段階的一般化法の高速化に関する考察
Comparison Study of Speedup for Step-Wise Generalization Method

八木　真平
Shinpei Yagi
広島市立大学情報科学研究科
Email:mu67032@edu.ipc.hiroshima-cu.ac.jp

田村　慶一
Keiichi Tamura
広島市立大学情報科学研究科
Email: ktamura@hiroshima-cu.ac.jp

北上 始
Hajime Kitakami
広島市立大学情報科学研究科
Email: kitakami@hiroshima-cu.ac.jp

Abstract—**An ambiguous query processing returns a large number of similar subsequences, called a mismatch cluster, to the user. It is difficult for the user to discover the characteristics from a mismatch cluster. The step-wise generalization method has proposed for extracting the minimum generalization set that expresses the mismatch cluster. In this paper, we discuss speedup for step-wise generalization method by using cache-conscious data structures and reducing the amount of data transfers between main memory and caches.**

I. はじめに

曖昧な問合せ処理では，非常に多くの類似部分文字列（ミスマッチクラスタ）が検索結果として得られる．ミスマッチクラスタをユーザが直接閲覧して規則性を把握することは困難である．そこで，ミスマッチクラスタを表現する最小汎化集合を抽出する手法として段階的一般化法が提案されている[1]．段階的一般化法は最小汎化集合を効率的に抽出することができるが，ミスマッチクラスタを構成する部分文字列が増えると計算時間が膨大となる．

そこで我々は，段階的一般化法を高速化するためにマルチコア CPU 上での並列化手法を提案している[2]．近年，CPU のマルチコア化が進んでおり，マルチコア CPU の計算機資源を最大限に活かすには，複数のコアを使って効率的に並列化を行うことと，キャッシュミスを軽減することやメインメモリとキャッシュ間のデータ転送量を最小限に抑える必要がある．そこで，マルチコア CPU の構造を意識したデータ構造やアルゴリズムを開発する研究が盛んに行われている．

我々は，これまでに，マルチコア CPU 上の複数コアを有効的に使用するための並列化モデルを提案しているが，キャッシュミスの軽減，メインメモリとキャッシュ間のデータ転送量を最小限に抑える方式はまだ未検討であった．そこで，本論文では，段階的一般化方においてキャッシュコンシャスなデータ構造を用いることと索引構造を利用することの効果について考察する．

具体的な内容は次の通りである．

- 段階的一般化法の主な処理は列挙木の生成であり，列挙木の主なデータ構造は木のノードとなる．文献[3]，[4]，[5]ではデータ構造をポインタベースで作成するよりも配列，つまり，キャッシュコンシャスなデータ構造を利用することで処理の高速化を行うことができることが示されている．そこで，列挙木のノードのデータ構造をポインタベースから配列に変更を行う．
- 段階的一般化法では列挙木の枝刈りを行うことで効率的に列挙木の生成を行なっているが，枝刈りの判定毎にミスマッチクラスタを毎回，メインメモリから CPU，つまりキャッシュに転送する必要がある．そこで，すべての ミスマッチクラスタを転送することを避けるためにハッシュテーブル，パトリシア木を用いた枝刈りの判定を行う．

提案する高速化手法を用いて，段階的一般化法を実際に実装し，文献[1]で用いられているデータセットに対して評価実験を行った．評価実験の結果，処理の高速化を確認することができ，キャッシュコンシャスなデータ構造を用いることと索引構造を利用することの効果を確認することができた．

本論文の構成は以下の通りである．第 2 章では段階的一般化法の説明や用語，記号の定義について説明する．第 3 章では提案手法，第 4 章では実験とその結果に対する評価を示す．第 5 章でまとめ，今後の課題について述べる．

II. 段階的一般化法

本章では，用語の定義，最小汎化集合の定義と段階的一般化法の処理手順を示す．

A. 用語の定義

1) 曖昧な問い合わせとミスマッチクラスタ

部分文字列Kと許容誤差r ($\geqq 0$)が，問合せQとして与えられ，ハミング距離d (K, K') $\leq r$を満たす部分文字列K'が配列データベースDBからすべて選択されるとき，検索結果として得られる長さkの部分文字列*<inst>*（以下，インスタンスと呼ぶ）の集合をミスマッチクラスタ*MIS*と呼ぶ．また，ミスマッチクラスタ*MIS*に含まれるイン

スタンスを正のインスタンス，含まれないインスタンスを負のインスタンスと呼ぶ．

2) 曖昧性を表現する汎化配列パターン

アルファベットΣの部分集合をΣ_iとするとき，*k*個のΣ_iを並べたパターンを*k* - 汎化配列パターンと呼び，$<pat^k>=<\Sigma_1\Sigma_2 \ldots \Sigma_{k-1}\Sigma_k>$と表記する．ただし，$\Sigma_i$は括弧[ ]の中に$\Sigma_i$の全要素を列挙した形で表記をする．例えば，2 - 汎化配列パターン<[AB][CD]>では，$\Sigma_1=\{A,B\}$，$\Sigma_2=\{C,D\}$となる．

3) インスタンスを導出する関数

k - 汎化配列パターンから*k* - インスタンスのすべて，すなわち長さ*k*の部分文字列の集合を導出する関数をEVAL($<pat^k>$)と表記する．例えば，*EVAL* (<[AB][CD]>) = {<AC>,<AD>,<BC>,<BD>}である．2つの*k*-汎化配列パターン$<pat_1^k>$と$<pat_2^k>$について，$EVAL(<pat_1^k>) \supseteq EVAL(<pat_2^k>)$が成立するとき，$<pat_2^k>$は$<pat_1^k>$に冗長であるという．

4) 最汎パターン

また，ある*k* - インスタンスの集合をI^kとする．ここで，$1 \leqq j \leqq k$に対して，$\Sigma_j = \{inst[j] \mid inst \in I^k\}$であるとする．ただし，*inst*[*j*]は，*inst*の先頭から*j*番目の文字を意味する．このとき，$<\Sigma_1\Sigma_2 \ldots \Sigma_{k-1}\Sigma_k>$を最汎パターンと呼び，$MGP(I^k)$と表記する．例えば，インスタンスの集合{<ABF>,<AEF>,<DBF>}の最汎パターン*MGP*は<[AD][BE]F>となる．

B. 最小汎化集合の定義

ある集合$MGS = \{G_1, G_2, \ldots, G_m\}$が，*k* -汎化配列パターン$<pat^k>$および*k* -インスタンス$<inst^k>$から構成されているとする($1 \leqq m \leqq |MIS|$)．ただし，EVAL($<pat^k>$)$\subseteq MIS$ かつ$<inst^k> \in MIS$を満たすものとする．この集合 *MGS* が次の性質を満たすとき，*MGS* を*MIS* に対する最小汎化集合と呼ぶ．

(1) *EVAL*(*MGS*)＝*MIS* が成立する．

(2) *MGS* の任意の2要素G_i, G_jに対して，G_iとG_jの間には冗長な関係が存在しない($1 \leqq i \leqq j \leqq m$)．

(3) *MGS* に含まれるどの要素G_iも極大である($1 \leqq i \leqq m$)．すなわち，さらに汎化すると*MIS*に存在しないインスタンスを含んでしまうことになる．

(4) 上記の(1) ～ (3)を満たす任意の *MGS'*に対して，$|MGS'| \leqq |MGS|$が成立する．

C. 処理手順

図 1 に段階的一般化法の処理手順を示す．以下の手順(1)と(2)は，列挙木を深さ優先に生成するための変数 *Stack* と候補解を格納するための変数 *Candidate* について，初期化を行う処理である．手順(2)(a)は，親ノードから子ノードを列挙する処理である．手順(3)(b)①は列挙木ノードの最汎パターンを計算する処理であり，手順(3)(b)②は列挙木ノードの枝刈り処理である．手順(3)(c)は候補解集合の構成要素を収集する処理であり，手順(4)は候補集合から冗長な葉ノードを除去するための処理である．

手順(3)(b)②は列挙木ノードの枝刈り処理では，汎化配列パターンから導出されるすべてのインスタンスに負のインスタンスが存在するかどうかを判定している．つまり，負のインスタンスを含む場合，そのノードは負のパターンとなり，候補解とはなないため，枝刈りすることができる．文献[1]では，*EVAL*($<pat>$)から導出されるすべてのインスタンスではなく，親パターンとの差分から導出されるインスタンスのみ対象とすることで，この判定を高速化に行なっている．

(1) *Stack* : = { {1}, {2},・・・,{n}}; ただし，{ *i* }∈*Stack* の *i* はミスマッチクラスタ *MIS* に存在する要素の識別番号とし，*Stack* の要素は辞書式順に取り出せるように管理する．
(2) *Candidate* := { }; ただし，*Candidate* を候補解を格納する領域とする．
(3) ***while*** (*Stack* ≠ϕ)
 for each *S*∈ *Stack*
 (a) *Next* := *S* を親ノードとして列挙された子ノード *C* の集合;
 (b) ***for each*** *C*∈ *Next*
 ① $<pat>$:= *MGP*(*C* に対する *SubMIS*);
 ② ***if*** *EVAL*($<pat>$)－*MIS* ≠ ϕ ***then*** *C* を捨てる;
 else *Stack* : = *Stack* ∪ {*C*};
 (c) ***if*** *S* が葉ノードの条件を満たす ***then***
 Candidate : = *Candidate* ∪ *MGP*(*S* に対する *SubMIS*);
(4) *Candidate* から冗長な要素を除去;

図 1: 段階的一般化法

III. 提案手法

本章では，提案する段階的一般化法の高速化手法について説明する．

A. ポインタを用いないデータ構造への改良

段階的一般化法では，列挙木ノードのデータを格納するためにポインタ構造を使用している．具体的には第 2 章で示したアルゴリズムの子ノード集合をポインタ構造で格納している．ポインタ構造を使ったデータ構造はデータが不連続な領域に格納されるためキャッシュミスを増加させてしまう．

そこで，あるノードの子ノード集合を格納するための連続した配列領域を確保し，ノードを配列に格納する．子ノード集合を読み出す場合，

ポインタ構造を用いるよりも配列を用いることで，キャッシュミスの発生を防ぐことができる．

B. データ転送量削減のための枝刈り判定

マルチコア CPU 上において効率的な並列化を目指す場合，キャッシュ，バスやディスク I/O など共有資源の競合を防ぐ必要がる．特に，メインメモリとキャッシュ間のデータ転送量を減らすことは，バスやキャッシュの競合を防ぐことになり，性能向上が期待できる．

段階的一般化法では，枝刈り判定時に負のインスタンスと呼ばれるミスマッチクラスタに存在しないインスタンスを見つけるために，インスタンスがミスマッチクラスタに存在するかどうかをノード生成時に毎回チェックする必要がある．

この枝刈判定について，（1）ハッシュテーブルを用いる手法，（2）パトリシア木を用いる手法を提案する．

1) ハッシュテーブルによる枝刈判定

ミスマッチクラスタの各インスタンスについて，先頭文字をキーとしたハッシュテーブルを作成する．*EVAL*(<*pat* >)から導出されるインスタンスについて，インスタンスの先頭文字をキーとしてハッシュテーブルを探索し，インスタンスが負のインスタンスかどうか判定する．

2) パトリシア木による枝刈判定

ミスマッチクラスタの全インスタンスを文字列として，パトリシア木を作成する．文字列集合を格納するトライ木に基づいて作成された多分の木構造である．*EVAL*(<*pat* >)から導出されるインスタンスをキーとしてパトリシア木を探索することで，インスタンスが負のインスタンスかどうか判定する．

IV. 評価実験

評価実験では，*Kringle*，*Homeobox*，*PTS_EIIA*，のミスマッチクラスタのデータセット（文献[1]）を用い，提案手法を評価した．利用した計算機環境は，カスタムメイド PC（CPU：Phenom Ⅱ X6 1090T Six-Core/3.2G/L3：6MB，メモリ：4.0GB，OS：ubuntu11.04）である．通常の段階的一般化法を swg，データ構造として配列を用いた段階的一般化法を cswg，ハッシュテーブルを用いた手法を h-cswg，パトリシア木を用いた手法を radix-cswg と表記する．

A. 実験 1

swg，cswg，h-cswg，radix-cswg にて各データセットで 3 回ずつ実行することで処理時間の評価した．表 1〜表 3 に結果を示す．結果から処理時間は *Kringle* を除き， radix-cswg が最も処理時間が小さい．

B. 実験 2

swg，cswg，h-cswg，radix-cswg にてキャッシュミス回数を比較する．測定方法は（実験 1）と同じである．L1 キャッシュでのキャッシュミス，L2 キャッシュでのキャッシュミス，総サイクル数，待ちサイクル数を計測した．図 2，図 3 に各手法の *Kringle*，*Homeobox*，*PTS_EIIA* の L1 キャッシュミス回数と L2 キャッシュミス回数を示す．図 4，図 5 に各手法の *Kringle*，*Homeobox*，*PTS_EIIA* の総サイクル数と待ち率を示す．

C. 考察

まず，キャッシュコンシャスデータ構造を用いることで，キャッシュミスが大幅に削減することができている．このことが，性能向上に大きく貢献していることが分かる．次に，判定処理による効果についてはデータ転送量を直接計測することはできないが，L2 キャッシュミスが軽減していることからデータ転送量が軽減することができていることが確認できる．以上の結果から，キャッシュコンシャスなデータ構造はキャッシュの軽減，また，判定処理はデータ転送量の削減に貢献できていると考えられる．

表 1: *Kringle* の処理時間比較

手法	処理時間（秒）
swg	15.905
cswg	12.438
h-cswg	6.839
radix-cswg	11.025

表 2: *Homeobox* の処理時間比較

手法	処理時間（秒）
swg	74.262
cswg	44.85
h-cswg	38.33
radix-cswg	15.061

表 3: *PTS_EIIA* の処理時間比較

手法	処理時間（秒）
swg	171.247
cswg	126.869
h-cswg	71.742
radix-cswg	48.056

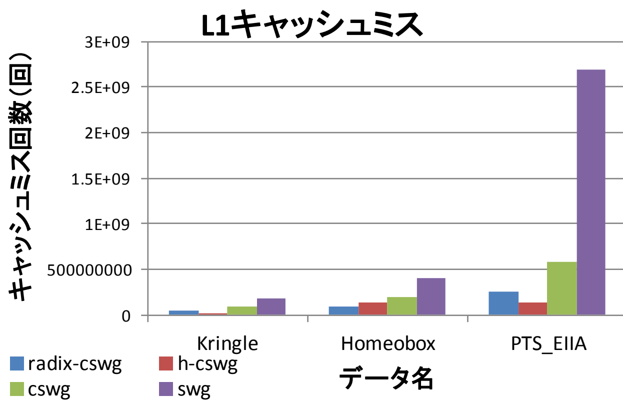

図 2: L1 キャッシュミス回数

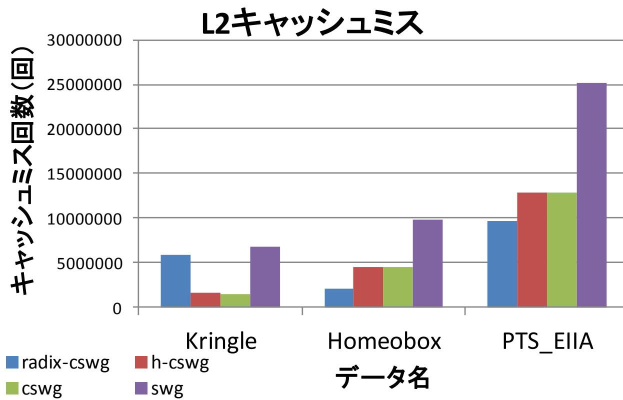

図 3: L2 キャッシュミス回数

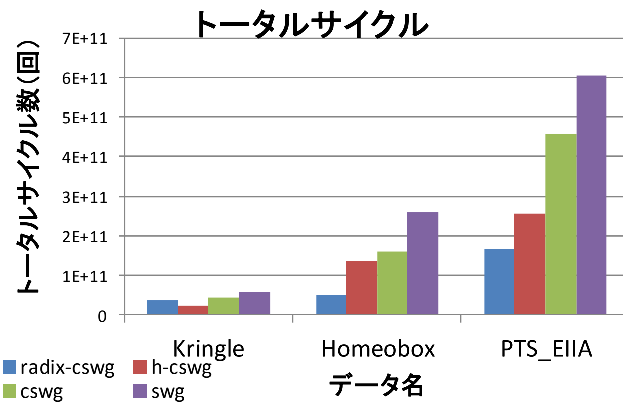

図 4:総サイクル数

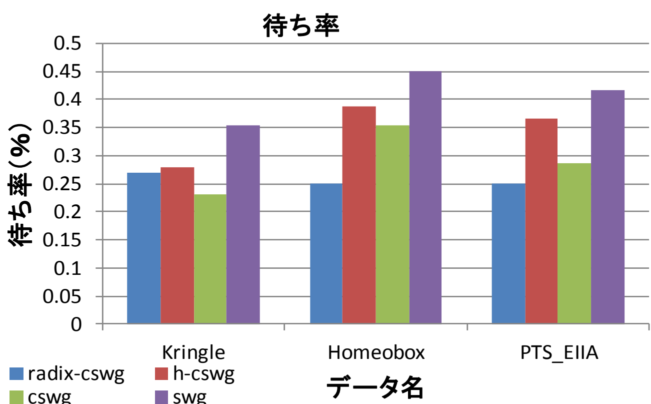

図 5:待ちサイクル数

V. おわりに

本論文では，段階的一般化法の高速化に関する考察を行った．評価実験より，キャッシュコンシャスなデータ構造を使用すること，データ転送量を削減する判定処理を組み込むことで段階的一般化法を高速化することが分かった．今後の課題として，今回の手法を，並列化モデルに組み込むことが挙げられる．

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)，文部科学省・科学研究費補助金（若手研究(B)，課題番号：23700124）の支援により行われた．

参考文献

[1] 田村慶一，木村浩明，荒木廉太郎，北上始：段階的一般化法によるミスマッチクラスタを表現する最小汎化集合の効率的抽出，電子情報通信学会論文誌 D, Vol. J93-D, No. 3, pp.189-202, 2010.

[2] 八木 真平, 田村 慶一, 北上 始, “マルチコアCPU 上での段階的一般化法の並列処理，” IEEE SMC Hiroshima Chapter 若手研究会, pp.75-78, 2011.

[3] Amol Ghoting, Gregory Buehrer, Srinivasan Parthasarathy, Daehyun Kim, Anthony Nguyen, Yen-Kuang Chen, and Pradeep Dubey, “Cache-conscious frequent pattern mining on a modern processor, ” In Proceedings of the 31st international conference on Very large data bases (VLDB '05), pp.577-588, 2005.

[4] Won-Sik Kim, Woong-Kee Loh, and Wook-Shin Han, “Performance analysis of the cache conscious-generalized search tree, ” In Proceedings of the 6th international conference on Computational Science - Volume Part III (ICCS'06), pp.648-655, 2006.

[5] Kamen Yotov, Tom Roeder, Keshav Pingali, John Gunnels, and Fred Gustavson, “An experimental comparison of cache-oblivious and cache-conscious programs, ” In Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures (SPAA '07), pp.93-104,2007.

問い合わせ先
〒731-3194
広島市安佐南区大塚東 3 丁目 4 番 1 号
広島市立大学　情報科学研究科　知能工学専攻
八木　真平