

飽和パターンマイニングの並列化のための一手法

長尾 雅弘 世木 博久

名古屋工業大学大学院情報工学専攻

1. はじめに

データベースから知識を発見する手法の一つであるパターンマイニングでは、膨大な数のパターンが発見され、それによる処理時間の増大や、ユーザにとって不必要なパターンの出現が起きる。飽和パターンは同じ意味を持つパターン集合に唯一存在する極大元となるパターンであり、そのみをマイニングすることで、発見するパターンの数を抑えることができる[1]。

パターンマイニングに関連した手法に、形式概念分析がある。これは分析対象とそれが持つ性質から概念単位を得る方法であり、この概念が飽和パターンとその出現集合に対応する。

また、パターンマイニングでは大規模なデータを処理対象とするため、処理時間の短縮が課題となる。Krajca ら[2]はマイニング処理を並列化することでこの課題に対応したが、分散のバランスを調節するためのパラメータが適切な値でなかった場合に処理時間があまり短縮されなくなる問題があった。そこで本研究では、パラメータを用いずに均等な負荷分散を行う並列化方法を提案し実装する。

2. 形式概念とその計算

形式概念分析では、対象集合 G とその属性集合 M からなる形式文脈を扱う。対象の部分集合 $X \subseteq G$ と属性の部分集合 $Y \subseteq M$ に対して X のすべての対象が共通して持つ属性の集合が Y であり、 Y をすべて有する対象の集合が X であるとき、組 (X, Y) を形式概念と呼ぶ。

Algorithm 1 は与えられた形式文脈から全ての形式概念を発見するアルゴリズム GenerateFrom [2]であり、 $(\langle \emptyset, \emptyset \rangle, 0)$ により開始する。このアルゴリズムは再帰処理であり、その処理過程で探索木を形成する。

3. 並列処理の実装

形式概念計算の処理過程で形成される探索木

On Parallelization of Closed Pattern Mining
Masahiro Nagao, Hirohisa Seki,
Nagoya Institute of Technology

Algorithm 1 GenerateFrom($\langle A, B \rangle, y$)Input: formal concept $\langle A, B \rangle$, a number y

```

1 print  $\langle A, B \rangle$ ;
2 if  $y > |M|$  then return;
3 for  $j$  from  $y$  upto  $|M|$  do
4   if  $j \notin B$  then
5      $\langle C, D \rangle \leftarrow \text{closure}(\langle A, B \rangle, j)$ ;
6     if  $\langle C, D \rangle$  fails the canonicity test then continue;
7     call GenerateFrom( $\langle C, D \rangle, j+1$ );
8 end
9 end

```

の兄弟ノード部分は独立に実行することが可能である。Krajca らによる並列化実装(図 1 上)では、ユーザが定めたしきい値を超えない深さまでのノードを逐次処理で実行し、その先のノードから派生する処理を一時的にタスクとしてキューに保存する。しきい値までの逐次処理がすべて完了した後、並列処理を行うスレッドに保存したタスクを割り当てることですべての形式概念を計算する。

並列処理においては負荷分散が均等でない場合にスレッド間でのタスク処理時間のばらつきが生じ、処理時間の増加を引き起こす。Krajca らによる実装では、しきい値を変化させることでタスクとなる部分木の大きさや量を調整し、負荷が均等になるようにしていた。しかし、この手法には次の問題点が存在する。(1)しきい値を適切に設定できなかった場合、適切な設定の場合に比べて処理時間が増加する。(2)適切なしきい値の事前予測が困難である。

そこで本研究では、形式概念計算の探索木をノード単位で動的に分割し並列処理する、しきい値の入力を必要としない方式を実装した(Algorithm 2)。この方式では再帰呼び出しを一つのタスク単位とし、タスク内での再帰呼び出しは直接実行せず、新たなタスクとしてキューに保存する。並列処理を行うスレッドはすべての形式概念を発見するまで、このタスクの生成と消費を繰り返す。また、この方式による探索木の分割は図 1 下のようになる。

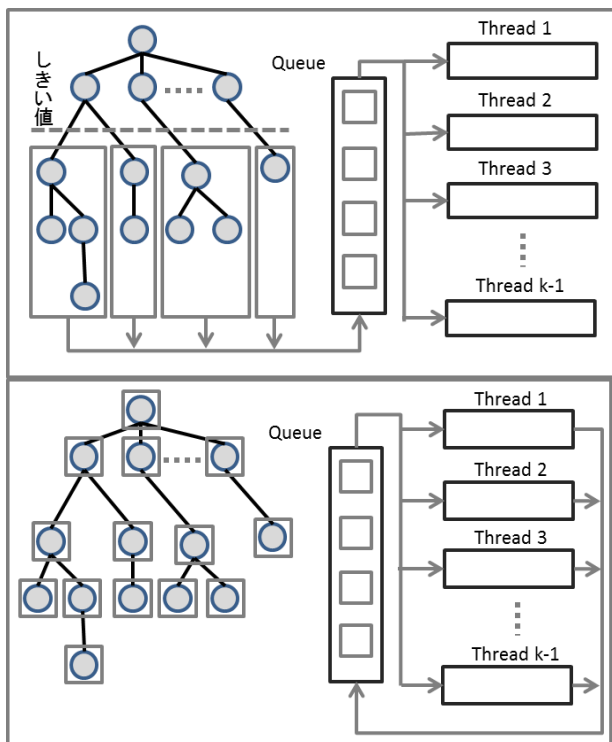


図1 二つの手法による並列処理の図式
上:Krajca らによる方式, 下:提案する方式

Algorithm 2 node_para-GenerateFrom

Input: the number of threads P

```

1 store  $(\langle \emptyset, \emptyset \rangle, 0)$  to queue  $Q$ ;
2 for each thread  $r$  ( $r = 0, \dots, P-1$ ) do in parallel
3 while poll  $(\langle A, B \rangle, y) \in Q$ 
4   if  $y > |M|$  then continue;
5   for  $j$  from  $y$  upto  $|M|$  do
6     if  $j \notin B$  then
7        $\langle C, D \rangle \leftarrow \text{closure}(\langle A, B \rangle, j)$ ;
8       if  $\langle C, D \rangle$  fails the canonicity test then continue;
9       store  $(\langle C, D \rangle, j+1)$  to queue  $Q$ ;
10    end
11  end
12 end

```

4. 実験

実験では、UCI 機械学習リポジトリで公開されている Mushroom と Anonymous Microsoft Web Data の 2 つのデータを対象にした。実験環境は CPU:AMD Opteron 4180, コア数 6, メインメモリ:32GB, OS:Ubuntu 11.04 で、Java1.7 で実装したアルゴリズムを実行しその処理時間を計測した。また、Krajca らによる従来方式においてはしきい値 L を 1 から 3 の間に動かし最適な値を見つけた。

測定結果を図 2 に示す。図中のスレッド数 1 の結果は逐次処理のものである。実験の結果、提案方式においても従来方式とほぼ同じ速度向上率を出すことができた。また、従来実装において、適切でないしきい値を選択した場合に期待される速度向上が得られなくなることを確認した。

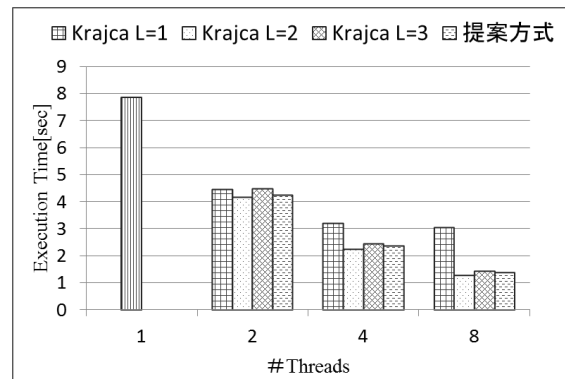


図 2a 実験結果 (Mushroom)

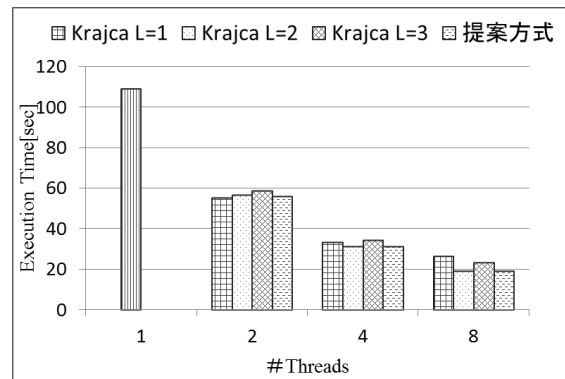


図 2b 実験結果 (Anonymous)

#Threads=1 の結果は逐次処理のもの

5. まとめ

本稿では、ユーザによるしきい値の入力を必要とせず、形式概念計算の処理負荷を均等に分散する並列化手法を実装した。また、実験により提案実装において、最適なしきい値を設定した従来実装とほぼ同じ速度向上率が出ることを確認した。今後の課題としては、スレッド数に応じた処理時間の速度向上が得られるようなデータ構造の工夫を行うことなどがある。

参考文献

- (1) 宇野 毅明 他: 統計数理, Vol.53, pp.317-329, 2005
- (2) P. Krajca., et al.: AMAI, 59(2), pp.257-272, 2010.