrow b$ を置換操作，編集操作 $a \rightarrow \varepsilon$ を削除操作，編集操作 $\varepsilon \rightarrow b$ を挿入操作と呼ぶ．また，編集操作にコストを設定し，$\gamma(a \rightarrow b) \geq 0$ で表す．

アルファベット Σ 上の 2 つの系列 A, B が与えられたとき，系列 A に上記の編集操作を複数回適用して系列 B に変換することを考え，変換するために必要な編集操作の系列を $S = s_1, s_2, \cdots, s_m$ とする．また操作の系列 S のコストを $\gamma(S) = \sum_{i=1}^{m} \gamma(s_i)$ とする．系列 A から系列 B に変換することのできるすべての操作の系列 S を考え，操作の系列 S のコスト $\gamma(S)$ の最小値を編集距離と定義する．

2 つの系列 A, B が与えられたとき，系列 A を系列 B に変換する場合の編集距離 $\delta(A(i), B(j))$ はダイナミックプログラミング法を用いて以下のように求めることができる．$A(i) =< 1 : i >$，$B(j) =< 1 : j >$，$D(i,j) = \delta(A(i), B(j))$，$0 \leq i \leq m$，$0 \leq j \leq n$，$m = |A|$，$n = |B|$ とすると，

$$D(i,j) = min\{ D(i-1,j-1) + \gamma(A<i> \rightarrow B<j>),$$

$$D(i-1,j)+\gamma(A<i>\rightarrow\varepsilon),$$
$$D(i,j-1)+\gamma(\varepsilon\rightarrow B<j>)\} \quad (1)$$

ここで，すべての $i,j, 0\leq i\leq m$，$0\leq j\leq n$ において，$D(0,0)=0$ とし，$D(i,j)$ の範囲外の値は∞とする．

このアルゴリズムは実行時に $D(0,0)$ から順にすべての要素の計算を行う．したがって，$O(mn)$ の計算時間を必要とする．以降，行列 D を編集距離行列と呼ぶ．

2.2 問題の定式化と DP アルゴリズム

本論文では，テキスト検索問題を対象とする．この問題は以下のように定式化することができる．アルファベット Σ 上の 2 つの系列 P（パターン）と T（テキスト），および，ある閾値 k が与えられたとき，パターン P との編集距離が k 以下であるような T の部分文字列を見つける問題である．

この問題はダイナミックプログラミング法で解くことができ，基本的に 2.1 節で示した編集距離の計算と似たものとなる [8]．実際，テキスト検索問題は，$D(0,j)=0, 0\leq j\leq n$ と設定すること以外は，2.1 節のダイナミックプログラミング法と全く同じ解法で解くことができる．図 1 は，$P=annual$，$T=annealing$，$k=2$ とした時の編集距離行列 D の結果である．編集コストは同じ文字の場合は 0 とし，それ以外はすべて 1 としている．例えば，D(2,3) における編集距離は，$D(1,2)$ の結果に文字の置換コストを加えた編集コスト $D(1,2)+\gamma(P<2>\rightarrow T<3>)=1+0=1$，$D(1,3)$ の結果に文字の削除コストを加えた編集コスト $D(1,3)+\gamma(P<2>\rightarrow\varepsilon)=1+1=2$，$D(2,2)$ の結果に文字の挿入コストを加えた編集コスト $D(2,2)+\gamma(\varepsilon\rightarrow T<3>)=0+1=1$，これら 3 つの最小値である 1 となる．この例では，パターン P に似た部分文字列は $annea$，$anneal$，$anneali$ の 3 つである．

P＼T	0	1 a	2 n	3 n	4 e	5 a	6 l	7 i	8 n	9 g
0	0	0	0	0	0	0	0	0	0	0
1 a	1	0	1	1	1	0	1	1	1	1
2 n	2	1	0	1	2	1	1	2	1	2
3 n	3	2	1	0	1	2	2	2	2	2
4 u	4	3	2	1	1	2	3	3	3	3
5 a	5	4	3	2	2	1	2	3	4	4
6 l	6	5	4	3	3	2	1	2	3	4

図 1: テキスト検索の例

3 ハードウェアアルゴリズム

3.1 アーキテクチャ

本論文では，テキスト検索問題に対し，FPGA 上で実装するハードウェアアルゴリズムを提案する．アルゴリズムの基本的なアイデアは以下のとおりである．2 つの系列が与えられたとき，編集距離は編集距離行列を計算することで求めることができる．また，編集距離行列の同一対角線上の値はデータの依存関係がないことから，並列に計算することができる．そこで，提案アルゴリズムでは，編集距離行列の各行に計算ユニットを割り当て，図 2 に示すように同一対角線上にある要素を各計算ユニットで同時に計算することで，左上から右下に向って高速に計算を行う．

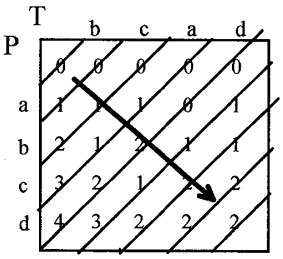

図 2: 編集距離行列の計算順序

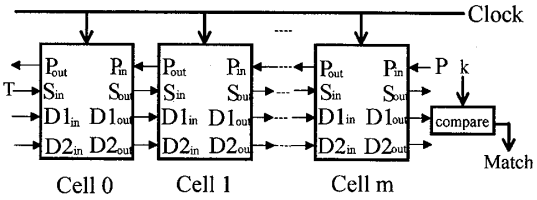

図 3: 提案アルゴリズムのアーキテクチャ

提案するハードウェアアルゴリズムの概略図を図 3 に示す．$|P|=m$ とするとアーキテクチャは，1 文字の比較演算を行う m+1 個の計算ユニット（セルと呼ぶ）を 1 次元配列状に直列接続することで実現される．上述のように，アルゴリズムは編集距離行列の対角線上の要素を並列かつパイプライン処理で計算する．計算された編集距離は最も右のセルから出力され，比較器に入力される．比較器はあらかじめユーザに指定されたパターメータ k の値と比較を行い，編集距離が k 以下の値の場合，マッチ信号を出力する．アルゴリズムの詳細については以下で説明する．

3.2 アルゴリズムの入力

提案ハードウェアアルゴリズムの入力について説明する．パターン $P=p_1p_2\cdots p_m$ とすると，P の先頭に開始記号 θ を付加することで入力パターン P' を生成し，P' を右端のセルの Pin からクロックに同期して入力する．入力パターン P' は編集距離の計算を実行する前に予め入力しておく．パターン P の i 番目 (入力パターン P' の $i+1$ 番目) の文字は左から $i+1$ 番目のセルに記憶され，行列 D の i 行目の値を計算する．編集距離の計算では行列 D の 0 行目も計算しなければならないため，入力パターンの先頭文字である θ を記憶しているセル (左端のセル) で 0 行目の値を計算する．一方，パターン P との編集距離を計算する入力系列 $T=t_1,t_2,\cdots,t_n$ に対する入力系列は $T'=\theta t_1,t_2,\cdots,t_n\lambda$ となる．編集距離の計算実行時に入力系列 T' を左端のセルの Sin からクロックに同期して入力する．

3.3 セルの構造

提案ハードウェアはセルと呼ばれる計算ユニットから構成されている (図 4)．P,C は入力系列を 1 文字記憶するラッチで，P にはパターン P の 1 文字が記憶され，C には入力系列 T の 1 文字が記憶される．DM はコスト関数 γ の値を格納するメモリである．提案ハードウェア

アルゴリズムでは，距離行列と呼ばれる DM の値を，P と C に記憶されている文字の組み合わせによって決定する．つまり，任意の a と b という文字に対し，$DM(a,b)$ には $\gamma(a \to b)$ の値が格納される．距離行列のすべての要素の値はストリングマッチングが始まる前に設定される．$D1, D2$ は行列 D の 1 つの要素を記憶するレジスタで，$D1$ はそのセルが計算した最新の値が，$D2$ には最新の 1 つ前の値が，それぞれ記憶される．E はセルのアルゴリズムで使われているフラグである．E の初期値には 0 が入力される．

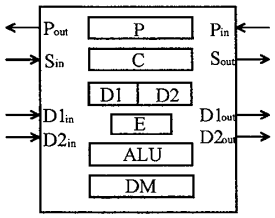

図 4: セルの構造

3.4 セルの動作

各セルの動作はパターン入力フェーズと編集距離計算フェーズの 2 つに分類される．パターン入力フェーズではパターン文字列が最も右のセルから 1 文字ずつ入力され，すべての文字が入力されるまで左にシフトされる．次に，実際にテキスト検索を行う編集距離計算フェーズについて説明する．

編集距離計算フェーズの動作記述を図 5 に示す．各クロックサイクルは 2 相クロック $(t0, t1)$ から構成されている．アルゴリズムの del，ins，sub はそれぞれ，削除，挿入，置換の操作コストを表し，編集距離の定義により，これらはそれぞれ $\gamma(a \to \varepsilon)$，$\gamma(\varepsilon \to b)$，$\gamma(a \to b)$ に対応する．a と b はそれぞれ P と C に格納された文字である．

セル i は，パターンの文字 p_i を格納し，編集距離行列の $D(i,*)$ のすべての要素を計算する．セル i が左隣のセルからテキスト文字 t_j を受け取り，ラッチ C に格納しクロックサイクル T_k で $D(i,j)$ を計算する動作を図 6 に示す．要素 $D(i,j)$ を計算するために，$D(i,j-1)$，$D(i-1,j)$，$D(i-1,j-1)$ の要素が必要となる．$D(i,j-1)$ は，クロックサイクル T_{k-1} のときセル i で計算され，ラッチ $D1$ に保持されている．$D(i-1,j)$ もクロックサイクル T_{k-1} のときセル $i-1$ で計算され，$D1$ に保持されている．$D(i-1,j-1)$ はクロックサイクル T_{k-2} のときセル $i-1$ で計算され，$D1$ に保持されていた値であり，この値は，サイクル T_{k-1} のとき D2 にシフトされ，保持されている．そのため，$D(i,j)$ を計算する値はすべてセル i もしくはセル $i-1$ に格納されており，図 6 に示されるように，それらの値を容易に収集できる．

4 近似正規表現マッチング

3 章で述べた近似文字列照合は “ドントケア”や “否定” などのマッチングを行うことができない．しかし，テキスト検索ではそれらを用いたパターンでの検索は有用なものである．本論文では，正規表現マッチングを近似文字列照合に導入することに焦点を当てている．正規表現演算を導入する際，マッチングの結果がパターンとテキストの編集距離として評価されるように，編集距離を算出する方法を考えなければならない．

```
t0: {
    C ← Sin;
    if  (Sin == θ)  E=1;
    if  (Sin == λ)  E=0;
};
t1: {
    if  (E == 1) {
          if (C == θ) D1=D1in + del;
          else D1 = min{ D1in + del,
                   D1 + ins, D2in + sub};
    }
    else D1 = 0;
    D2 = D1;
};
```

図 5: セルの動作記述

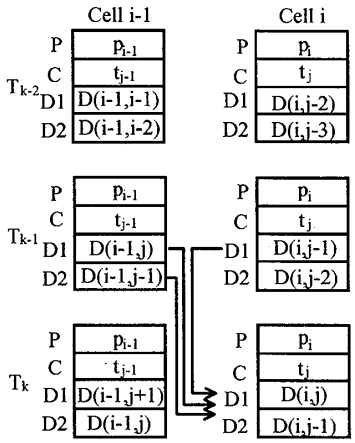

図 6: アルゴリズムの動作

4.1 パターンの拡張

1. 1 文字のドントケア（SCDC，？）
   ’?’ は Σ 上の任意の 1 文字とマッチする．
2. 任意長のドントケア（VLDC，?* ）
   ’?*’ は Σ 上の任意の 0 文字以上の文字列にマッチする．例えば，$a?^*$ は空語 ε にもマッチする．
3. 否定演算（$\bar{p}$）
   p を Σ 上の任意の文字とし，$\bar{p}$ は指定した文字以外のすべての文字にマッチする．
4. 空語（ε）
   p を Σ 上の任意の文字とし，$p@$ は p もしくは ε にマッチする．
5. 同一文字数の照合演算（$[p_1p_2\cdots p_s]$）
   任意の系列 $P = p_1p_2\cdots p_s$ は同じ長さのテキスト $T = t_1t_2\cdots t_s$ にマッチする．
6. イグザクトマッチ（$< [p_1p_2\cdots p_s] >$）
   任意の系列 $P = p_1p_2\cdots p_s$ がテキスト $T = t_1t_2\cdots t_s$ に完全に一致すれば，P と T の編集距離は 0 となり，それ以外は ∞ となる．

7. クリーネ閉包（p^*）
   Σ 上の任意の文字 p^* は，文字 p の任意回の繰り返しの文字列とマッチする．

3 章で示したハードウェアアルゴリズムに上述の機能を導入することで，近似文字列照合を用いる種々のアプリケーションに役立つ．

上述のパターンは正規表現とみなすことができるため，上述のパターンとマッチする文字列のクラスは，正規集合のサブクラスとなる．しかし，本論文で対象としているテキスト検索のアプリケーションは上記のサブクラスを超える複雑な正規表現でパターンが記述されることがほとんどないため，定義したサブクラスで，実用上十分にパターンを表現することができる．

一般に，正規表現を用いたストリングマッチングは正規表現マッチングと呼ばれる [8]．正規表現マッチングでは，パターンは入力文字列に完全に一致しなければならないのに対し，本論文のマッチングでは，閾値 k を正の値に設定することでいくつかの間違いを許すことができるマッチングである．したがって，本論文で説明したストリングマッチングは正規表現マッチングとは異なるものであるといえる．

4.2 DP に基づく解法

3 章で説明した近似文字列照合のアルゴリズムを拡張することで，前節のマッチング演算を可能にすることができる．

1 文字のドントケア ? は，置換のコスト関数を式 (2) のように拡張することで実現できる．

$$sub(p,t) = \begin{cases} \gamma(p \to t) & \texttt{if } p \in \Sigma \\ 0 & \texttt{if } p = ? \end{cases} \quad (2)$$

他の演算における del，ins，sub の値は，表 1 に示す．

表 1: マッチング演算

演算	del	ins	sub
演算なし p	$del(p)$	$ins(t)$	$sub(p,t)$
1. SCDC [1] ?	$del(?)$	$ins(t)$	$sub(?,t)$
2. VLDC [2] ?*	0	0	0
3. 否定演算 $\bar{p}$	$del(\bar{p})$	$ins(t)$	$sub(\bar{p},t)$
4. 空語 $p@$	0	$ins(t)$	$sub(p,t)$
5. CC マッチ [3] $[p_1p_2\cdots p_t]$ $p_i(1 \le i < t)$ p_t	 ∞ ∞	 ∞ $ins(t)$	 $sub(p_i,t)$ $sub(p_t,t)$
6. イグザクトマッチ $\langle[p_1p_2\cdots p_t]\rangle$ $p_i(1 \le i < t)$ p_t	 ∞ ∞	 ∞ $ins(t)$	 $equ(p_i,t)$ $equ(p,t)$
7. クリーネ閉包 p^*	0	$\min\{ ins(t), sub(p,t)\}$	$sub(p,t)$

[1]SCDC：1 文字のドントケア．
[2]VLDC：任意長のドントケア．
[3]CC マッチ：同一文字数の照合演算．

1 文字のドントケア ? の削除コスト $del(?)$ は，式 (3) のように定義する．

$$del(?) = min\{del(q) | q \in \Sigma\} \quad (3)$$

p の否定演算の削除，置換コストはそれぞれ式 (4)，式 (5) のように定義する．

$$del(\bar{p}) = min\{del(q) | q \in \Sigma \ \texttt{ and } q \neq p\} \quad (4)$$

$$sub(\bar{p},t) = \begin{cases} 0 & \texttt{if } p \neq t \\ \texttt{min}\{sub(q,t) | q \in \Sigma \texttt{ and } q \neq t\} & \texttt{otherwise} \end{cases} \quad (5)$$

イグザクトマッチにおいては，置換コスト $sub(p,t)$ の代わりに，等価関数 $equ(p,t)$ を使用する．$equ(p,t)$ を式 (6) に示す．

$$equ(p,t) = \begin{cases} 0 & \texttt{if } p = t \\ \infty & \texttt{otherwise} \end{cases} \quad (6)$$

表 1 に示した DP の定式化について説明する．ページ数制限のため，イグザクトマッチとクリーネ閉包以外の説明は割愛する．まず，イグザクトマッチについて説明する．$P = \langle[p_1p_2\cdots p_t]\rangle$ をパターンとすると，イグザクトマッチの定義から，P がテキスト内の部分文字列 $T = [t_1t_2\cdots t_t]$ にマッチするのは $P = T$ の場合のみである．これは $equ(p_i,t_i)(1 \le i \le t)$ をチェックすることで確認できる．パターンの文字はテキストの文字にマッチすることなく削除することは許されないので，削除コストは∞となる．同様にテキストの文字はパターンの最後の文字以外の文字にマッチすることなく挿入することは許されないので，挿入コストは∞となる．パターンの最後の文字に対しては，パターンのマッチングは終了しているので，テキスト文字を挿入することができる．したがって，テキスト文字の挿入コストを計算する必要がある．

次に，1 文字のクリーネ閉包 p^* について説明する．p^* は空語とマッチする，すなわち 1 文字もテキストとマッチしなくてもよい．したがって，パターンの文字の削除コストは 0 となる．また，p^* は 1 以上の長さの任意の文字列とマッチする．つまり，テキストの文字の挿入コストは挿入コストと置換コストの最小値となる．p^* の置換コストは通常通りとなる．

4.3 ハードウェア実装

ここでは，4.1 節で説明した近似正規表現マッチングのハードウェア実装について説明する．DP の定式化は del，ins，sub のコストを表 1 のように変更するだけで実装が可能である，したがって，3 章で説明したアーキテクチャと同様のものになる．各セルに対し，新たに否定演算，VLDC，クリーネ閉包などの演算を指定するための $Func$ というレジスタを追加した．$Func$ の値はパターン入力フェーズで入力される．

4.4 ユニオンと空単語

4.1 節で説明した近似正規表現マッチングは，図 3 で示したハードウェアの全体の構造を変更する必要はないが，ハードウェア構成に若干の変更を加えるだけで，さらに多くのパターンのマッチング演算を導入することができる．以下に，4 つのパターンの拡張を示す．

1. ユニオン ($\{p_s, p_{s+1}, \cdots, p_t\}$)
   テキスト文字 t は $p_s, p_{s+1}, \cdots, p_t$ 内の文字のいずれかとマッチする．t とパターンの編集距離は t とそれぞれの p_i との編集距離の最小値となる．
2. イグザクトユニオン ($\langle\{p_s, p_{s+1}, \cdots, p_t\}\rangle$)
   テキスト文字 t は $p_s, p_{s+1}, \cdots, p_t$ 内の文字のいずれかとマッチする．t とパターンの編集距離は $t = p_i$ となる p_i が存在すれば 0 となり，それ以外は∞となる．
3. 空語付きイグザクトマッチ ($\langle[p_s p_{s+1} \cdots p_t]\rangle$@)
   テキストの部分文字列は，パターンと完全にマッチする，もしくは空語 ε とマッチする．
4. 空語付きイグザクトユニオン ($\langle\{p_s, p_{s+1}, \cdots, p_t\}\rangle$@)
   テキスト内の文字がパターンのいずれかの文字に完全にマッチする，もしくは空語 ε とマッチする．

以上のパターンの拡張を実現するため，ハードウェアの構造をわずかに変更する必要がある．以下に，拡張した DP の定式化を示す．拡張のために行列 $U(i,j)$ と $E(i,j)$ を新たに追加する．

1. ユニオン ($\{p_s, p_{s+1}, \cdots, p_t\}$)
   $D(s,j) = D(s-1, j-1)$, $U(s,j) = sub(p_s, t_j)$.
   $D(i,j) = D(i-1,j)$, $U(i,j) = \min\{sub(p_i, t_j), U(i-1,j)\}$, $s < i < t$.
   $D(t,j) = \min\{D(t-1,j) + sub(p_t, t_j), D(t-1,j) + U(t-1,j), D(t,j-1) + ins(t_j)\}$.
2. イグザクトユニオン ($\langle\{p_s, p_{s+1}, \cdots, p_t\}\rangle$)
   $D(s,j) = D(s-1, j-1)$, $U(s,j) = equ(p_s, t_j)$.
   $D(i,j) = D(i-1,j)$, $U(i,j) = \min\{equ(p_i, t_j), U(i-1,j)\}$, $s < i < t$.
   $D(t,j) = \min\{D(t-1,j) + equ(p_t, t_j), D(t-1,j) + U(t-1,j), D(t,j-1) + ins(t_j)\}$.
3. 空語付きイグザクトマッチ ($\langle[p_s p_{s+1} \cdots p_t]\rangle$@)
   $D(s,j) = D(s-1, j-1) + equ(p_s, t_j)$, $E(s,j) = D(s-1,j)$.
   $D(i,j) = D(i-1, j-1) + equ(p_i, t_j)$, $E(i,j) = E(i-1,j)$, $s < i < t$.
   $D(t,j) = \min\{D(t-1,j) + ins(t_j), D(t-1, j-1) + equ(p_t, t_j), E(t-1,j)\}$.
4. 空語付きイグザクトユニオン ($\langle\{p_s, p_{s+1}, \cdots, p_t\}\rangle$@)
   $D(s,j) = D(s-1, j-1)$, $U(s,j) = equ(p_s, t_j)$, $E(s,j) = D(s-1,j)$.
   $D(i,j) = D(i-1,j), U(i,j) = \min\{equ(p_i, t_j), U(i-1,j)\}$, $E(i,j) = E(i-1,j)$, $s < i < t$.
   $D(t,j) = \min\{D(t-1,j) + equ(p_t, t_j), D(t-1,j) + U(t-1,j), D(t,j-1) + ins(t_j), E(t-1,j)\}$.

イグザクトユニオン以外も同様であるため，紙面の都合上，イグザクトユニオンの DP の定式化のみを説明する．図 7 に示すように，テキスト文字 t_j がパターン ($\langle\{p_s, p_{s+1}, \cdots, p_t\}\rangle$) のどの文字にマッチするかを探す場合を考える．$t_j = p_i$, $s \le i \le t$ となるような p_i が存在すればマッチングは成功する．マッチングの結果は $D(t,j)$ として得ることができ，もし，マッチング成功ならば，$D(t,j) = \min\{D(s-1, j-1), D(t,j-1) + ins(t_j)\}$ となり，失敗ならば，$D(t,j) = D(t,j-1) + ins(t_j)$ となる．まず，$D(s-1, j-1)$ の値が $D(s,j)$ に入り，この値がセルを移動してセル t まで送られる．$s \le i \le t$ の各セル i は，$t_j = p_i$ かどうかを判定し，1 つでも $t_j = p_i$ となるセルがあれば，U は 0 となる．それ以外の場合，U は∞となる．U の値もセル t まで送ることで，セル t は $D(t,j)$ の値を正確に計算することができる．

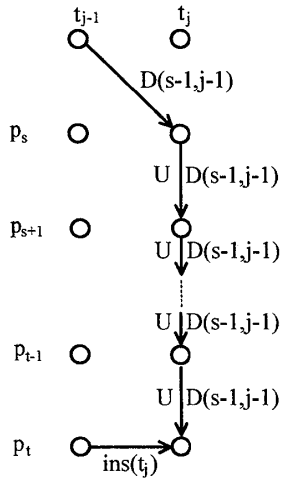

図 7: イグザクトユニオン

上で説明した拡張したパターンを使用する近似正規表現マッチングのハードウェアの構造について説明する．上述の DP の式を実現するには，各セルにセル自身もしくは隣のセルにデータを送る必要がある．これを実現するために，U, E のレジスタを各セルに追加し，隣り合うセルの 3 章で説明したストリングマッチングのように U と E を接続する．紙面の都合上，詳細は割愛する．

4.5 その他の拡張

近似正規表現マッチングのハードウェア構造に対する，いくつかの拡張がある．

1. マッチした部分文字列のトレースバック
   提案アルゴリズムは，テキスト内のマッチした部分文字列の最後の文字の位置を出力する．いろいろなアプリケーションのために，マッチした部分文字列の最初の文字の位置を見つける必要がある．ストリングマッチングを逆に行えば，最初の文字を計算することが可能である．最も単純な方法は，パターンを逆に入力し，テキストを逆からアルゴリズムに入力することである．もう一つの方法は，データの流れの向きが変わるようにハードウェアを拡張することである．この拡張ではアルゴリズムは 2 つのモードを持つ：ノーマルモードとリバースモードである．ノーマルモードでは，テキストは左端から右端に入力される．これに対しリバースモードでは，テキストは右端から左端へ入力される．この拡張では，マッチした部分文字列のトレースバックを実行する際，パターンを逆向きのパターンに入れ替える必要はない．

2. ストリングマッチング
   2 章で説明したように，2 つの文字列の近似テキスト検索と近似ストリングマッチングは，ほぼ同じ DP アルゴリズムで解くことが可能である．2 つのアルゴリズムの異なる点は，$D(0,j)$, $0 \le j \le n$ の値のみである．したがって，提案ハードウェアをストリ

ングマッチングモードとテキスト検索モードの 2 つのマッチングが可能となるようにアーキテクチャに拡張を加える．

3. 文字クラス
テキストデータベースへ応用する場合，文字のクラスをパターンの表記に導入することは有用である．文字クラスの例としては，アルファベット文字 $(a, b, c, \cdots)$，数字 $(1, 2, 3, \cdots)$，句読点 (“.”, “,”, “;”, “:”, $\cdots$)，などが挙げられる．これはユニオン演算を使用することで実現することができるが，パターンの記述に便利な表記を導入し，それに対応するハードウェア構造を用いることで，より簡潔にパターンを記述することができる．

これらは比較的容易に実装できるが，本論文で対象とする高速なテキスト検索エンジンの設計に直接関係するものではない．そこで，本論文では，提案した検索エンジンの性能を正確に把握するために，実験においてこれらの拡張は実装しなかった．

5 FPGA 実装

本論文で提案したストリングマッチングのための専用ハードウェアを Verilog HDL で記述し，FPGA 上に実装した．FPGA 設計に用いた設計環境を以下に示す．論理合成ツールに Altera Quartus II 9.1，また，論理合成の対象とする FPGA は Altera 社の Stratix EP1S60F1020C7 を使用した．設計にあたって，置換操作のコストは同じ文字の置換ならばコスト 0，異なる文字の置換ならばコスト 1 とした．また，削除コスト，挿入コストはともに 1 とした．

3 章のセルの動作の説明において，2 相クロックを用いたが，実際の回路設計では FPGA の設計で通常使用されている 1 相クロックを使用する．回路は 2 つの状態 T_0 と T_1 を持っている．それは 3 章のセルの動作で説明した t_0 と t_1 に対応する．また，2 つの状態はパイプライン処理される．例えば，テキスト内の i 番目の文字をマッチングする T_1 の回路は，$(i+1)$ 番目の文字をマッチングしている T_0 と同時に行われている．したがって，テキスト 1 文字のパターンマッチングは 1 クロックサイクルで実行される．

近似正規表現マッチングのアルゴリズムをソフトウェアで実現し，提案したハードウェアアルゴリズムとの計算時間の比較を行った．ソフトウェアは CPU が Pentium D 3.0GHz，メインメモリが 2GB の PC 上で実行した．また，コンパイラには Blorland C++ 5.5.1 を使用し，最適化オプションとして-O2 を使用した．

設計結果は，250 個のセルを実装した場合，論理ブロック数は 41,491 個となり，論理ブロック数の使用率は 72.6% となった．また，セル 1 個当たりの論理ブロック数は 166 個となった．最高動作周波数は 79.92MHz となった．また，FPGA 上で回路を構成し，正しく回路が動作するかを確認した．

ソフトウェアとハードウェアの実行時間の比較について示す．入力パターンの文字数を 250 文字，入力系列の文字数はパターンの 100 倍の 25000 文字とした．提案ハードウェアはソフトウェアに対する速度向上比が 364 倍となっており，十分な高速化が達成されていると言える．これは，ソフトウェアでは編集距離を計算する際，行列 D の要素を逐次的に計算し，時間計算量が $O(mn)$ であるのに対し，ハードウェアでは，並列処理を導入することで $O(m+n)$ の計算量で計算することができるからだと考えられる．

6 おわりに

近似正規表現マッチングに対するハードウェアアルゴリズムを提案し，FPGA 上で実装した．与えられたテキスト内からパターンを検索するために，普通の近似文字列照合よりも柔軟な検索ができるようにパターンに拡張を加え，いくつかのマッチング演算子をパターンに導入した．また，ソフトウェアとの実行時間の比較により，提案ハードウェアの有効性を確認した．今後の課題として，パターンのさらなる拡張などが挙げられる．

参考文献

[1] J.Aoe(eds), *Computer Algorithms: String Matching Strategies*, IEEE Computer Society Press, 1994.

[2] J. Bispo, I. Sourdis, J.M.P.Cardoso, S.Vassailiadis, “Regular expression matching for reconfigurable packet inspection”, *Proc. 2006 IEEE International Conference on Field Programmable Tecnology*, pp.119-126, 2006.

[3] B.C.Brodie, R.K.Cytron, D.E.Taylor, “A scalable architecture for high throughput regular-expression pattern matching”, *Proc. 33rd International Symposium on Computer Architecture*, pp.191-202, 2006.

[4] P.A.V.Hall and G.R.Dowling, “Approximate string matching”, *ACM Computing Surveys*, Vol.12, No.4, pp.381-402, 1980.

[5] M.C.Herbordt, J.Model, Y.Gu, B.Sukhwani, T.VanCourt, “Single pass, BLAST-like, approximate string matching on FPGAs”, *Proc. 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pp.217-216, 2006.

[6] D.E.Knuth, J.H.Morris, Jr., V.R.Pratt, “Fast pattern matching in strings”, *SIAM Journal on Computing*, Vol.6, No.2, pp.323-350, 1977.

[7] E.W.Myers, W.Miller, “Approximate matching of regular expressions”, *Bulletin of Mathematical Biology*, 51, pp.7-37, 1989.

[8] G.Navarro, M.Raffinot, *Flexible Pattern Matching in Strings*, Cambridge University Press, 2002.

[9] R. A. Wagner and M. J. Fischer, “The string-to-string correction problem”, *J.ACM, Vol.21, No.1*, pp.168-178, 1974.

[10] B.West, R.D.Chamberlain, R.S.Indeck, Q.Zhang, “An FPGA-based search engine for unstructured database”, *Proc. 2nd Workshop on Application Specific Processors*, pp.25-32, 2003.

[11] C.W.Yu, K.H.Kwong, K.H.Lee, and P.H.W.Leng, “A Smith-Waterman systolic cell”, *Proc. Int'l Conf. Field-Programmable Logic and Applications*, LNCS 2778, pp.375-384, 2003.