情報処理学会論文誌　データベース　Vol.8　No.2　57–67 (June 2015)

幾何学的接尾辞木の高速処理方式

高橋 誉文[1,a)]　田村 慶一[1,b)]　黒木 進[1,c)]　北上 始[1,d)]

受付日 2014年12月21日, 採録日 2015年4月6日

概要：ディスク上に蓄積された幾何学的接尾辞木は，蛋白質立体構造データベースに対する高速な類似検索のための索引構造として利用可能である．しかしながら，蛋白質立体構造データベースの大規模化により，その索引構造の構築に多くの時間を費やすだけではなく，索引構造を用いた類似検索に要する時間にも影響を与える．本論文では，幾何学的接尾辞木の高速処理を実現するために，幾何学的接尾辞木の構築方法の改良および構築処理と検索処理の双方をマスタワーカ法や分散型ワーカ法を用いて並列化する方法を提案する．構築方法の改良では，幾何学的接尾辞木の従来の構築法が並列化に直接向いていないという点に着目し，並列化をする前に，あらかじめ，この構築方法を従来の座標配列を 1 本ずつ処理する逐次構築法から全データをまとめて処理するトップダウン構築法に変更している．また，データページを管理するバッファ管理法の変更も行っている．さらに，構築と検索のそれぞれに対して，データ分割法とタスク分割法による並列化を実施し，並列性能を評価している．実験により並列性能を測定した結果，幾何学的接尾辞木の並列処理においては，マルチバッファ・マスタワーカ法がシングルバッファ・マスタワーカ法や分散型ワーカ法よりも優れていることが判明した．

キーワード：データベース，蛋白質，立体構造検索，幾何的接尾辞木

High-speed Processing Method for Geometric Suffix Tree

Yoshifumi Takahashi[1,a)]　Keiichi Tamura[1,b)]　Susumu Kuroki[1,c)]
Hajime Kitakami[1,d)]

Received: December 21, 2014, Accepted: April 6, 2015

Abstract: Geometric suffix trees stored on a disk can be used as indices to achieve a high-speed similarity search on large-scale, 3-D protein structure databases. However, due to an increasing number of data records in the protein structure database, a large amount of time must be taken to construct the index, greatly affecting the search speed. This paper proposes a high-speed processing method using a master worker and distributed worker methods in order to parallelize improvement of construction methods and both the construction processing of the index and the similarity search processing using the index constructed in the protein structure database. In the improvement of construction methods, the authors focus on the viewpoint that the existing construction method of the geometric suffix tree is not suitable for direct parallelization. Therefore, beforehand, the authors changed the existing sequential construction method to process one by one the data in the database into a top-down construction method to process all of the data together in the database. Moreover, the authors changed the existing buffer control method into an adaptive method for parallel processing. In order to parallelize both the construction and the similarity search, both data partition and task partition methods are applied to a given problem, such as the construction and similarity search, and sub-problems generated from the given problem are executed in parallel using each worker model. The performance evaluation experiments resulted as follows: in the parallel processing, a multi-buffer master worker method was better than a single-buffer master worker method and the distributed worker method.

Keywords: database, protein, similar structure search, geometric suffix tree

[1] 広島市立大学大学院情報科学研究科
Graduate School of Information Sciences, Hiroshima City University, Hiroshima 731–3194, Japan
[a)] dt65002@e.hiroshima-cu.ac.jp
[b)] ktamura@hiroshima-cu.ac.jp
[c)] kuroki@hiroshima-cu.ac.jp
[d)] kitakami@hiroshima-cu.ac.jp

1. はじめに

蛋白質は生命活動の生体機能にかかわる重要な物質であり，生命科学の研究や創薬対象となっている．蛋白質は，アミノ酸配列が異なっていても，立体構造と生体機能の間には密接な関係があることが分かっている．近年，蛋白質の立体構造データが急増してきているため，蛋白質立体構造の類似部分構造を高速に見つけ出す方法の研究がさかんに行われている．その代表的な方法として，ディスク上に幾何学的接尾辞木を構築し，それを利用する類似構造検索の方法が提案されている [1], [2], [3], [4], [5], [6].

しかしながら，幾何学的接尾辞木の構築と検索の並列化による高速化の研究がまだ行われていない．与えられた問題を並列化するために，その問題を分割する方法（問題分割法）をはじめとして，問題分割によって生成されるタスクの負荷分散法を明らかにすることが重要である．問題分割法には，データ分割法とタスク分割法が知られているが，どちらも問題を分割する方法のみに関心があるだけで，プロセッサの稼働率を 100%に近づけることを保証していない．このため，問題分割法を明らかにしたら，稼働率を 100%に近づけるための負荷分散法について検討する必要がある．

しかしながら，従来の幾何学的接尾辞木では，逐次構築法（複数の座標配列のそれぞれを幾何学的接尾辞木に逐次追加する方法）を用いているため，十分な並列性能を得られないという問題がある．

本論文では，このような問題を解決する方法を提案する．提案手法では，幾何学的接尾辞木の構築と検索を並列化するために，以下の 2 つの仕組みを持っている．

(1) トップダウン構築
逐次構築法により幾何学的接尾辞木を構築する過程では，データの修正が頻繁に発生する．このデータの修正回数の削減と，十分な並列性能を確保するため，幾何学的接尾辞木の構築方法をトップダウン構築法に変更して座標配列の全件をまとめて構築する．また，逐次構築法からトップダウン構築法への変更にともない，中間ノードページのバッファ管理法を TOP-Q から LRU に変更し，隠れ配列ページのバッファ管理法を行わないように変更する．

(2) 並列化による高速処理
幾何学的接尾辞木に対するデータ分割法やタスク分割法による問題分割法のほかに，マスタワーカ法や分散型ワーカ法による負荷分散法を組み合わせた並列化を行い，幾何学的接尾辞木の構築と検索を高速化する．

本論文の構成は以下のとおりである．2 章では接尾辞木の関連研究について述べる．3 章では，従来手法であるディスクベースの幾何学的接尾辞木について述べる．4 章では，提案手法について述べる．5 章では提案手法の有効性を示すための実験による評価を行い，6 章では本論文のまとめを行う．

2. 関連研究

文字配列に対する接尾辞木の概念は 1973 年，Weiner [7] によって初めて紹介された．その後，1976 年 McCreight ら [8] によって構築法を単純化され，1995 年には Esko が線形時間で接尾辞木を構築するアルゴリズム [9] を紹介した．しかしながら，大規模な文字列データで接尾辞木を構築するには，その文字列データよりも多くの記憶領域を必要とするだけではなく，1970 年代のコンピュータ性能では接尾辞を構築・利用することが現実的ではなかったため，接尾辞木に関する研究がさかんではなかった．それ以後，コンピュータの性能向上とディスクの記憶容量の低価格化が劇的な速さで進んだため，実用化を意図した接尾辞木に関する多くの研究がさかんに行われてきた [10], [11]．特に，接尾辞木をメモリからディスク上へと展開する研究 [12], [13], [14] は 2001 年頃から注目され，接尾辞木の並列化の研究 [15], [16], [17], [18], [19], [20], [21] は，2006 年頃から注目されるようになった．ゲノム規模の文字配列に対する接尾辞木のディスクに基づく研究 [15], [22] は，2007 年頃から行われている．

これに対して，蛋白質立体構造に対する接尾辞木 [1], [2], [3], [4], [5], [6] は，2 つの座標点集合どうしを RMSD [23], [24], [25], [26], [27], [28] により空間的に重ね合わせる方法を用いた方法であり，2004 年頃から研究されている．この接尾辞木は，幾何学的接尾辞木と呼ばれ，データを辺に格納する方法 [1], [2] とノードに格納する方法 [3], [4], [5], [6] が研究されている．前者の格納方法は，メモリ上で幾何学的接尾辞木を構築する研究で利用されているが，後者はディスクに基づく幾何学的接尾辞木の研究で使用されている．しかしながら，幾何学的接尾辞木の高速化の研究はまだ行われていない．なお，前者に比べて後者の格納方法が考えやすいため，本論文の幾何学的接尾辞木の高速化では，後者の格納方法に着目している．

3. 幾何学的接尾辞木

本章では，まず構造間の比較に用いられる非類似度の尺度について述べる．次に，データをノードに格納するノード格納法 [3], [4], [5], [6] の構成について説明する．その中では，検索時間の短縮のための隠れ配列の仕組みやディスク上に木データを保存するためのバッファ管理法についても述べる．以上をふまえ，最後に幾何学的接尾辞木の問題点について述べる．

3.1 接尾辞木

文字列の接尾辞木 [7], [8] とは与えられた文字列 S の接尾辞の集合をトライ構造で表現したデータ構造である．文

字列 S は，あるアルファベット Σ 上で定義された有限の文字の並びであり，S の終端記号を \$ とするとき，S の接尾辞木は，S\$ のサフィックスをもとに構築されたトライ木である．ただし，$\$ \notin \Sigma$ を満たし，S\$ は S の最右端に終端記号 \$ を追加した文字列とする．この木の各葉は文字列 S の接尾辞 ST の 1 つを表現し，根から葉に至るパス上の各ノードは ST の部分文字列を表す．これ以降，文字列および座標配列を $\langle\rangle$ で囲んで表記する．

3.2　非類似度の尺度

幾何学的接尾辞木は，蛋白質を構成する各アミノ酸のアミノ基（-NH_2）とカルボキシル基（-COOH）の両方と結合している炭素原子（以下では C_α 原子と呼ぶ）を，蛋白質のアミノ基側である N 末端からカルボキシル基側である C 末端に並べた座標配列をもとに構築された接尾辞木である．

幾何学的接尾辞木に対する類似構造検索では，座標配列間の非類似度の尺度として，2 つの座標配列間の幾何学的な非類似度を決定する手段の 1 つである RMSD（root mean square deviation）[23], [24], [25], [26], [27], [28] が利用されているが，RMSD を類似構造検索に直接利用すると，接尾辞木の深さ方向に関する単調性が保証されない．このため，RMSD の代わりに，単調増加である MSSD（minimum sum squared distance）を利用することで，単調性を保証している [1], [2].

2 つの座標配列を $Q = \langle q_1, q_2, \cdots, q_m\rangle$ および $T = \langle t_1, t_2, \cdots, t_m\rangle$，座標配列の回転行列を R，移動ベクトルを v とすると，RMSD は

$$E(Q,T,R,v) = \sqrt{\sum_{i=1}^{m} \left| q_i - (R \cdot t_i + v) \right|^2 / m}$$

を最小化した以下の式である．

$$\mathrm{RMSD}(Q,T) = \min_{R,v}\{E(Q,T,R,v)\}$$

また，MSSD は以下のとおりである．

$$\mathrm{MSSD}(Q,T) = m(\mathrm{RMSD}(Q,T))^2$$

幾何学的接尾辞木では，2 つの座標配列の接尾辞の先頭からの間の MSSD の値と分岐の閾値 b とを比較し，b 以下であれば類似構造として同じ辺で表し，閾値 b を超えたとき，非類似構造として辺を分岐させている．本論文では座標配列の接尾辞を接尾辞座標配列と呼ぶ．一般に，中間ノード $node$ に記録されるべき接尾辞座標配列は，複数存在するが，接尾辞木構築中の分岐処理に最初に利用された接尾辞座標配列を代表として記録している．

3.3　ノード格納法

幾何学的接尾辞木とは，座標配列に対する接尾辞木を構築したものである．幾何学的接尾辞木で 2 つの座標配列の接尾辞が辺を共有する条件は，先頭からの間の MSSD の値が分割パラメータ b 以下となることである．

表 1　座標配列データベースの例
Table 1　Example of coordinate sequences database.

ID	座標配列
1	$\langle(3,5,2),(4,6,3),(4,5,3),(7,7,4),(4,4,2)\rangle$
2	$\langle(2,5,1),(3,5,3),(2,7,4),(6,7,3),(3,4,2)\rangle$

複数本の座標配列のそれぞれに識別子 ID が振られているとするとき，接尾辞木の各ノード $node_k$ で $[(i,j), length(node_k) - 1, Count_k]$ のデータを記録する．(i,j) は ID が i の座標配列について，先頭から j 番目の座標点から始まる接尾辞座標配列の位置情報を意味する．以後，この接尾辞座標配列を $S_{(i,j)}$ と表記する．$length(node_k)$ は，根ノード $root$ からノード $node_k$ までの経路に対応する座標配列の長さを意味し，$length(node_k)-1$ は幾何学的接尾辞木の各ノード $node_k$ までの座標配列の座標間の数を意味する．その $node_k$ の代表となる座標配列は $S_{(i,j)}[1..length(node_k)]$ と表記される．$Count_k$ は，この $node_k$ 以下に接続される葉ノードの総数を意味する．

3.3.1　隠れ配列を用いた類似構造検索手法

幾何学的接尾辞木の検索では検索キーの長さに対応する深さの中間ノードまで木の探索を行っても，葉ノードを使って類似部分構造を検索するためには，その中間ノード以下に存在するすべての葉ノードを取り出すために，検索キーの長さより長い接尾辞座標配列を持つ葉ノードの探索を行わなければならない．すべての部分構造を調べるためには複数の中間ノードと葉ノードの探索が必要となり，子の探索には時間がかかる．しかし，その中間ノード以下に存在するすべての葉ノードの情報を，隠れ配列として持たせておくことで検索処理を削減できる．隠れ配列は，そのノードに含まれる部分構造の中でノード情報に含まれないすべての部分構造である．隠れ配列は（ID，参照開始位置）として表し，その集合を H と定義する．$node_k$ の隠れ配列の集合を H_k，H_k に含まれる隠れ配列 (i,j) を $H_{k,(i,j)}$ と定義する．構築時に中間ノードを通過するとき，その中間ノードの隠れ配列の集合 H に追加し，辺を分割するときには子ノードの隠れ配列を新しいノードにコピーする．

たとえば，**表 1** の 2 本の座標配列を用いて，MSSD の閾値を $b = 7$ とするとき，**図 1** に示されるような幾何学的接尾辞木が得られる．

この例で構築された幾何学的接尾辞木は，7 つの中間ノードと隠れ配列，そして，8 つの葉ノードからなる．図中の $node_2$ に記録されているノード情報 $[(1,1),2,3]$ の $(1,1)$ は，ID が 1 の座標配列で 1 番目の座標点から始まる接尾辞配列 $S_{(1,1)} = \langle(3,5,2),(4,6,3),(4,5,3),(7,7,4),(4,3,2)\rangle$ であることを意味する．

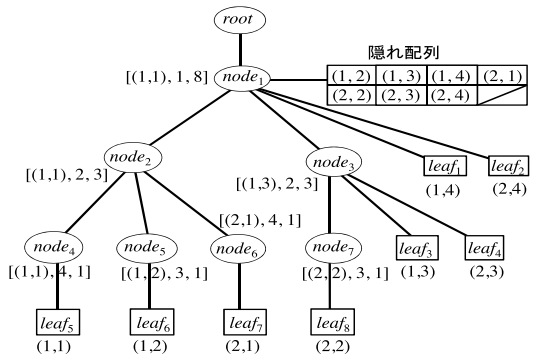

図 1　ノード格納法による幾何学的接尾辞木の構築例
Fig. 1　Example of construction of the geometrical suffix tree by the node storage method.

ノード情報 $[(1,1),2,3]$ の 2 は $node_2$ における長さ $length(node_2)-1=2$ を表し，接尾辞配列 $(1,1)$ の 1 番目から 3 番目までの部分座標配列を表現する座標配列 $S_{(1,1)}[1..3]=\langle(3,5,2),(4,6,3),(4,5,3)\rangle$ であることが分かる．ノード情報 $[(1,1),2,3]$ の 3 は $node_2$ の隠れ配列 H_2 内のデータ数と，$node_2$ 以下に接続される葉ノードの総数がそれぞれ 3 であることを意味する．また，葉ノード $leaf_4$ に記録されている $(2,3)$ は，ID が 2 の座標配列 $\langle(5,5,1),(3,5,3),(2,7,4),(6,7,7),(3,4,2)\rangle$ の 3 番目の座標点 $(2,7,4)$ から始まる接尾辞配列 $S_{(2,3)}=\langle(2,7,4),(6,7,7),(3,4,2)\rangle$ を意味する．

図 1 の幾何学的接尾辞木の検索を行い，$node_2$ で検索キーの長さまでの探索が終わったとする．その後，葉ノードを用いた検索を行うと，$node_2$ には $(1,1)$，$(1,2)$，$(2,1)$ が含まれることが分かるが，$node_2$ 自体は $(1,1)$ しか存在しない．すべての部分構造を調べるためには 3 つの中間ノードと 3 つの葉ノードの探索が必要となり，子の探索には時間がかかる．しかし，隠れ配列を用いた検索を行うと，$node_2$ では，$(1,2)$，$(2,1)$ を隠れ配列として保持しており，これを調べることで中間ノードの探索が削減できる．

構築されている幾何学的接尾辞木から，長さ m の検索キーを Q とし，許容誤差を ϵ とするとき，これらの問合せ条件が満たされる類似部分構造（部分座標配列）をすべて検索する手順は以下のとおりである．また，$R_k[1..m]$ は $node_k$ での $S_{(i,j)}[1..m]$ である．

(1) 根から木をたどり，以下の条件をすべて満たす $node_k$ をすべて見つける．
　① $length(node_k)\geq m$
　② $\mathrm{MSSD}(Q,R_k[1..m])\leq m\times(\epsilon+\sqrt{\frac{b}{m}})^2$
(2) 中間ノード $node_k$ とそこに存在する隠れ配列 H_k を調べ，$\mathrm{RMSD}(Q,U[1..m])\leq\epsilon$ を満たす部分構造 $U[1..m]$ をすべて見つける．ただし，U は $node_k$ と隠れ配列 H_k 内に含まれる接尾辞座標配列とする．

検索キー $Q=\langle(2,4,1),(5,5,3),(4,7,5)\rangle$，RMSD の許容誤差 $\epsilon=2.0$ で検索を行う．まず，$node_1$ まで探索を行うと，$length(node_1)=2<3$ かつ $length(node_2)=length(node_3)=3\geq 3$ により，$node_2$ および $node_3$ は，上記の (1) ① の条件を満たす（$k=1$，$l\in\{2,3\}$，$m=3$）．また，$R_2=\langle(3,5,2),(4,6,3),(4,5,3)\rangle$，$R_3=\langle(4,5,3),(7,7,4),(4,4,2)\rangle$，$\mathrm{MSSD}(Q,R_2[1..3])=5.33$，$\mathrm{MSSD}(Q,R_3[1..3])=6.59$，$b=3(2.0+\sqrt{(7/3)})^2=7.53$ により，$node_2$ および $node_3$ は上記 (1) ② の条件を満たす．

次に，上記 (2) の処理に入ると，$node_2$ の隠れ配列 H_2 は，$\{(1,2),(2,1)\}$ であり，$node_3$ の隠れ配列 H_3 は，$\{(2,2),(2,3)\}$ であることが分かる．$node_2$ の $(1,1)$ と $node_3$ の $(1,3)$ も含めたこれらの合計 6 本の接尾辞座標配列のそれぞれについて，Q との RMSD を計算すると，$\mathrm{RMSD}(Q,S_{(1,2)}[1..3])=1.41$，$\mathrm{RMSD}(Q,S_{(2,2)}[1..3])=1.11$，$\mathrm{RMSD}(Q,S_{(1,1)}[1..3])=1.68$, $\mathrm{RMSD}(Q,S_{(2,1)}[1..3])=0.81$, $\mathrm{RMSD}(Q,S_{(1,3)}[1..3])=2.08$，$\mathrm{RMSD}(Q,S_{(2,3)}[1..3])=1.10$ となる．許容誤差以内の部分座標配列は，$S_{(1,2)}[1..3]$, $S_{(2,2)}[1..3])$, $S_{(1,1)}[1..3])$, $S_{(2,1)}[1..3]$，$S_{(2,3)}[1..3]$ の 5 件であり，これらが問合せ Q に対して許容誤差 $\epsilon=2$ 以内にある類似部分構造となる．

隠れ配列を用いた検索方法では，葉ノードまでの木の探索に必要な $node_4$ から $node_7$ の参照の削除，および，$node_2$ 以下の葉ノード $leaf_5$，$leaf_6$，$leaf_7$ と，$node_3$ 以下の葉ノード，$leaf_3$，$leaf_4$，$leaf_8$ の 6 回の参照を隠れ配列 H_2 と H_3 の 2 回に減少することができる．

以下では構築処理と検索処理の時間計算量について説明する．なお，RMSD と MSSD の計算処理を加えても計算時間は変わらないので，それらの説明を省略する．

構築処理の時間計算量については，長さ m の座標配列を幾何学的接尾辞木に追加するために，$m-1$ 回ノードとの比較を行う．その座標配列のすべての接尾辞に対して同様に比較を行うので $(m-1)\times(m-1)$ 回，さらにすべての座標配列 n 件に対して同様に比較を行うので $(m-1)\times(m-1)\times n$ 回となる．幾何学的接尾辞木のノードの作成と修正処理は $O(1)$ なので，構築処理における時間計算量は $O((m-1)\times(m-1)\times n)=O(nm^2)$ となる．幾何学的接尾辞木のデータ量は MSSD の閾値によりできあがる木が変わり，ノードでは MSSD の閾値が非常に大きい場合の座標配列の長さである $O(m)$ と，MSSD の閾値が非常に小さい場合の入力データの全座標数である $O(nm^2)$ の間に，隠れ配列は入力データの全座標数である $O(nm^2)$ となる．なお，全座標配列を 1 本の座標配列につなげた場合について考えると，$n=1$ となるので，文献 [2] の計算量 $O(m^2)$ と一致する．

検索処理の時間計算量については検索キーの長さを l とすると，素朴な方法では全座標配列に対して検索キーを 1 座標ずつずらしてすべての組合せで比較しているので，$O(n\times(m-l+1))=O(nm)$ となる．幾何学的接尾辞木

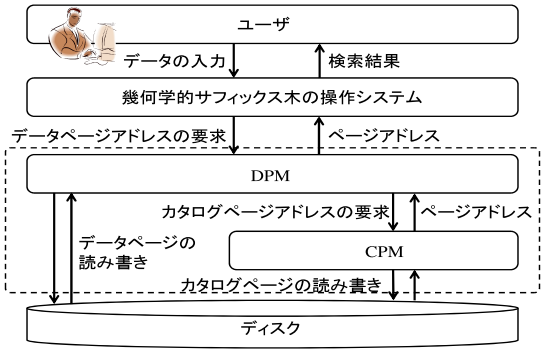

図 2　システム構成図
Fig. 2　System configuration diagram.

の探索処理では最小が $O(l)$ となり最大が全ノードを探索する $O(nm^2)$ となる．そして探索によって検索キーと類似していると判断されたノードの隠れ配列を調べる．RMSD と MSSD の閾値が大きくなると素朴な方法の計算時間に近づくので，検索時間が素朴な方法より遅くなる．

3.3.2　バッファ管理

従来のディスクベースの幾何学的接尾辞木 [3], [4], [5], [6] は，幾何学的接尾辞木をディスク上に構築したものであり，これを用いて，類似構造検索を行うことができる．**図 2** にディスクベースの幾何学的接尾辞木のシステム構成を示す．点線で囲まれた範囲がバッファ管理の動作である．バッファ管理は，DPM（データページ管理モジュール）と CPM（カタログページ管理モジュール）の 2 つのモジュールから構成される．DPM はメモリ上のデータページのアドレスの管理とディスクとの読み書き，CPM はメモリ上のカタログページのアドレスの管理とディスクとの読み書きを行う．なお，データページには．幾何学的接尾辞木のデータが格納され，カタログページには，データページを管理するためのページテーブルが格納されている．ページテーブルは，すべてのデータページのディスク領域とメモリ領域それぞれのアドレスを持ち，バッファ領域 1 つにつきページテーブルは 1 つである．

幾何学的接尾辞木をディスクベースで扱うために必要なバッファ管理法について説明する．

図 2 のデータページとは中間ノード，葉ノード，隠れ配列を格納したページである．幾何学的接尾辞木のデータである中間ノード，葉ノード，隠れ配列のすべてをディスク上に格納しながら幾何学的接尾辞木の構築，検索を行う．各データページとカタログページのバッファ管理の方式を**表 2** で示す．LRU とは，バッファ領域のページの優先度を参照された時間で管理する方法である．この方法はバッファ領域に空きがないとき，バッファ領域のページの中で参照された時間が最も古いページを削除する．TOP-Q [29] とは，バッファ領域のページの管理を二分ヒープで行う TOP と，TOP から削除されたページを持つキューを組み合わせた方法である．この方法は，幾何学的接尾辞木の深さを優先度として持たせることで，幾何学的接尾辞木の浅いデータをバッファ領域に優先的に確保できる．

表 2　逐次構築法のバッファ管理方式
Table 2　Buffer management scheme of each page.

ページ名		構築処理	検索処理
データページ	中間ノードページ	TOP-Q	TOP-Q
	葉ノードページ	LRU	-
	隠れ配列ページ	LRU	-
カタログページ		LRU	LRU

3.4　幾何学的接尾辞木の問題点

従来手法であるノード格納法による逐次構築法の幾何学的接尾辞木の構築処理と検索処理に対する高速化を並列化で対処するには 2 つの問題点がある．1 つ目の問題は，逐次構築法ではノードや隠れ配列の情報の修正回数がデータの大規模化にともない増大することがあげられる．逐次構築法で図 1 の幾何学的接尾辞木を構築すると，$node_2$ は接尾辞 $(1,1)$ と $(1,2)$ を幾何学的接尾辞木に加えたときに作成される．その後すべての接尾辞が追加されるまでの間に，接尾辞 $(2,1)$ を追加することでノード情報と隠れ配列情報の修正がそれぞれ 1 回行われる．このような例でも明らかだが，さらに，大規模なデータで幾何学的接尾辞木を構築すると，ノード情報と隠れ配列情報の修正回数が増大する．また，現在の幾何学的接尾辞木では，各ノードに含まれるかどうかはノード情報の接尾辞を基準にしている．幾何学的接尾辞木から座標配列データの削除を行う場合を考えると，隠れ配列からの削除は簡単に行うことができる．しかし，ノード情報の接尾辞を削除する場合には，削除されたノード情報の代わりに隠れ配列の中から他の接尾辞をノード情報に使用する．しかし，その場合には新しいノード情報と類似しない接尾辞が隠れ配列内に存在する場合がある．よって，一度構築に使用した座標配列データの削除を行うことができない．蛋白質立体構造データベースではデータの修正や削除が起こるため，データベースの更新に合わせて幾何学的接尾辞木を更新しようとすると，最初から作り直すことになる場合がある．

2 つ目の問題は，問題分割法（データ分割法とタスク分割法）と負荷分散法（マスタワーカ法と分散型ワーカ法）の組合せが複数あるため，どの組合せによる並列化が効率的であるのかという点が明らかにされていないことである．

4.　提案手法

本章では，木を改良して高速な検索を行うために，従来の幾何学的なサフィックス木の 2 つの問題点を解決する方法について提案する．

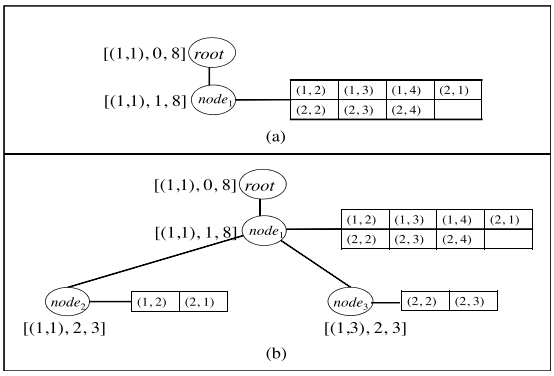

図 3　トップダウンによる構築方法
Fig. 3　Construction method of top-down.

4.1　構築方法の改良

構築方法の改良を行うために，トップダウン構築法とバッファ管理法の変更について述べる．

4.1.1　トップダウン構築法

ノード情報と隠れ配列情報の修正回数を削減するために，従来の構築方法を改良する方法について説明する．従来の構築方法では逐次構築法を行っているが，本研究では座標配列全件をまとめてトップダウンで構築を行う．構築処理で親ノードに存在するすべての子ノードを作成することを，1 つの処理単位としてタスクと呼ぶ．タスクは，タスクプールと呼ばれるタスクを管理する領域におく．タスクの処理は，親ノード $node_k$ の隠れ配列を見ることで部分構造の長さが $length(node_k)+1$ になる子ノードの作成を行う．この方法では，作成されたノード情報と隠れ配列情報に対する修正を行うことがない．

また，従来手法では葉ノードと隠れ配列の両方を用いているが，$node_k$ での隠れ配列 H_k とそのノード以下に存在するすべての葉ノードでは，同じ接尾辞の情報を含んでいる．そのため，幾何学的接尾辞木の検索処理に隠れ配列を用いることで葉ノードの使用を削除でき，構築処理やバッファ管理を減らすことができる．

表 1 で用いた例を使い構築方法の説明を行う．**図 3** (a) は，幾何学的接尾辞木構築の初期段階であり，この $node_1$ と H_1 の組合せを初期タスクと定義する．$node_1$ の $S_{(1,1)}$ の長さ 5 は $length(node_1) = 1$ よりも長いので $node_2$ が作成される．$node_1$ の隠れ配列 $(1,3)$ は $\mathrm{MSSD}(node_2[1..3], H_{1,(1,3)}[1..3]) > 7$ となるので $node_3$ が作成される．同様に残りの隠れ配列も処理して $node_2$ と $node_3$ の隠れ配列に含まれる．その結果が図 3 (b) となる．$(1,4)$ と $(2,4)$ は長さが 1 で $length(node_1)$ と同じになり，葉ノードの作成を行わないため，$node_1$ より深い位置には存在しない．

トップダウン構築法で幾何学的接尾辞木を構築することで，構築処理と類似構造検索処理のそれぞれで高速化が可能となる．

以上の改良方法による幾何学的接尾辞木のトップダウン構築の処理手順は，以下のとおりである．

(1) タスクプールからタスクを受け取る．タスクプールが空の場合，処理を終了する．
(2) 親ノード $node_k$ の接尾辞の長さが $m = length(node_k)$ より長い場合，$node_k$ の深さを 1 増やした子ノードを作成．
(3) $node_k$ の隠れ配列の集合 H_k のすべてに対して (4) を行う．
(4) すべての子ノード $node_l$ に対して $\mathrm{MSSD}(node_l[1..m+1], H_{k,(i,j)}[1..m+1])$ が最小となる $node_l$ をみつける．$\mathrm{MSSD}(node_l[1..m+1], H_{k,(i,j)}[1..m+1]) < b$ である場合 H_l に追加，そうでなければ新しい子ノードを作成する．
(5) 子ノードをすべてタスク化して，タスクプールに格納し (1) に戻る．

以下では，トップダウン構築法の構築処理と検索処理の時間計算量について説明する．長さ 2 で幾何学的接尾辞木を構築すると，$n \times (m-1)$ 回の比較が行われる．これを長さ m まで繰り返すと，$n \times (m-1) \times (m-1)$ 回の比較が行われるので時間計算量は $O(nm^2)$ となる．したがって，逐次構築法と同じになる．また，逐次構築法とトップダウン構築法では，できあがる木が同じになるためデータ量や検索時間についても同じになる．しかし，トップダウン構築法では構築時のノードの作成順によりディスクとのデータの読み書きにかかる回数が少なくなるので，逐次構築法より高速に処理ができる．

4.1.2　バッファ管理法の変更

構築方法を逐次構築法からトップダウン構築法に変更することで，中間ノードページでは，中間ノードの作成は木の深さが浅いものから行われ，兄弟ノードはページ上に固まって配置される．よって，検索処理で深い中間ノードを含むページが読み込まれず，ディスクからの読み込み時間の短縮が可能となる．また，隠れ配列ページでは，中間ノードページと同様に，木の深さが浅いものから作成される．よって，構築処理では，作成された隠れ配列ページに書き込み途中となるページが存在しないため，すぐにディスクに書き込みを行う．そのページを再度読み込むことがないので，ディスクからの読み込み時間の短縮が可能となる．

構築方法の改良により，バッファ管理方法も**表 3** のように変更する．その結果，データページは中間ノードページだけを扱う．

4.2　幾何学的接尾辞木の並列化

幾何学的接尾辞木の構築処理と類似構造検索処理のそれぞれについて，マスタワーカ法と分散型ワーカ法を適用した並列化方法を提案する．並列化方法には，座標配列のデータセットに対してデータ分割を行い複数の木を扱う方

表 3　トップダウン構築法のバッファ管理方式
Table 3　Top-down construction method of buffer management scheme.

ページ名		構築処理	検索処理
データページ	中間ノードページ	TOP-Q	TOP-Q
	葉ノードページ	-	-
	隠れ配列ページ	-	-
カタログページ		LRU	LRU

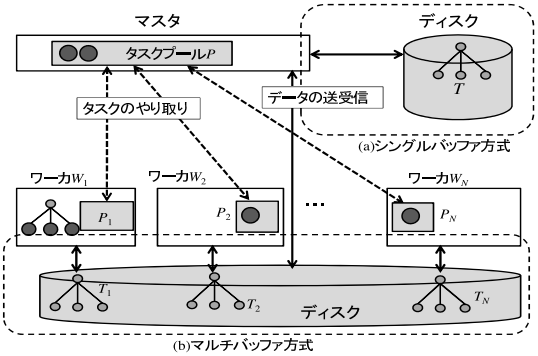

図 4　マスタワーカ法の構成
Fig. 4　Configuration of master worker method.

法と，データ分割を行わずに 1 つの木を扱う方法がある．ここでは，データ分割を行う方法を説明する．

4.2.1　マスタワーカ法の適用

マスタワーカ法は，データを処理するワーカとデータやタスクの管理を行うマスタで構成される．バッファ管理は，マスタで行うシングルバッファ方式と，各ワーカが行うマルチバッファ方式を用いる．この構成を図 4 に示す．

並列構築では，入力は蛋白質立体構造データベース DB とデータベースの分割数 N，出力は N 個のディスクベースの幾何学的接尾辞木の集合 $\{T_1, T_2, \cdots, T_N\}$ とする．初期タスクの作成方法は，入力データベース DB を N 個のデータセットに分割し，各データセットから初期タスクを作成する．

並列検索では，入力は蛋白質立体構造データベース DB，データベースの分割数 N，N 個のディスクベースの幾何学的接尾辞木の集合 $\{T_1, T_2, \cdots, T_N\}$ および問合せ Q，出力は問合せ Q に対する類似部分構造検索結果とする．T_i の初期タスクを作成する．

以下のように並列構築/検索処理を行う．

(1) マスタ側で，初期タスクを作成し，作成されたすべての初期タスク J_i をマスタ側のタスクプール $P = \{J_1, J_2, \cdots, J_N\}$ に保存する（$1 \leq i \leq N$）；

(2) マスタ側が空いているワーカ W_i のタスクプール P_i にタスク J を割り付ける；タスクプールを $P_i = P_i - \{J\}$ とする；

(3) ワーカ W_i は，割り付けられたタスク J で構築/検索処理を行う．if $P_i \neq \phi$ then P_i からタスク J を取り出し，$P_i = P_i - \{J\}$ として (3) の先頭にもどる；else マスタに通知する；

(4) マスタ側はワーカ W_i からの通知を受け取る；　if $P_j \neq \phi$ then 上記 (2) にもどる；else $P_j \neq \phi$ となっている他ワーカ W_j にタスクを要求する；

　　if 他ワーカ W_j からタスクを取得 then そのタスクを通知元のワーカ W_i に割り付け，(3) の先頭にもどる；

　　else 通知元のワーカ W_i の処理を終了する；

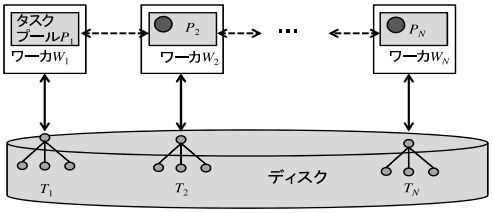

図 5　分散型ワーカ法の構成
Fig. 5　Configuration of distributed worker method.

4.2.2　分散型ワーカ法の適用

分散型ワーカ法は，データの処理とデータやタスクの管理を行うワーカのみで構成される．バッファ管理は，各ワーカが行うマルチバッファ方式を用いる．この構成を図 5 に示す．

並列構築では，入力は蛋白質立体構造データベース DB とデータベースの分割数 N，出力は N 個のディスクベースの幾何学的接尾辞木の集合 $\{T_1, T_2, \cdots, T_N\}$ とする．初期タスクの作成方法は，入力データベース DB を N 個のデータセットに分割し，各データセットから初期タスクを作成する．

並列検索では，入力は蛋白質立体構造データベース DB，データベースの分割数 N，N 個のディスクベースの幾何学的接尾辞木の集合 $\{T_1, T_2, \cdots, T_N\}$ および問合せ Q，出力は問合せ Q に対する類似部分構造検索結果とする．T_i の初期タスクを作成する．

以下のように並列構築/検索処理を行う．

(1) あるワーカ W_i で，入力データベース DB を N 個のデータセットに分割する；各データセットから初期タスクを作成し，作成されたタスクの集合 $\{J_1, J_2, \cdots, J_N\}$ を各ワーカ W_i のタスクプール P_i に保存する（$1 \leq i \leq N$）；

(2) ワーカ W_i で，あるタスクの処理が終了；if $P_i \neq \phi$ then P_i からタスク J を取り出し，$P_i = P_i - \{J\}$ とする．タスク J で幾何学的接尾辞木の構築/検索処理を行い，(2) の先頭にもどる；else $P_j \neq \phi$ となっている他ワーカにタスクを要求する；

　　if タスクを取得 then そのタスクをもとに，ディスクベースの幾何学的接尾辞木の構処理を行い，(2) の先頭にもどる；

　　else ワーカ W_i の処理を終了する；

5. 評価実験

本章では新しく前章で提案した幾何学的接尾辞木の並列化について評価を行う．

5.1 実験方法

提案手法の評価を行うために，wwPDB に 2012 年 1 月 11 日時点で登録されていた PDBML 形式の座標配列データファイルを，PDB の ID に含まれる鎖ごとにアミノ酸配列の C_α 原子の座標データを取り出し，評価実験データに使用した．PDB から取得した実験データは 366,146 件，データサイズは 1,400 MB である．検索キーに使用したタンパク質立体構造データは，1A0H，1X2N，1LCP を用いた．これらの検索キーには，それぞれ，Kringle，Homeobox，Leucine の構造モチーフを含むので，以下では，これらの検索キーを Kringle，Homeobox，Leucine と呼ぶ．分岐パラメータは $b = 400$，検索パラメータは $\epsilon = 2.5$，ページサイズは 8 KB，データページのバッファサイズは 70 GB，カタログページのバッファサイズは 70 MB とする．実験環境は CPU：Xeon CPU　E5-2670 2.60 GHz × 2（8 × 2 コア），メモリ：192 GB，ディスク容量：12 TB である．

5.2 評価

ここでは，まず，従来手法と提案手法を比較し，提案手法の有効性を確かめる．さらに，提案手法の 3 つの並列化手法を比較し，マルチバッファ・マスタワーカ法の有効性を確かめる．その方法として，構築と検索にかかる時間を測定する．測定項目は，(1) 構築/検索処理と (2) バッファ管理処理，(3) その他の処理の 3 つである．(1) の構築/検索処理と (2) のバッファ管理処理については実行時間としてともに CPU 時間を測定し，(3) のその他の処理については実時間からほかの 2 項目の CPU 時間を引いたものとする．さらに検索処理では，検索精度の確認のために，問合せ Q をもとに蛋白質立体構造データベース DB の先頭の配列データから最後の配列データまで，RMSD に基づくデータベーススキャンの実施結果を正解のデータとした．

5.2.1 従来手法との比較

ここでは幾何学的接尾辞木の構築・検索の並列化による高速化の効果を従来手法と提案手法それぞれで調べる．従来手法と提案手法の条件を合わせるために，使用するワーカ数にデータを均等に分割し，1 つの分割データを 1 つのワーカが処理する．このとき，各ワーカ間でタスク処理の共有は行わない．使用するワーカ数は 1 から 16 でデータサイズは 88 MB から 1,400 MB である．その結果が**図 6** と**図 7** となり，構築処理と検索処理の両方で処理時間がデータサイズに比例していることが確認できた．また，提案手法が従来手法に比べ，高速に処理を行っている．このことは，検索キー Homeobox と Leucine についても確認できた．よって，逐次構築法からトップダウン構築法に変更することで，構築・検索時間の高速化ができることを確認できた．データサイズに比例していることが確認できた．さらに，検索では，従来手法と提案手法の両方で素朴な方法と同じ検索結果が得られた．

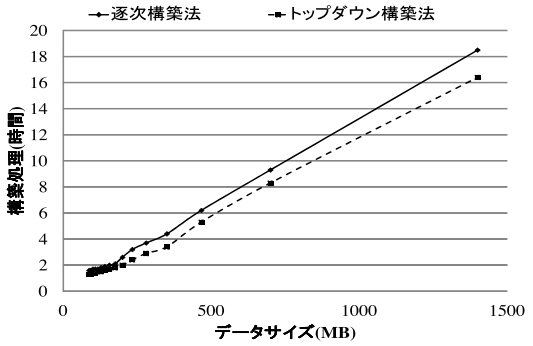

図 6　データサイズごとの構築処理にかかる時間
Fig. 6　Time of construct process for each data size.

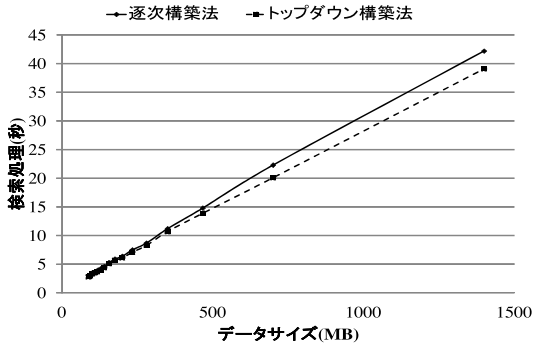

図 7　データサイズによる検索処理にかかる時間（Kringle）
Fig. 7　Time of search process for each data size (Kringle).

5.2.2 幾何学的接尾辞木の並列化

ここでは幾何学的接尾辞木の構築・検索の並列化による高速化の効果を提案手法の 3 種類の方法それぞれで調べる．

データ分割の方法として，構築する木の数や分割データの統合方法を変えて**表 4** のように行う．評価実験では使用するワーカ数を M，データ分割数を L とし，$M \leq L$ とする．本実験では，使用ワーカ数は 16，データ分割数は 1000 とする．構築する木の数を複数にする場合には，方法 3 の構築する木の数と分割データ数を合わせる場合が並列化に適していると考えられる．使用するワーカ数：16 データ分割：方法 3 で評価実験を行った結果が**表 5** と**表 6** となり，マルチバッファ・マスタワーカ法が最も効果が良いことが分かった．このことは，構築と検索それぞれの並列化手法と表 4 の実験方法の組合せ，および検索キー Kringle 以外についても確認できた．また，検索精度では，すべての実験で素朴な方法と同じ類似部分構造検索の結果が得られた．

表 4　データ分割を用いた評価実験手法
Table 4　Evaluation experiment method using the data partitioning.

ページ名	方法 1	方法 2	方法 3
1 ワーカが構築する木の数	L/M	1	1
構築処理でのノードデータの修正	なし	あり	なし
分割データの統合	-	構築処理時	構築処理前
検索における問題点	・複数の木の探索が必要 ・隠れ配列による木の探索ノード数減少の利点が少ない	・データの配置がばらばらになるのでページの読み込みが多くなる ・トップダウンの利点がなくなる	なし（方法 1 と方法 2 の問題点の解決）

表 5　各並列化手法の構築性能の比較（時間）
Table 5　Comparison of construction performance of each parallelization method (hour).

並列化法	合計	構築処理	バッファ管理	その他
シングルバッファ・マスタワーカ法	2.6	1.3	0.8	0.5
マルチバッファ・マスタワーカ法	2.0	1.3	0.5	0.2
マルチバッファ・分散型ワーカ法	2.2	1.4	0.5	0.3

表 6　各並列化手法の検索性能の比較（秒）
Table 6　Comparison of retrieval performance of each parallelization method (second).

並列化法	合計	検索処理	バッファ管理	その他
シングルバッファ・マスタワーカ法	2.6	1.3	0.8	0.5
マルチバッファ・マスタワーカ法	2.0	1.3	0.5	0.2
マルチバッファ・分散型ワーカ法	2.2	1.4	0.5	0.3

5.3　考察

従来手法との比較では，図 6，図 7 で構築および検索時間の高速化が確認できる．この高速化が可能となった理由を確認するために，**表 7** でノードの作成回数と修正回数を，**表 8** で構築時のページ I/O の回数を，**表 9** で検索時のノード参照数とページ読み込み回数を測定した．構築では，表 7 の測定結果から構築時のノード修正回数，表 8 の測定結果からページ I/O のそれぞれの回数の減少から，構築処理時間とバッファ管理時間ともに時間の短縮の効果がみられる．検索では，表 9 の測定結果から読み込むノードページの減少からバッファ管理時間の高速化が確認できる．

提案手法である並列化手法の比較では，表 5，表 6 の測定結果から構築および検索時間ともにマルチバッファ方式を用いたマスタワーカ法が最も高速に処理することができていることが確認できる．シングルバッファ方式では，バッファ管理とその他の時間がマルチバッファ方式に比べ多くの時間がかかっている．このことは，バッファ管理で複数のワーカからのディスクとのデータの読み書きの要求がマスタに集中しているので，複数のワーカで待ち状態になるからである．また，分散型ワーカ法では，隣り合うワーカからしかタスクを受け取ることができないので，多くのワーカがタスクを要求しているときには，すべてのタ

表 7　構築処理におけるノード作成数
Table 7　Node creation number in the construction process.

ワーカ数	従来手法		提案手法	
	ノード作成回数	ノード修正回数	ノード作成回数	ノード修正回数
1	3,674,681	4,375,145	3,674,681	0
4	3,942,762	4,864,325	3,942,762	0
8	4,235,548	5,312,146	4,235,548	0
12	4,687,435	5,834,532	4,687,435	0
16	4,931,352	6,275,438	4,931,352	0

表 8　構築処理におけるページ I/O
Table 8　The page I/O in the construction process.

ワーカ数	従来手法		提案手法	
	ページ読み込み	ページ書き込み	ページ読み込み	ページ書き込み
1	0	0	0	0
4	453	1,342	0	781
8	4,857	9,312	0	4,245
12	36,455	51,357	0	13,457
16	61,432	82,145	0	19,723

表 9　検索処理におけるデータ参照数
Table 9　Data reference number in the search process.

ワーカ数	従来手法		提案手法	
	ノード参照数	ページ読み込み	ノード参照数	ページ読み込み
1	104	3	104	1
4	406	10	406	4
8	821	18	821	8
12	1,231	26	1,231	12
16	1,614	33	1,614	16

スクの処理が終わるか他のワーカからタスクが回ってくることを待つ必要がある．

データ分割の方法では，方法 3 が構築および検索時間ともに最も高速に処理することができていることが確認できる．このことは，方法 1 では，1 つの木の処理時間は少ないが，データ数が多いので処理時間が大きくなる．また，方法 2 では，構築した木に何度も追加することで，同じデータに対して複数の参照が起こる．さらに，書き込み済みのデータの後ろにさらにデータが書き込まれることで．トップダウン構築の利点であるデータの配置が悪くなるからである．

6. まとめ

本研究では，幾何学的接尾辞木の高速処理を行うために 2 つの問題点を解決する方法を提案した．提案手法は，(1) トップダウン構築法とバッファ管理の変更，(2) 幾何学的接尾辞木の構築と検索の高速化のための問題分割法と負荷分散法を組み合わせた並列化の方法より構成されている．提案手法の評価を行うために，wwPDB からアミノ酸で構成された蛋白質の全座標配列を取得し，幾何学的接尾辞木の構築および検索実験を行った．評価実験の結果，高精度性を維持しつつ，従来手法より高速な幾何学的接尾辞木の構築と類似部分構造検索ができることを確認した．さらに，幾何学的接尾辞木の並列化ではマルチバッファ方式を用いたマスタワーカ法がシングルバッファ方式を用いたマスタワーカ法と分散型ワーカ法より高速に処理することができていること，データ分割数の方法で方法 3 が方法 1，2 より高速に処理できていることが確認できた．

今後の課題として，幾何学的接尾辞木の並列化のために，さらに適した方法の検討などがあげられる．

謝辞　本研究の一部は日本学術振興会・科学研究費補助金（基盤研究（C），課題番号：20500137 および 26330139）の助成を受けたものです．

参考文献

[1] Shibuya, T.: Geometric Suffix Tree: A New Index Structure for Protein 3-D Structures, *Combinatorial Pattern Matching 2006*, LNCS 4009, pp.84–93 (2006).
[2] Shibuya, T.: Geometric suffix tree: Indexing protein 3-D structures, *J. ACM*, Vol.57, No.3, pp.15:1–15:17 (2010).
[3] 高橋誉文，黒木　進，田村慶一，北上　始：複数の蛋白質立体構造データに対するディスク版索引構造の構築方式，データ工学と情報マネジメントに関するフォーラム DEIM2009，電子情報通信学会データ工学研究専門委員会，p.8 (2009).
[4] Takahashi, Y., Kuroki, S. and Kitakami, H.: Efficient Query Processing in Protein Structure Databases, *Proc. 2nd International Workshop with Mentors on Databases, Web and Information Management for Young Researchers* (*iDB Workshop 2010*), pp.47–55 (2010).
[5] 高橋誉文，田村慶一，黒木　進，北上　始：幾何学的なサフィックス木による高速類似構造検索手法，情報処理学会論文誌データベース，Vol.6, No.5, pp.62–70 (2013).
[6] 高橋誉文，田村慶一，黒木　進，北上　始：幾何学的サフィックス木に対する並列処理性能の評価，Vol.2014-MPS-98, No.21, pp.1–8 (2014).
[7] Weiner, P.: Linear pattern matching algorithms, *Proc. 14th Annual Symposium on Switching and Automata Theory* (*swat 1973*), *SWAT '73*, pp.1–11 (1973).
[8] McCreight and Edward, M.: A Space-Economical Suffix Tree Construction Algorithm, *J. ACM*, Vol.23, No.2, pp.262–272 (1976).
[9] Esko, U.: On-Line Construction of Suffix Trees, *Algorithmica*, pp.249–260 (1995).
[10] Ramesh, H.: Optimal parallel suffix tree construction, *Proc. 26th annual ACM*, pp.290–299 (1994).
[11] Bieganski, P., Riedl, J., Cartis, J.V. and Retzel, E.F.: Generalized suffix trees for biological sequence data, *Applications and implementation*, pp.35–44 (1994).
[12] Bedathur, S.J. and Haritsa, J.R.: Engineering a fast online persistent suffix tree construction, *20th Int'l Conference on Data Engineering*, pp.720–731 (2004).
[13] Ching-Fung, C., Jeffrey, X.Y. and Hongjun L.: Constructing Suffix Tree for Gigabyte Sequences with Megabyte Memory, *IEEE Trans. Knowledge and Data Engineering*, Vol.17, No.1, pp.90–105 (2005).
[14] Tian, Y., Tata, S., Hankins, R.A. and Patel, J.M.: Practical methods for constructing suffix trees, *VLDB Journal*, Vol.14, No.3, pp.281–299 (2005).
[15] Benjarath, P. and Mohammed J.Z.: Genome-scale Disk-based Suffix Tree Indexing, *SIGMOD'07*, pp.833–844 (2007).
[16] Amol, G. and Konstantin M.: Serial and Parallel Methods for I/O Efficient Suffix Tree Construction, *SIGMOD'09*, pp.827–840 (2009).
[17] Watanuki, Y., Tamura, K., Kitakami, H. and Takahashi, Y.: Parallel Processing of Approximate Sequence Matching using Disk-based Suffix Tree on Multi-core CPU, *IWCIA2013, IEEE SMC Hiroshima Chapter*, p.6 (2013).
[18] Watanuki, Y., Tamura, K., Kitakami, H. and Takahashi, Y.: Multiple Buffering for Parallel Approximate Sequence Matching using Disk-based Suffix Tree on Multi-core CPU, *GSTF Journal on Computing*, *JoC*, Vol.3, No.3, pp.51–57 (2014).
[19] 澤田祐介, 田村慶一, 荒木康太郎, 高木　允, 北上　始：PC クラスタ上でのディスクベースサフィックス木の並列構築方式，DEWS2008, Online proceedings, D2-3 (2008).
[20] Sawada, Y., Tamura, K., Araki, K., Takaki, M. and Kitakami, H.: Parallel Construction Method of a Disk-Based Suffix Tree on a PC Cluster, *PDPTA'08*, pp.797–803 (2008).
[21] 澤田祐介，田村慶一，荒木康太郎，高木　允，北上　始：分散並列環境におけるディスクベースサフィックス木の構築と検索，MPS70, pp.39–42 (2008).
[22] Benjarath, P. and Mohammed, J.Z.: TRELLIS+: An Effective Approach for Indexing Genome-Scale Sequences Using Suffix Trees, *Pacific Symposium on Biocomputing*, pp.90–101 (2008).
[23] Arun, K.S., Huang, T.S. and Blostein, S.D.: Least-Squares Fitting of Two 3-D Point Sets, *IEEE Trans. Pattern Anal. Mach. Intell.*, Vol.9, No.5, pp.698–700 (1987).
[24] Eggert, D.W., Lorusso, A. and Fisher, R.B.: Estimating 3-D rigid body transformations: a comparison of four major algorithms, *Mach. Vision Appl.*, Vol.9, No.5-6, pp.272–290 (1997).

[25] Kabsch, W.: A solution for the best rotation to relate two sets of vectors, *Acta Crystallographica Section A*, Vol.32, No.5, pp.922–923 (1976).
[26] Kabsch, W.: A discussion of the solution for the best rotation to relate two sets of vectors, *Acta Crystallographica Section A*, Vol.34, No.5, pp.827–828 (1987).
[27] Schwartz, J.T. and Sharir, M.: Identification of partially obscured objects in two and three dimensions by matching noisy characteristic, *Int. J. Rob. Res*, Vol.6, No.2, pp.29–44 (1987).
[28] Umeyama, S.: Least-Squares Estimation of Transformation Parameters Between Two Point Patterns, *IEEE Trans. Pattern Anal. Mach. Intell.*, Vol.13, No.4, pp.376–380 (1991).
[29] Srikanta, J. and B., Jayant, R.H.: Engineering a Fast Online Persistent Suffix Tree Construction, *ICDE '04 Proc. 20th International Conference on Data Engineering*, Vol.29, pp.720–731 (2004).

高橋 誉文　（学生会員）

2008 年広島市立大学情報科学部知能情報システム工学科卒業．2010 年同大学大学院情報科学研究科博士前期課程修了．同年同博士後期課程進学．日本データベース学会会員．

田村 慶一　（正会員）

広島市立大学情報科学研究科准教授．博士（情報科学）．1998 年九州大学工学部情報工学科卒業．2000 年同大学大学院システム情報科学研究科知能システム学専攻修士課程修了．2003 年同大学院システム情報化学府知能システム学専攻博士後期課程単位取得のうえ満期退学．広島市立大学情報科学部助手，広島市立大学大学院情報科学研究科助教，同講師を経て，2011 年より現職．データマイニングとその並列処理に関する研究に従事．IEEE，日本データベース学会，人工知能学会，日本知能情報ファジイ学会各会員．

黒木 進　（正会員）

広島市立大学大学院情報科学研究科准教授．博士（工学）．1990 年東京大学大学院工学系研究科博士前期課程修了．九州大学助手，広島市立大学助教授を経て，2007 年より現職．主に時空間データ検索の研究に従事．ACM，IEEE，電子情報通信学会，日本データベース学会，人工知能学会等各会員．

北上 始　（正会員）

広島市立大学情報科学研究科教授．博士（工学）．1976 年東北大学大学院工学研究科博士前期課程修了．同年富士通株式会社，以後，富士通研究所，新生代コンピュータ技術開発機構（ICOT）主任研究員，国立遺伝学研究所客員助教授を経て，1994 年広島市立大学情報科学部教授，2007 年より現職．筆頭著書『データベースと知識発見』，『ビッグデータ時代のゲノミクス情報処理』．1985 年情報処理学会 25 周年記念論文賞，2003 年日本工学教育協会論文・論説賞ほか．情報処理学会シニア会員．日本データベース学会（論文誌編集委員），電子情報通信学会，人工知能学会（評議員），日本バイオインフォマティクス学会，IEEE，ACM 各会員．

（担当編集委員　坂本 比呂志）