A-010

SeqBDD を使用した文字列間類似結合

高嶋　宏之†　　白井　康之†,††

† JST-ERATO 湊離散構造処理系プロジェクト　〒 060-0814 札幌市北区北 14 条西 9 丁目
†† 北海道大学大学院情報科学研究科　〒 060-0814 札幌市北区北 14 条西 9 丁目
E-mail: †{takashima,shirai}@erato.ist.hokudai.ac.jp

あらまし　近年，データベース分野やデータマイニング分野において，文字列間類似結合（String Similarity Joins）に関する研究が注目を集めている．文字列間類似結合の方法論としては，データエントリの傾向に着目したフィルタリング手法に基づくものが多いが，本稿では，より汎用的な方法として，系列二分決定グラフ（SeqBDD）を用いた手法を提案する．SeqBDD を用いた文字列間類似結合アルゴリズムを示すとともに，トライ構造に基づく実装と比較し，データセットの規模に応じて探索時間が向上する可能性のある実例を示す．

キーワード　系列二分決定グラフ, 文字列間類似結合

1. まえがき

近年，データベース分野や自然言語処理分野，あるいはデータマイニング分野において，アイテム集合を要素とする異なる集合（トランザクションデータベース）間での類似結合 (Set Similarity Joins) 手法が注目を集めている [1] [2] [4] [6] [13] [14].

集合間類似結合は，余弦や Jaccard 係数，編集距離といった尺度に基づき，2 つの大規模なトランザクションデータベースから類似したトランザクションのペアを正確にすべて抽出・列挙する手法であり，一般に要素間の組み合わせ数は膨大になることから，いかに効率的に与えられた条件を満たすペアを発見できるかが課題となっている．

著者らは，以前ゼロサプレス型二分決定木 (Zero-suppressed Binary Decision Diagrams，以下 ZDD) [8] [10] [11] を用いることで，集合間類似結合に対する効率的なマッチングアルゴリズムを提案した [12]．ZDD は R. E. Bryant による二分決定木（Binary Decision Diagrams，以下 BDD）[3] の変種であるが，大規模な組み合わせデータ集合に対してより圧縮効果の高いデータ構造である．

一方，ZDD を発展させた系列二分決定グラフ（Sequence BDD，以下 SeqBDD）[5] [9] は系列集合を扱うように拡張された無閉路有向グラフである．本稿では，文字列照合への応用として，SeqBDD をベースにしたアルゴリズムを提案する．

本稿の構成は以下のとおりである．2. 章では，関連する既存研究を概観し，3. 章で，基本的定義ならびに ZDD，SeqBDD を用いた類似結合アルゴリズム，4. 章で実装上の工夫点を示す．5. 章では，実行効率評価として，トライ構造を用いた実装 [6] [13] との比較結果を示す．6. 章では，本稿のまとめと今後の課題を示す．

2. 関連研究

類似ペアをすべて正確に列挙する集合間類似結合に関して，近年，さまざまな取り組みが報告されている [1] [2] [4] [6] [13] [14]

Chaudhuri と Arasu ら [4] [1] は，SSJoin と呼ばれる一般的な演算を導入し，編集距離や Jaccard 係数，ハミング距離等さまざまな尺度へ拡張可能な集合間類似結合技術を提案している．SSJoin を用いた PartEnum や WtEnum といったアルゴリズムは，データの分割とシグネチャの生成・比較といった処理に基づき，効率的なフィルタリングを行う手法である．

Bayardo ら [2] は，与えられた閾値以下の関連度を持つトランザクションの組み合わせを効率的に除外するフィルタリング手法を示した．彼らの提案する All-Pairs アルゴリズムは，大規模なレコードサイズに対してもスケーラブルであることが示されている．また，Xiao [14] らは，ミスマッチの個数に着目した Ed-Join アルゴリズムを示し，効率的な候補の絞り込み手法を提案している．

これらのアプローチは，いずれも不要な照合を可能な限り回避するためのフィルタリング手法である．すなわち，条件を満たさないペアを早期に枝狩りし，可能性のあるペアについては，テストフェーズとして検証が行われる．こうした枝狩り手法は，アイテム数（文字数）やレコード長に対して，一般にセンシティブであり，適切にパラメータを設定できる問題に対しては有効であるものの，例えば，比較的長さの短い文字列に対しては，オーバーヘッドとなることも指摘されている [6] [13].

以上のようなフィルタリング手法に対して，Feng，Wang ら [6] [13] はトライ構造を用いて，Trie-Join と呼ばれる探索手法と枝狩り手法を提案している．また，Trie-Join を改良したより効率的なアルゴリズムも提案されている [7].

本稿では，Trie-Join [6] [13] 同様に，フィルタリング手法ではなく，データの表現方法とこれに基づく類似データの探索手法を DAG 形式のダイアグラムを用いて検討するものとする．

DAG 形式のダイアグラムでは，同じ部分的構造を共有することができるため，従来のツリー構造の方法に比べて，重複した部分の探索を効率化することが可能である．

3. SeqBDD を用いた文字列間類似結合

本研究では，DAG 形式のダイアグラムとして，SeqBDD [5] [9] を用いたアルゴリズムを提案する．以下，基本的定義ならびに SeqBDD のベースとなっている BDD(ZDD)，ならびに SeqBDD について概説し，SeqBDD を用いた類似結合手法について示す．

3.1 準　　備

以下，いくつかの用語と記法を定義する．

アイテムは，アルファベット小文字 $a, b, c, \ldots$ を使って表わす．また，アイテムの系列を文字列と呼ぶ．たとえば $bcadega$ は長さ 7 の文字列である．文字列の集合を文字列データベースと呼び，$\mathcal{S}$ あるいは $\mathcal{T}$ などの記号を用いて表わす．文字列データベースを単に “データベース” と呼ぶこともある．本稿では，2 つの文字列間の距離を編集距離で表わす．たとえば，以下のような文字列データベース $\mathcal{S}$，$\mathcal{T}$ があったとする：

$$\mathcal{S} = \{ab, ac, c\} \quad (1)$$

$$\mathcal{T} = \{ad, abc, b\} \quad (2)$$

このとき，編集距離 1 を条件とした場合，$\mathcal{T}$ の $\mathcal{S}$ に対する類似結合の解は，以下の通りである：$\{ab, ad\}$, $\{ab, abc\}$, $\{ab, b\}$ $\{ac, ad\}$, $\{ac, abc\}$, $\{c, b\}$.

また，編集距離 2 を条件とした場合には，上記にさらに以下が加わる．$\{ac, b\}$, $\{c, ad\}$, $\{c, abc\}$.

3.2 ZDD

本節では，BDD ならびに ZDD について概説する．

BDD [3] [8] は，大規模なブール関数を効率的に表現する無閉路有向グラフである．縮約ルールは，ノード削除ルール（2 つのエッジがともに同じノードを指しているようなノードを冗長なものとして削除する）ならびにノード共有ルール（等価なサブグラフを共有する）からなる．一方，ZDD [8] [10] [11] は，大規模な組み合わせアイテム集合，特に疎なアイテム集合を効率的に取り扱うことができるように改良された BDD の変種である．ZDD の縮約ルールは，以下のとおりである．

- 等価なノードを共有する（BDD と同様）．(図 1 の (1)).
- 1-edge が直接 0-ターミナルノードを指しているようなノードを削除し，0-edge の到達先に直接接続させる．(図 1 の (2)).

ZDD は，BDD と比較して，購買履歴データなどのような疎なデータ構造に対してより効率的であることが知られている．たとえば，1000 アイテムから 10 アイテムを選択するような組み合わせ集合の表現においては，ZDD は BDD と比較して，100 倍程度簡潔に表現できる場合もある．

3.3 SeqBDD

SeqBDD [5] [9] は系列集合を扱うように拡張された無閉路有向グラフである．本研究では文字列照合への応用として，SeqBDD をベースにしたアルゴリズムを実装した．

SeqBDD では，与えられたダイアグラム中で，1-終端節点にいたる経路がそれぞれの系列を表現する．図 2 に，文字列集合 $\{aec, ba, ea\}$ を表す SeqBDD の例を示す．

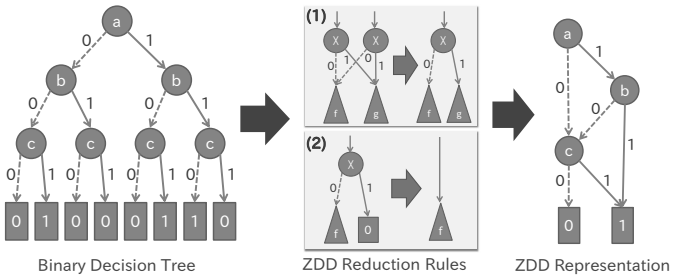

図 1　ZDD による二分決定木の縮約

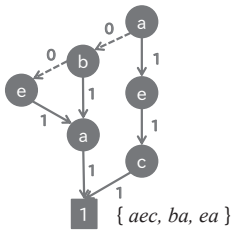

図 2　SeqBDD の例

3.4 SeqBDD に基づく文字列間類似結合

本手法では，まず与えられた 2 つの文字列データベースを SeqBDD 構造に変換し，2 つの SeqBDD 構造に対する文字列間類似結合を行う．SeqBDD 構造上での探索アルゴリズムの概要をアルゴリズム 1 に示す．また，図 3 は SeqBDD を用いた類似結合の例である．この図では，編集距離 1 の制約のもとで $\mathcal{S}$ (左) と $\mathcal{T}$ (右) において類似した文字列を見つけることを目的としている．なお図では実線が 1-枝，点線が 0-枝を表し，0-終端節点は省略した．

探索プロセスは，トップノードである $\mathcal{S}$ 上の a より開始される（アルゴリズム 1 における $search\_seqbdd(n)$）．

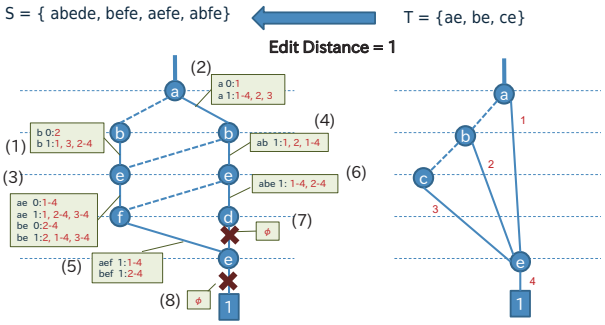

図 3　SeqBDD による類似結合 (1)

各エッジ上のボックスは，$\mathcal{S}$ の探索パス，編集距離，及びその編集距離に対応する $\mathcal{T}$ 上のパス（枝の系列）のリストを意味する（実質 1 枝のみデータが更新されるため，0 枝のボックスは省略）．

たとえば，$\mathcal{S}$ 上のボックス（1）の b 0 : 2 は，$\mathcal{S}$ の探索パスが b，$\mathcal{T}$ 上のエッジ 2（b）に対して，カレントの編集距離が 0 であることを表わしている．(1) の b 1 : 1, 3, 2-4 も同様に，$\mathcal{S}$ の探索パスが b，$\mathcal{T}$ 上のエッジ 1（a），3（c），2-4（be）に対して，カレントの編集距離が 1 であることを表わしている．同様の形で，ボックス (2) も生成される．

続いてボックス (3) を生成するためには，(1) の結果に加えて，(2) の結果も必要である．木構造と異なり，ダイアグラムでは枝の合流があるため，すべての親エッジが処理されたのちに探索を行う横型探索がベースとなっている．

ボックス（1）からの探索パス be に対しては，$\mathcal{T}$ 上のエッジ 2-4（be）がカレントの編集距離 0，エッジ 2（b），1-4（ae），3-4（ce）が編集距離 1 となる．ボックス（2）からの探索パス ae に対しては，$\mathcal{T}$ 上のエッジ 1-4（ae）が編集距離 0，エッジ 1（a），2-4（be），3-4（ce）が編集距離 1 となり，合計 8 パターンの候補が生成される．

以降，同様の形で（4）～（6）と処理が進むが，(7）は制約である編集距離 1 の条件を満たす候補が存在しない．このため，ボックス (7) は空集合となり，以降の探索は枝狩りされる．

（8）については（5）からの候補のみとなるが，ここですべての要素が条件を満たさなくなり，探索プロセスは停止する．

Algorithm 1 SeqBDD 構造上の探索アルゴリズム

```
n_0 is a top node of the SeqBDD S;
n_0.cand = {{∅ 0 : 0}};
search_seqbdd(n_0);

function SEARCH_SEQBDD(n)
    if all of ancestors of n have not been processed yet then
return
    end if
    if n is a terminal node then return cand;
        // output candidates
    else
        n_1 = n.edge_1.dest; // n_1 : destination of 1 edge of n
        n_0 = n.edge_0.dest; // n_0 : destination of 0 edge of n
        n_1.cand = update_candidate(n.edge_1, n.cand, n_1.cand);
        n_0.cand = update_candidate(n.edge_0, n.cand, n_0.cand);
        // update candidates for edge 1 and 0
        n_1.cand = reduce(n_1.cand);
        n_0.cand = reduce(n_0.cand);
        // reduction of the candidate set
        // and check the constraints
        if n_1.cand is not NULL then return search_seqbdd(n_1);
        end if
        if n_0.cand is not NULL then return search_seqbdd(n_0);
        end if
    end if
end function
```

なお，図 3 で示した例では，制約を満たす解は存在しないが，図 4 で示した例では，1 つの結果，すなわち，$\{bcae\}$ （$\{bca\}$ とは編集距離 1）を得る．

なお，アルゴリズム中の $update\_candidate$ は現状の候補系列 $n_1.cand$ と $n_0.cand$ に対して，新しい候補を追加することを意味する．また，$reduce$ は候補集合を縮約し，制約検査を行うものである．

3.5 動的計画法

上述した編集距離に基づく文字列の探索を行うには，動的計画法を用いるのが一般的である．動的計画法により，文字の挿入や削除・置換によって，一つの文字列を別の文字列に変形するのに必要な手順の最小回数を求めることができる．例えば $bcde$ と bcd の例では，図 5 のように編集距離 1 であることがわかる．一方 $bcde$ と be に関しては，編集距離が 2 以上となり，編集距離 1 の条件をみたさない．

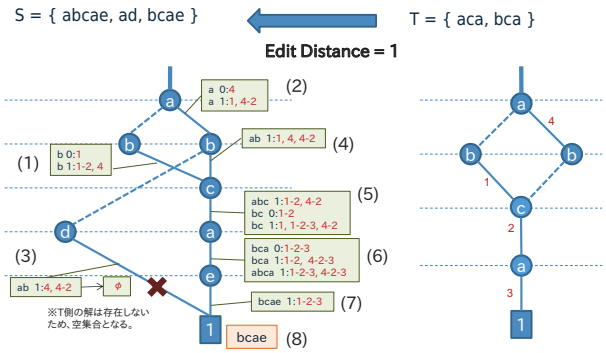

図 4　SeqBDD による類似結合 (2)

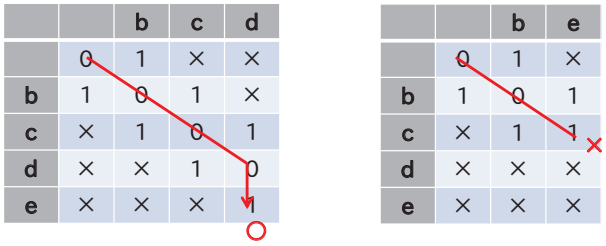

図 5　動的計画法による文字列マッチング

文字列間の類似結合においては，データベース間において考えられる全ての文字列間について動的計画法を適用し，編集距離を求めることになるが，ノードは SeqBDD で共有されているため，共有部分に関しては動的計画法の照合回数を削減させることが可能である．

以下，本節で示した SeqBDD に基づく文字列間類似結合を SeqBDD-Join と呼ぶ．SeqBDD-Join は，SeqBDD データ構造を構築し，それらをもとにした類似結合アルゴリズムを C++ により実装したものである．

4. 実装上の工夫

前章の動的計画法により，制約上必要なノードのみを対象として処理が進められるが，データベースの状態によっては他にも様々な観点で，効率的に枝刈りを行うことが可能である．以下，探索の効率化のために導入した探索手法・枝刈り戦略を示す．

4.1 最小距離増分による枝刈り

文字列を含むデータベース間で文字列数に比較的大きな差がある場合，明らかに長さが異なる文字列に対し事前に枝刈りを行うことが可能である．

例えば，編集距離 3 という条件のもと，$sapzapo$ と $zapo$ という文字列があった場合，動的計画法では図 6（左）の数字を記載したパターン全て計算しなければならない．しかし $sapzapo$

の s と $zapo$ の z の文字に着目すると，下位から数えてそれぞれ 7 文字目，4 文字目に対応しており，この文字間に 3 文字分の差があることは明らかである．

図 6（右）ではこの文字間の編集距離は 1 となるが，実際は編集距離 4（1+3）相当の差分があることがわかり，編集距離 3 の条件を満たさないことは明らかである．

我々は，この文字間の差分を最小距離増分と定義し，動的計画法で探索を行う前に，枝刈りを行う形で実装した．

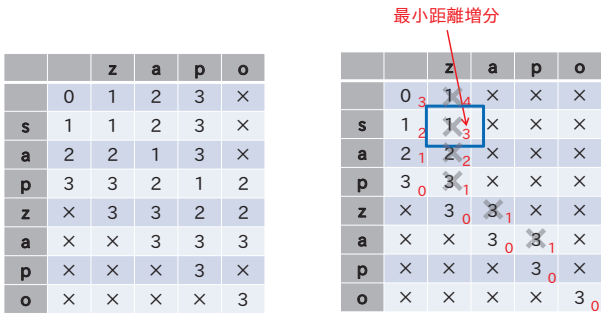

図 6　動的計画法での枝刈り

4.2　Hybrid 型による探索

ある文字列のデータベースにおいて，文字列の上位（前半）箇所が比較的類似しているデータベースでは，下位から探索を行った方が制約を満たさない組合せを早期に検出できるため，効率的である．逆に，下位（後半）箇所が比較的類似している場合は，上位から探索を行った方が効率的である．一般には，与えられた問題が上位が類似しているのか，下位が類似しているのか判断できないため，探索の過程で，TopDown 型，BottomUp 型の探索を組み合わせ，状況に応じて効率的に制御を行う Hybrid 型による探索を検討した．探索のイメージを図 7 に示す．図は 3. 章で示した，$\mathcal{S}$ 側のデータベースを表しており，TopDown からの探索に加えて，BottomUp からも同様の形で探索を行うものとする．Hybrid 型で効率的に探索を行う方法として，以下の 3 手法を実験した．

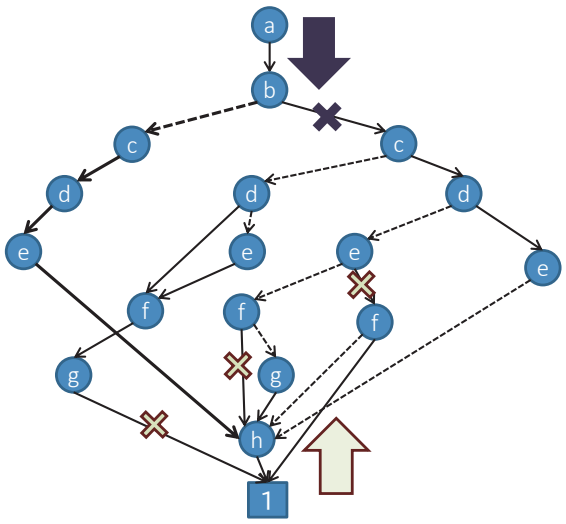

図 7　Hybrid 型による探索

a）　手　法　1

照合結果である候補数をもとにして，TopDown，BottomUp 用の昇順キューを用意し，TopDown，BottomUp の最小の要素を比較することで，候補数が少ない方のノードから探索する．例えば，TopDown の最小候補数が 6 で BottomUp の最小候補数が 5 の場合，BottomUp のノードから探索を行う．

b）　手　法　2

手法 1 で探索が極度に片方向に偏らないようにゆらぎを与える．例えば比率を 0.3 に設定すると，全体の 3 割は候補数が多い側から探索されることとなる．

c）　手　法　3

関連する候補の編集距離の平均値が大きいノードから探索する．手法 1 の候補数の代わりに，編集距離の平均をキーとする．TopDown，BottomUp 用のキューを用意し，編集距離の平均をキーに降順に格納し，手法 1 と同様の形で TopDown，BottomUp 間で比較を行う．

5.　Trie-Join との比較評価

本システムでの結果と Trie-Join [6] [13] での結果の比較を以下に示す．本実験では，乱数を用いて人工的に生成された長さ 10 の文字列（アルファベット（“a”–“z”））を実験データとして使用する．表 1 に，Trie-Join との比較実験で使用したデータの例を表わす．

表 1　比較実験に使用したデータ例

Input Data1	Input Data2
aegklorstw	aegklprstw
bcegjmtvxy	bcegjmtvxz
filmrsuxjr	filmrsuxjr
jyaeinprst	jfaeinprsz
laefghlqvx	ohjygqbzfa
pohkqrvwyh	tgqqoktabd
ykdijkmqrt	xuqartlhns
zcgknoruxy	zcgkmoruxy

データは 100,000，500,000，1,000,000 文字列からなる 3 セットを用意し，編集距離 1 の制約のもと，文字列間でマッチングを行った．

SeqBDD-Join ならびに Trie-Join [(注1)] による実行結果を表 2 に示す．なお，以降の実験は，いずれも Red Hat Enterprise Linux Server 6.5 上で，128 Intel Xeon CPUs (2.13 GHz) ならびに 2.048 TB RAM を用いて実行したものである．

表 2 の探索時間を見る限り，絶対値は Trie-Join の方が上回っているが，Ratio(100,000 の規模をベースに，文字列の規模と探索時間の比率を表したもの) に関しては，SeqBDD-Join の方がやや小さい値となっていた．さらに文字列の規模が大きくなるにつれ，SeqBDD-Join の探索時間がより向上する可能性が考えられる．

続いて前節で述べた実装上の工夫に関しての実験結果を表 3 に示す．最小距離増分の手法を考慮した場合としていない場合を測定した結果，考慮した場合の方が約 20 %性能が改善された．2 つのデータベースの各ノード間で文字列の長さに差があ

(注1)：Trie-Join プログラムは以下にあるものを使用した．
http://dbgroup.cs.tsinghua.edu.cn/wangjn/codes/triejoin.tar.gz

表 2　SeqBDD-Join と Trie-Join との実行比較（Edit Distance=1）

SeqBDD-Join

Size of DB1	Size of DB2	Num. of Results	Traverse.Time(sec)	Ratio
100,000	100,000	4	7.5	1.0
500,000	500,000	22	55.4	7.4
1,000,000	1,000,000	35	119.3	15.9

Trie-Join

Size of DB1	Size of DB2	Num. of Results	Traverse.Time(sec)	Ratio
100,000	100,000	4	3.1	1.0
500,000	500,000	22	30.0	9.7
1,000,000	1,000,000	35	56.6	18.3

ることにより，最小距離増分の枝刈りの効果が有効であったことがわかる．

続いて Hybrid 型に関する実験結果を示す．上記と同一環境で，10 文字からなる文字を 10,000 文字列用意し編集距離 1 の制約のもと，文字列間でマッチングを行った．

データベースは以下の 3 種類を用意し，各データベースに対し，TopDown のみ，BottompUp のみ，Hybrid 型の 3 パターンで探索を行い，探索ノード数の違いを調べた．

- 上位が異なるデータベース
- 下位が異なるデータベース
- ランダムデータベース

なお，Hybrid 型については前章に記載した 3 手法を確認した．表 4 の探索ノード数を見る限り，上位，下位が異なるデータに関しては，いずれの Hybrid 型手法も，TopDown のみ，BottompUp のみの場合の中間の探索ノード数となった．特に下位が異なるデータに関しては，手法 1，手法 3 の探索ノード数が大幅に削減された．一方上位が異なるデータに関しては，手法 2 が最も小さい値であった．ランダムデータに関しては，いずれの Hybrid 型手法も大差はなかった．

以上の結果から，一般に Hybrid 型は従来の TopDown 方式に比べて探索ノード数が多くなる可能性があるが，下位が異なるデータに関しては，有効に働く可能性がある．また手法 2 のようなゆらぎを与えることで，探索の偏りを抑制することも確認した．

6. まとめと今後の課題

本研究では，SeqBDD をベースにして，集合間文字列照合を行うアルゴリズムを構築した．文字列間類似結合に対して，動的計画法に基づき効率的に文字列間類似検索を行う手法を提案するとともに，明らかに長さが異なる文字列に対し事前に枝刈りを行う，最小距離増分の手法を実装した．

文字列間類似結合において効率的な実装として知られている Trie-Join と比較し，文字列の規模が大きくなるにつれ，SeqBDD-Join の方がより探索時間がより向上する可能性のある例を示した．

今後の課題は以下のとおりである．

- 編集距離が増大した場合には，Trie-Join においても，SeqBDD-Join においても，管理すべき候補集合が増大し，それが大きなオーバーヘッドになることがわかっている．一方フィルタリング手法では，与えられた編集距離に依存したさまざまな枝刈り処理を行うため，一般的には大きな値をもつ編集距離に対しては，フィルタリング手法の方が有利であるといわれている．今後，より大きな編集距離に対して，効率的な枝刈りを可能とする探索手法について検討を行う．
- Hybrid 型による 3 つの最適化手法を提示し，データに偏りがある場合 Hybrid の手法が有効に働く可能性を示した．最適化方法を工夫し，ランダムデータに関しても改善できないか検討する．
- SeqBDD（ZDD）は，演算の結果や途中結果を圧縮したまま保持し，必要に応じて保存・取り込みが可能である．また構造を保持したまま，集合間の和や積などの演算を行うことができる．文字列間照合の現実的な応用では，複数のデータベース（たとえば日々のトランザクション）間での集合演算が必要と考えられるため，こうした応用面への適用についても今後検討を進めたい．

文　献

[1] A. Arasu, V. Ganti, R. Kaushik : Efficient Exact Set-Similarity Joins, In Proc. of 32nd International Conference on Very Large Data Bases (VLDB 2006), 2006

[2] R. J. Bayardo, Y. Ma, R. Srikant : Scaling Up All Pairs Similarity Search, In Proc. of 16th international conference on World Wide Web, 2007

[3] R. E. Bryant : Graph-based algorithms for Boolean function manipulation, IEEE Transactions on Computers, Vol. 35 Issue 8, 1986

[4] S. Chaudhuri, V. Ganti, R. Kaushik : A Primitive Operator for Similarity Joins in Data Cleaning, In Proc. of 22nd International Conference on Data Engineering (ICDE '06), 2006

[5] 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, DEIM Forum 2010 E3-4, 2010

[6] J. Feng, J. Wang, G. Li : Trie-join: a trie-based method for efficient string similarity joins, The VLDB Journal 21:437.461, 2012

[7] K. Gouda, M. Rashad : PreJoin: An Efficient Trie-based String Similarity Join Algorithm, The 8th International Conf. on Informatics and Systems (INFOS), 2012

[8] D. E. Knuth : The Art of Computer Programming, Vol. 4, No.1, Bitwise Tricks & Techniques, pp.117-126, Addison-Wesley, 2009

[9] E. Loekito, J. Bailey, J. Pei : A Binary decision diagram based approach for mining frequent subsequences, Knowledge and Information Systems, 24(2), 2010

[10] S. Minato : Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems, In Proc. of 30th ACM/IEEE Design Automation Conference (DAC'93), 1993.

[11] S. Minato : Implicit Manipulation of Polynomials Using Zero-Suppressed BDDs, In Proc. of IEEE The European Design and Test Conference, 1995.

[12] 白井康之，高嶋宏之，鶴間 浩二，小山 聡：ゼロサプレス型二分決定木を用いた集合間類似結合, DEIM Forum 2013, F10(デー

表 3　最小距離増分の効果の比較（Edit Distance=1）

SeqBDD-Join

Size of DB1	Size of DB2	Min Distance Increment	Num. of Results	Exec.Time(sec)
1,000,000	1,000,000	On	35	119.3
		Off	35	149.5

表 4　Hybrid 型による探索

上位が異なるデータ

Patern	Num. of Top	Num. of Bot	Num. of Total
TopDown Only	29,660	−	29,660
BottomUp Only	−	45,502	45,502
Hybrid（手法 1）	2,321	42,147	44,468
Hybrid（手法 2）	23,730	9,462	33,192
Hybrid（手法 3）	14,732	27,880	42,612

下位が異なるデータ

Patern	Num. of Top	Num. of Bot	Num. of Total
TopDown Only	45,859	−	45,859
BottomUp Only	−	29,674	29,674
Hybrid（手法 1）	3,436	27,564	31,000
Hybrid（手法 2）	36,379	6,414	42,793
Hybrid（手法 3）	1,762	28,816	30,578

ランダムデータ

Patern	Num. of Top	Num. of Bot	Num. of Total
TopDown Only	33,165	−	33,165
BottomUp Only	−	33,026	33,026
Hybrid（手法 1）	2,563	30,634	33,197
Hybrid（手法 2）	26,524	7,043	33,567
Hybrid（手法 3）	4,266	29,241	33,507

タベースコア技術), 2013

[13] J. Wang, J. Feng, G. Li : Trie-Join: Efficient Trie-based String Similarity Joins with EditDistance Constraints, In Proc. of the VLDB Endowment, 3(1-2), 2010

[14] C. Xiao, W. Wang, X. Lin : Ed-join: an efficient algorithm for similarity joins with edit distance constraints, In Proc. of VLDB Endowment, 1(1), 2008