GPUを用いたグラフデータ分割手法

武藤　教宏† 　廣田　雅春†† 　横山　昌平††† 　石川　博†††

† 静岡大学大学院情報学研究科　〒 432–8011 静岡県浜松市中区城北 3–5–1
†† 静岡大学創造科学技術大学院　〒 432–8011 静岡県浜松市中区城北 3–5–1
††† 静岡大学情報学部　〒 432–8011 静岡県浜松市中区城北 3–5–1
E-mail: †gs11050@s.inf.shizuoka.ac.jp, ††dgs11538@s.inf.shizuoka.ac.jp ,
†††{yokoyama,ishikawa}@inf.shizuoka.ac.jp

あらまし　GPU を用いたデータ処理は，処理対象のデータが GPU のメモリサイズより大きい場合，GPU のメモリと CPU のメモリ間のデータ転送の回数が非常に多く，転送コストが大きな課題となる．データ分割は，分散処理の効率化や，データの転送コスト削減の手段として広く使われている．しかし，グラフデータのようにデータ間に関連が存在するデータを分割する場合，割り当てられていないデータを参照する回数を考慮する必要がある．また，データ処理に GPU を用いる場合，GPU のメモリのサイズや，プログラムの実行中に GPU のメモリを動的にロックすることが出来ないなどの GPU の性質を考慮する必要がある．そのため，本研究は，GPU の性質を考慮したグラフデータの分割手法の提案を行う．提案手法は，分割後の各クラスタのクラスタサイズを考慮し，ノードに隣接するノード集合の類似度に基づいたノードのクラスタリングを行う．

キーワード　GPU，グラフデータ処理，MinHash

Acceleration of Methods of Graph Partitioning Using GPU

Norihiro MUTO†, Masaharu HIROTA††, Shouhei YOKOYAMA†††, and Hiroshi ISHIKAWA†††

† Graduate School of Informatics,
Shizuoka University　3–5–1 Johoku, Naka-ku, Hamamatsu-Shi, Shizuoka, 432–8011 Japan
†† Graduate School of Science and Technology, Shizuoka University
3–5–1 Johoku, Naka-ku, Hamamatsu-Shi, Shizuoka, 432–8011 Japan
††† Faculty of Informatics, Shizuoka University
3–5–1 Johoku, Naka-ku, Hamamatsu-Shi, Shizuoka, 432–8011 Japan
E-mail: †gs11050@s.inf.shizuoka.ac.jp, ††dgs11538@s.inf.shizuoka.ac.jp ,
†††{yokoyama,ishikawa}@inf.shizuoka.ac.jp

Abstract　Graph data is widely used in represented as structured data such as the social user network. The size of graph becomes larger, so parallel and distributed processing has become very important. However, in almost every case, the data is larger than the memory size of GPU, so a very large number of data transfers between CPU's memory and GPU's memory. Therefore, the method of graph partitioning is very important. When data processing using GPU, it is necessary to consider the characteristics of GPU's memory. In this study, we propose two methods of partitioning graph data using GPU and CPU.

Key words　GPU, Graph data processing, MinHash

1. はじめに

近年，グラフデータはソーシャルネットワークにおけるユーザ間の関係などの構造を持つデータを表現する手段として広く利用されている．また，グラフデータの巨大化に伴い，複数マシンを用いた分散や，マシン内での処理の並列化などは重要となっている [9]．並列処理では，CPU と比較して演算コアの数が多く，高速なメモリアクセスが可能な GPU の有効活用が検討されている．GPU を用いたグラフデータ処理の例として，ソーシャルネットワーク分析のためのパターンマッチングなどがあげられる [7]．CPU のメモリと比較して，GPU のメモリのサイズは小さい．そのため，処理されるデータが大きくなるのに伴い，CPU メモリと GPU メモリ間のデータ転送に必要な時間は増加する．そのため，GPU を有効に活用するためには，処

理対象のデータのサイズや，処理中の使用メモリに関する制限が大きな課題となる．

処理の高速化のため，巨大なデータの分散処理は，データの分割が行われることがある．データの分割方法として，データのサイズを基準としたもの [10]，サンプリング [6]，クラスタリング [16] などがあげられる．巨大なグラフデータに対して，データ量を基準とした分割を行った場合，関連のあるデータ同士が別のブロックに分割される可能性が高くなる．そのため，分割後の処理で処理を行う各計算機に割り当てられた領域以外のデータを参照する回数が非常に多くなるという課題がある．また，サンプリングによる分割は，サンプルの抽出方法に分割結果が大きく依存するという課題がある．一方，クラスタリングによるデータの分割は，関連のあるデータが分割して配置される可能性を減少させることが可能である．分割後のデータ処理に GPU を用いる場合，分割後の各クラスタのサイズが GPU のメモリのサイズより小さいかが大きな課題となる．そのため，データ分割手法として，分割後の各クラスタののサイズを考慮する必要がある．本論文では，分割後の各クラスタのサイズを考慮したグラフデータ分割手法を提案する．提案手法では，各ノードの隣接ノードの類似度に着目する．分割手法として，既存のクラスタリング手法に基づいた手法を検討する．

本稿の章構成を説明する．2 章では関連研究について述べる．次に，3 章ではグラフデータの分割手法について説明する．4 章では，提案手法の有効性について実データをもとに作成したグラフを用いて評価する．5 章では，本論文のまとめと今後の課題を述べる．

2. 関 連 研 究

Buehrer らは，仮想ノードを用いたエッジ数の削減によるウェブグラフデータの削減手法を提案している [4]．仮想ノードの作成により，類似度が高いノード群間に頻出するエッジを削減している．類似度が高いノード群の判定において，隣接ノード集合の Jaccard 係数とノードのアドレスの類似度を利用している．Buehrer らの手法と比較して，本研究ではグラフデータの分割による分析の効率化を目標としている．

Satuluri らは，類似度が低いノード間のエッジを削減することによるグラフデータの削減手法を提案している [14]．隣接ノード集合の Jaccard 係数をノード間類似度として，類似度の低いノード間のエッジを削減することで，グラフデータの削減を行っている．削減するエッジの数は，削減前のエッジに基づいて決定している．Satuluri らの手法と比較して，本研究ではノードのクラスタリングによるデータの削減を目標としている．

巨大なデータの分析において，MinHash(Min-wise independent permutations locality sensitive hashing scheme) [3] を利用した計算量の削減が行われている [4], [14]．MinHash は，集合間の類似度の算出における計算量を大幅に削減でき，Hash 関数を共有するだけで処理の分散が可能であるため，高速化において広く利用されている．

グラフデータの分割の例として，処理の分散以外にもコミュニティ発見 [13]，可視化などがあげられる [5]．コミュニティ発見，可視化でも処理の高速化や，ノード間の類似度を算出するなど分割と同様の検討が行われている．

3. 提 案 手 法

本研究では，GPU の性質を考慮したグラフデータ分割手法として，ノードのクラスタリング手法を提案する．提案手法では，ノードを並列単位の基準とした並列処理を行う．隣接ノードの数が多いノードが処理に関係する場合，ノード間類似度の算出に必要な処理量が安定せず，高速化の妨げとなる可能性がある．そのため，類似度の算出における処理の並列度を均一にするために，隣接ノード集合を MinHash [3] を用いてハッシュ値に変換し特徴量の次元を削減する．ノード間類似度として，隣接ノード集合の Jaccard 係数を利用する．クラスタリングの手法は，凝集法と k-means 法のそれぞれに基づく 2 つの手法を提案する．

3.1 MinHash

はじめに，クラスタ間の類似度を算出するための手法について述べる．MinHash は LSH(Locality-sensitive hashing) (注1) [8] において Jaccard 係数を扱うための手法である．提案手法において利用するノード間類似度は，式 (1) である．

$$J(A,B) = \frac{|E_A \cap E_B|}{|E_A \cup E_B|} \tag{1}$$

E_A，E_B は，それぞれノード A，ノード B の隣接ノード集合を表す．

MinHash の算出において，集合間の Jaccard 係数を算出するためにハッシュ値を用いることで計算量を削減している．MinHash を用いたノード間類似度を算出するための Jaccard 係数は式 (2) で表される．

$$J'(A,B) = Pr(min\{hash(E_A)\} = min\{hash(E_B)\}) \tag{2}$$

$hash(E_A)$ は，集合内の各要素をハッシュ化する関数である．

提案手法において，同じ要素をハッシュした際に異なる値となるハッシュ関数 $(hash_1(), hash_2(), \cdots, hash_H())$ を作成し，各要素にハッシュ関数を適用したときの最小値の集合 $(min\{hash_1(E_A)\}, min\{hash_2(E_A)\}, \cdots, min\{hash_H(E_A)\})$ をクラスタリング時のノードの特徴量として利用する．

提案手法でノード隣接集合のハッシュ値を算出するアルゴリズムの概要をアルゴリズム 1 に示す．アルゴリズム中の $hash_h(e)$ は，隣接ノードを引数として受け取るハッシュ関数，$Hash\_MAX$ はハッシュ関数が返すハッシュ値の最大値とする．

提案手法では，ハッシュ関数を基づいてノード間類似度を式 (3) を用いて算出する．

$$J''(A,B) = \frac{1}{H}(\sum_{h=1}^{H} I[min\{hash_h(E_A)\} = min\{hash_h(E_B)\}]) \tag{3}$$

H は，ハッシュ関数の数である．

提案手法でノード間の類似度を算出するアルゴリズムの概

(注1)：高次元データの確率的な次元削減手法であり，類似するデータを同一のバケットに高確率でマッピングするハッシュ関数を利用する．

Algorithm 1 Compute hash value

Require: Edge list E_v
Require: Number of hashes H
Ensure: Hash list $Hash_v$
  for all $h = 1$ to H in parallel **do**
    $Hash_v[h] = HASH\_MAX$
    for all $e \in E_v$ **do**
      $Hash_v[h] = Min(hash_h(e), Hash_v[h])$
    end for
  end for

Algorithm 2 Compute jaccard coefficient

Require: Hash list $Hash_{v_1}$
Require: Hash list $Hash_{v_2}$
Require: Number of hashes H
Ensure: Jaccard coefficient $similarity$
  for all $h = 1$ to H **do**
    $similarity = 0$
    if $Hash_{v_1}[h] = Hash_{v_2}[h]$ **then**
      $similarity = similarity + 1$
    end if
  end for
  $similarity = similarity/H$

要をアルゴリズム 2 に示す．アルゴリズム中の $similarity$ は，v_1，v_2 の Jaccard 係数である．

3.2 凝集法に基づくグラフデータ分割手法

凝集法は，各データ間の類似度に基づいてクラスタリングを行うため，類似するデータが同一のクラスタに属しやすいという利点がある．そのため，クラスタのサイズを一定以下に制限する本研究では，本来 1 つになるクラスタがサイズ制限により分割される場合，各クラスタに類似するデータを配置可能になると考えられる．また，クラスタリングの進行に伴い，最終的なクラスタリング結果が局所的最適解になる場合がある．そのため，凝集するクラスタの決定は，全てのクラスタを並行して行う．これにより，並列度がクラスタ数と等しくなるため，GPU に適した処理となると考えられる．また，凝集を行う 2 クラスタの決定は，凝集後のクラスタサイズが制限以内であり，クラスタリングのループ内での各クラスタの凝集回数が閾値以下であるかを判定して行う．類似度が最大のクラスタが複数個存在し，そのうちの 1 つのクラスタが条件を満たさない場合，2 つ目以降のクラスタが条件を満たすかの判定を行う．これにより，類似度が高いノード同士がクラスタサイズの制限によって同一のクラスタに属さない可能性が減少すると考えられる．また，クラスタの探索，凝集を並列化できるため，GPU に適した処理が可能になると考えられる．

凝集するクラスタを決定するアルゴリズムをアルゴリズム 3 に示す．アルゴリズム中の Sim は，クラスタごとに算出した類似度とクラスタ番号の組み合わせを類似度の降順に並び変えたリスト，MCS は，クラスタリング後のクラスタの最大サイズ，MAN は，ループ内でのクラスタの最大凝集回数を表す．

凝集するクラスタの決定後，クラスタのハッシュ値を算出する．凝集後のクラスタのハッシュ値として，凝集前の各クラスタのハッシュ値の最小値を利用する．凝集前のハッシュ値を凝集後のハッシュ値として用いることで，各ノードの隣接ノードを参照するコストを削減する．凝集後のクラスタのハッシュ値

Algorithm 3 Decide agglomerative combination

Require: Vertex list V
Require: Similarity list Sim
Require: Cluster lists $Cluster$
Require: Cluster numbers $Cluster\_num$
Require: Number of max cluster size MCS
Ensure: Cluster lists $Cluster$
Ensure: Cluster numbers $Cluster\_num$
  $Agg_i(i \in V) = 0$
  for all $v_1 \in V$ in parallel **do**
    for $i = 1$ to $|Sim_{v_1}|$ **do**
      $v_2 = \mathrm{GetVertexNumber}(Sim_{v_1}, i)$
      $c_1 = Cluster\_num_{v_1}$
      $c_2 = Cluster\_num_{v_2}$
      if $\mathrm{GetSimilality}(Sim_{v_1}, i) = \mathrm{GetSimilality}(Sim_{v_1}, 1)$
          And $c_1 \neq c_2$
          And $|Cluster_{c_1}| + |Cluster_{c_2}| < MCS$ **then**
        Add $v_3 (v_3 \in Cluster_{c_2})$ to $Cluster_{c_1}$
        $Cluster\_num_{v_3} (v_3 \in Cluster_{c_2}) = c_1$
        $Agg_{c_1} = Max(Agg_{c_1}, Agg_{c_2}) + 1$
        delete $Cluster_{c_2}$ in $Cluster$
      end if
    end for
  end for

Algorithm 4 Compute hash value after aggregation

Require: Hash list $Hash_{v_1}$
Require: Hash list $Hash_{v_2}$
Require: Number of hashes H
Ensure: Hash list $Hash_{v_1}$
  for all $h \in H$ in parallel **do**
    $Hash_{v_1}[h] = Min(Hash_{v1}[h], Hash_{v_2}[h])$
  end for

は，凝集前の各クラスタの隣接ノードのハッシュ値を算出した場合と等しくなる．凝集後のクラスタのハッシュ値を算出するアルゴリズムをアルゴリズム 4 に示す．

凝集法に基づくグラフデータ分割アルゴリズムの概要をアルゴリズム 5 に示す．提案手法は，クラスタサイズの制限によりクラスタリングが停止するため，停止条件を一般的な凝集法の停止条件であるクラスタ数が一定の数を下回った場合ではなく，クラスタ数の収束としている．ハッシュ値の算出は，各ノードの隣接ノード集合をアルゴリズム 1 を用いて，ノードとハッシュ関数ごとに並行して行い，$Hash$ に格納する．Jaccard 係数の算出は，全ノードの組み合わせについてアルゴリズム 2 を用いてクラスタごとに並行して行い，類似度とクラスタ番号の組み合わせを Sim に格納する．類似度を各クラスタごとに降順に並び変え，各クラスタから凝集するクラスタの決定をアルゴリズム 3 を用いて並行して行う．凝集後のクラスタのハッシュ値の算出は，クラスタ番号がノード番号と異なるもの (ループ中に凝集され，別のクラスタとなったクラスタ) についてアルゴリズム 4 を用いて行う．

3.3 k-means 法に基づくグラフデータ分割手法

k-means に基づくグラフデータ分割手法は，クラスタ情報を共有することで，並列度をノード数にすることが可能であるため，並列処理に適した GPU に適していると考えられる．提案するアルゴリズムでは，各ノードのクラスタへの割り当ては，GPU への適用のために，ノードを並列単位として並行して行う．また，クラスタリング後のクラスタのサイズが一定以上に

Algorithm 5 Graph clustering based on hierarchical agglomerative clustering

Require: A graph $G = (V, E)$
Require: Number of hashes H
Require: Number of max cluster size MCS
Ensure: Cluster lists $Cluster$
  $Hash_v(v \in G)$=Compute hash value (E_v, H) in parallel
  Add $v(v \in G)$ to $Cluster_v$
  $Cluster\_num_v(v \in G) = v$
  repeat
    $old\_N = |G|$
    Sim=Compute jaccard similarity
      $(v_1(v_1 \in G), v_2(v_2 \in G), H)$ in parallel
    Sort Sim by descending order of similarity in parallel
    Decide agglomerative combination
    for all $v_2 \in G$ in parallel **do**
      if $Cluster\_num_{v_2} \neq v$ **then**
        $v_1 = Cluster\_num_{v_2}$
        $Hash_{v_1}$=Compute hash value after aggregation
          $(Hash_{v_1}, Hash_{v_2}, H)$
        delete v_2 in G
      end if
    end for
  until $|G| = old\_N$

Algorithm 6 Dicide cluster number

Require: Number of vertices list V
Require: Number of clusters K
Require: Similarity list Sim
Require: Number of max cluster size MCS
Ensure: Node clusters $Cluster$
Ensure: Cluster numbers $Cluster\_num$
  $changed = 0$
  for all $v \in V$ in parallel **do**
    for all $i = 1$ to K **do**
      if GetSimilality(Sim_v, i) = GetSimilality$(Sim_v, 1)$ **then**
        c = GetClusterNumber(Sim_v, i)
        if $|Cluster_c| < MCS$ **then**
          $Cluster\_num_v = c$
          Add v to $Cluster_c$
          break
        end if
      else
        $Cluster\_num_v = K + 1$
      end if
    end for
  end for

ならないように制限を加えている．そのため，ノードをクラスタに割り当てる処理の途中でクラスタサイズが制限サイズと等しくなるクラスタが発生する．類似度が最大のクラスタが複数存在し，そのうちの 1 つのクラスタが条件を満たさない場合，2 つ目以降のクラスタが条件を満たすかの判定を行う．適切なクラスタが存在しない場合，クラスタリングのループ中はそのノードをクラスタへ割り当てない．

各ノードのクラスタ番号を決定するアルゴリズムをアルゴリズム 6 に示す．アルゴリズム中の Sim は，クラスタごとに算出した類似度とクラスタ番号の組み合わせを類似度の降順に並び変えたリスト，MCS は，クラスタリング後のクラスタの最大サイズを表す．

割り当てるクラスタの決定後，クラスタのハッシュ値を変更する．変更後のクラスタのハッシュ値として，クラスタに属す

Algorithm 7 Graph clustering based on k-means

Require: A graph $G = (V, E)$
Require: Random cluster $C = (K, E)$
Require: Number of hashes H
Require: Number of max cluster size MCS
Ensure: Node clusters $Cluster$
  $Hash_v(v \in G)$=Compute hash value (E_v, H) in parallel
  $C\_Hash_v(v \in C)$=Compute hash value (E_v, H) in parallel
  $Cluster\_num_v(v \in G) = |C| + 1$
  repeat
    Sim=Compute jaccard similarity
      $((v_1(v_1 \in G), v_2(v_2 \in C), H)$ in parallel
    Sort Sim by descending order of similarity in parallel
    Decide cluster number
    $Old\_Hash = C\_Hash$
    C_Hash=HASH_MAX
    for all $v \in |G|$ **do**
      if $Cluster\_num_v \neq |C| + 1$ **then**
        $c = Cluster\_num_v$
        C_Hash_c=Compute hash value after clustering
          $(C\_Hash_c, Hash_v, H)$
      end if
    end for
    $changed = 0$
    for all $v \in C$ **do**
      for all $h \in H$ **do**
        if $Old\_Hash_v[h] \neq C\_Hash_v[h]$ **then**
          $changed++$
        end if
      end for
    end for
  until $changed = 0$

る各ノードのハッシュ値の最小値を利用する．各ノードのハッシュ値を変更後のハッシュ値として用いることで，各ノードの隣接ノードを参照するコストを削減する．変更後のクラスタのハッシュ値は，クラスタに属する各ノードの隣接ノードのハッシュ値を算出した場合と等しくなる．割り当てるクラスタの決定後のクラスタのハッシュ値を算出するアルゴリズムは，凝集法に基づくグラフデータ分割手法で用いるアルゴリズム 4 と同様である．

k-means 法に基づくグラフデータ分割アルゴリズムの概要をアルゴリズム 7 に示す．ハッシュ値の算出は，各ノードの隣接ノード集合をアルゴリズム 1 を用いてノード，ハッシュ関数ごとに並行して行い，$Hash$ に格納する．クラスタの初期値として，ランダムに複数の隣接ノードを設定したノードを作成する．クラスタのハッシュ値の算出はグラフと同様にアルゴリズム 1 を用いてクラスタ，ハッシュ関数ごとに並行して行い，C_Hash に格納する．Jaccard 係数の算出は，ノードとクラスタの組み合わせについてアルゴリズム 2 を用いてノード，クラスタごとに並行して行い，類似度とクラスタ番号の組み合わせを Sim に格納する．類似度を各ノードごとに降順に並び変え，各ノードから割り当てられるクラスタの決定をアルゴリズム 6 を用いて並行して行う．割り当てるクラスタの決定後，クラスタのハッシュ値の算出をアルゴリズム 4 を用いて行う．ノードが 1 つも割り当てられていないクラスタは，次のループにおける各ノードのクラスタへの割り当て処理で割り当てる判定の対象としない．そのため，提案手法は，入力されたクラスタ数 K と出力時のクラスタ数が一致しない可能性がある．

表 2　収集した Twitter のユーザ情報を用いて作成したグラフの概要

|V|	|E|
1,024	9,612
4,096	84,055
16,384	899,037
65,536	12,202,276

4. 実　　験

本論文では，グラフデータに対してグラフの分割を行い，提案した 2 つのグラフデータ分割手法が有効であることを実験により検証する．また，提案手法を用いた分割手法の処理に CPU を用いた場合と，CPU と GPU を用いて処理した場合を比較して，GPU を用いることで，グラフデータ分割を高速化可能であることを検証する．また，GPU を用いることで，提案手法が高速にグラフを分割可能であることを示す．比較手法として，Modularity クラスタリング [11] を用いる．実験に用いた装置の概要を表 1 に示す．

4.1 分割手法およびパラメータ

本実験では，4 種類の分割手法を評価する．1 つ目は，提案手法の凝集法に基づくグラフデータ分割手法を GPU と CPU を用いて実行する手法 (HAC(GPU))(クラスタの結合を行う処理のみ CPU で実装)．2 つ目は，提案手法の k-means 法に基づくグラフデータ分割手法を GPU と CPU を用いて実行する手法 (k-means(GPU))．3 つ目は，提案手法の凝集法に基づくグラフデータ分割手法を CPU を用いて実行する手法 (HAC(CPU))．4 つ目は，提案手法の k-means 法に基づくグラフデータ分割手法を CPU を用いて実行する手法 (k-means(CPU))．5 つ目は，比較手法として，Modularity クラスタリング (Modularity) を評価する．

5 つの手法では，分割後の各クラスタの最大のノード数 (クラスタの最大サイズ) をパラメータとして設定する．加えて，k-means に基づいたクラスタリングでは，クラスタ数 K をパラメータとして設定する．実験では，各クラスタリング手法とパラメータの組み合わせについて複数回実行したときの実行結果の平均値を実験結果としている．各グラフに対する分割において，各クラスタの最大のノード数は，グラフのノード数の 1/8，1/4，1/2 の 3 つのパラメータでグラフデータの分割を行う．k-means を用いるクラスタリングの実験結果は，複数の K でクラスタリングを実行し，評価指標の値が最も高くなった結果を選択する．

4.2 データセット

本実験では，有向グラフをデータセットとして用いる．データセットの概要を表 2 に示す．グラフは，Twitter(注2) のユーザネットワークをクロールし，収集したユーザデータからランダムにノードを選択することで作成した．ノードの選択は，グラフ内の他のノードと 1 つ以上のエッジを有しているノードの中からランダムにノードを選択した．データセットの各グラフは，非連結グラフである．

(注2)：https://twitter.com/

4.3 評 価 指 標

実験では，各手法を用いて実行したクラスタリング結果に対して，4 種類の評価指標を用いて評価を行う．評価指標として，Silhouette coefficient [12]，coverage [2]，perfomance [15]，および inter-cluster conductance [1] を用いた．4 つの評価指標を用いて，分割前のグラフのノードの隣接関係と各ノードに隣接するノードの類似度がクラスタリング結果に適切に反映されているかを検証する．各評価指標は，値が高い場合，その指標が想定する良いクラスタリング結果であることを示す．

4.3.1 Silhouette coefficient

Silhouette coefficient は，クラスタリング結果の分離性を各ノードが属するクラスタと 2 番目に属するべきクラスタ間の非類似度を用いて評価する指標である．Silhouette coefficient が想定する分離性が高いクラスタリング結果とは，類似度が高いノードが同一のクラスタに属しており，類似度が低いノードが異なるクラスタに属しているクラスタリング結果である．Silhouette coefficient は，各ノードが属するクラスタとそのクラスタ内の他のノードとの非類似度の平均，各ノードが 2 番目に属するべきクラスタに属する各ノードとの非類似度の平均の 2 つを評価に用いる．各ノードが 2 番目に属するべきクラスタとは，ノードが属するクラスタ以外のクラスタの中で，ノードとそのクラスタに属するノードとの非類似度の平均が最も低いクラスタである．各ノードと 2 つのクラスタ間の非類似度を用いて，クラスタリング結果の分離性の評価を行う．本論文では，非類似度 $dissim(u,v)$ は，(1) を用いて算出した $J(A,B)$ を 1 から引いた値を用いる．Silhouette coefficient は，以下の式 (4) で算出する．

$$SC = \frac{1}{|C|}\sum_{C_i \in C}\frac{1}{|C_i|}\sum_{u \in C_i}\frac{|b(u)-a(u)|}{max(b(u),a(u))} \qquad (4)$$

$a(u)$ は，各ノードとそのノードが属するクラスタ内の他のノードとの非類似度の平均を表す．$b(u)$ は，各ノードとそのノードが 2 番目に属するべきクラスタとの非類似度の平均を表す．

ノード u と属するクラスタ C_i の非類似度 $a(u)$ 算出する式を (5)，各ノードが 2 番目に属するべきクラスタとの非類似度 $b(u)$ を算出する式を (6) に示す．

$$a(u) = \frac{1}{|C_i|-1}\sum_{v \in C_i, v \neq u} dissim(E(u),E(v)) \qquad (5)$$

$$b(u) = \min_{C_j \in C, j \neq i}\frac{1}{|C_j|}\sum_{v \in C_j} dissim(E(u),E(v)) \qquad (6)$$

$E(u)$，$E(v)$ はそれぞれ分割前のグラフにおいて，ノード u，v に隣接するノードの集合を表す．

4.3.2 coverage

coverage は，クラスタリング結果を分割後の各クラスタ内に存在する分割前のグラフに存在したエッジ数と分割前のグラフの全エッジ数の比を用いて評価する指標である．coverage は，クラスタリング結果のクラスタ数と負の相関が存在し，グラフを多数のクラスタに分割した場合，値が低くなる傾向がある．coverage は，クラスタリング結果を用いてデータの処理を行う際に，他のクラスタに属するデータを参照する頻度を表す．

表 1　実験に用いた装置

	HAC(GPU)	k-means(GPU)	HAC(CPU)	k-means(CPU)	Modularity
CPU	AMD Opteron 1222 dual-core		Intel Core i7 975 EE quad-core		
GPU	NVIDIA GTX 460 7SM 768MB				
Memory	8GB		12GB		
OS	Windows 7		Fedora 12		
コンパイラ (C++)	C/C++ Optimizing Compiler Version 15.00.21022.08 for x64		GNU コンパイラ (v2.4)		
コンパイラ (GPU)	CUDA toolkit 5.0				

coverage は，以下の式 (7) で算出する．

$$cov = \sum_{C_i \in C} \frac{Intra\_edge(i)}{|E|} \tag{7}$$

$Intra\_edge(i)$ は，クラスタ Ci に属するノード間に存在するエッジの数を表す．

$Intra\_edge(i)$ は，以下の式 (8) で算出する．

$$Intra\_edge(i) = \sum_{u,v \in C_i, u \neq v} E(u,v) \in G \tag{8}$$

$E(u,v)$ は，グラフ G でエッジを持つノードの組み合わせの集合を表す．

4.3.3　performance

performance は，クラスタリング結果を分割後の各クラスタ内に存在する分割前のグラフに存在したエッジ数，分割前のグラフにおいて非隣接関係にあったノードが別のクラスタに属している数の合計を用いて評価する指標である．performance は，ノード間のグラフ上での隣接関係に基づいてクラスタリング結果の分離性を評価する指標として利用可能である．coverage の評価に加えて，performance は，隣接していないノードが異なるクラスタに属しているかも評価する．そのため，クラスタリング結果のクラスタ数が少ない場合に高い値となりやすい coverage と比較して，performance は，非隣接関係のノードが別のクラスタに配置されている場合に高い値となる．そのため，クラスタリング結果のクラスタ数が少ない場合，低い値となる可能性がある．performance は，以下の式 (9) で算出する．

$$\begin{aligned} perf = & \frac{\sum_{C_i \in C} Intra\_edge(i)}{|V|(|V|-1)} \\ & + \frac{\sum_{C_i \in C, C_j \in C, i \neq j} Non\_adjacent(i,j)}{|V|(|V|-1)} \end{aligned} \tag{9}$$

$Non\_adjacent(i,j)$ は，クラスタ C_i，C_j に属する各ノードの組み合わせについて，分割前のグラフで隣接していなかったノードの数を表す．

$Non\_adjacent(i,j)$ を算出する式を式 (10) に示す．

$$Non\_adjacent(i,j) = \sum_{u \in C_i, v \in C_j} E(u,v) \notin G \tag{10}$$

4.3.4　inter-cluster conductance

inter-cluster conductance は，クラスタリング結果を分割前に各ノードに隣接していたノードの数と分割後のクラスタ内に存在する隣接していたノードの数を用いて評価する指標である．inter-cluster conductance は，クラスタごとに算出し，最も分割によって各ノードに隣接するノードが減少したクラスタを評価の対象とする．そのため，inter-cluster conductance は，クラスタリングの結果，ノード数の少ないクラスタが作成された場合，

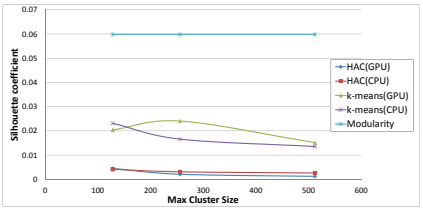

図 1　Number of Node 1024，Silhouette coefficient

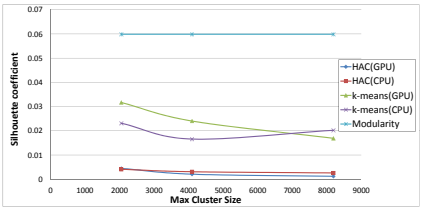

図 2　Number of Node 16384，Silhouette coefficient

非常に低い値となる可能性が存在する．inter-cluster conductance は，inter-cluster conductance は，以下の式 (11) で算出する．

$$icc = 1 - \max_{C_i \in C} \frac{\sum_{u \in C_i} E(u) - Intra\_edge(i)}{min(\sum_{u \in C_i} E(u), |E| - \sum_{u \in C_i} E(u))} \tag{11}$$

4.4　実験結果

データセットの 2 グラフに対して，5 つの分割手法を適用した際のクラスタリング結果に Silhouette coefficient を適用した結果をノード数 1024 の場合を図 1，ノード数 16384 の場合を図 2 に示す．Modularity と比較して，提案した 2 手法は，低い値である．k-means(GPU) が HAC(GPU) より高い値となった原因として，クラスタの最大サイズによって，同一のクラスタとなるノードが複数のクラスタに分割されるときの差が考えられる．HAC(GPU) では，クラスタが結合される順番に依存してクラスタリング結果が確定する．一方，k-means(GPU) では，複数のクラスタのハッシュが完全に一致しなければ，ノードを適切にクラスタリング可能である．そのため，クラスタの結合順序とクラスタリング結果に依存関係が存在する，HAC(GPU) と比較して，k-means(GPU) の値が高くなったと考えられる．HAC(CPU) と比較して，HAC(GPU) は，クラスタの最大サイズが大きくなるのに伴い，同程度から低い値に変化している．

データセットの 2 グラフに対して，5 つの分割手法を適用した際のクラスタリング結果に coverage を適用した結果をノード数 1024 の場合を図 3，ノード数 16384 の場合を図 4 に示す．Modularity と比較して，HAC(GPU) は，高い値である．また，Modularity と比較して，k-means(GPU) は，同程度，あるいは高い値である．GPU を用いた場合と CPU を用いた場合の比較では，提案した 2 つの手法の値は，クラスタの最大サイズと類似する傾向を示している．このことより，提案した手法を実行

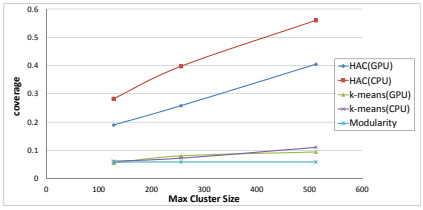

図 3　Number of Node 1024，coverage

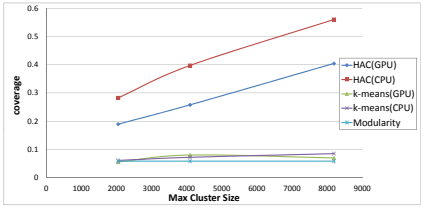

図 4　Number of Node 16384，coverage

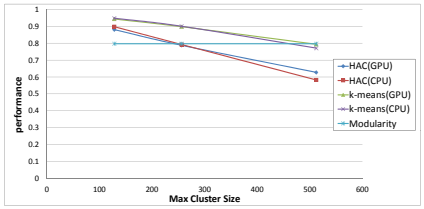

図 5　Number of Node 1024，performance

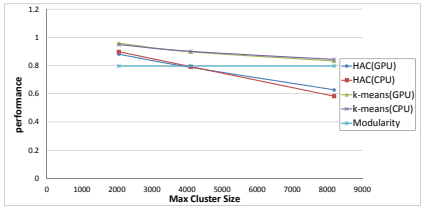

図 6　Number of Node 16384，performance

する際に GPU を用いる場合と CPU を用いる場合の差は，少ないと考えられる．

データセットの 2 グラフに対して，5 つの分割手法を適用した際のクラスタリング結果に performance を適用した結果をノード数 1024 の場合を図 5，ノード数 16384 の場合を図 6 に示す．Modularity と比較して，HAC(GPU) は，クラスタの最大サイズが小さい場合，高い値であるが，クラスタの最大サイズの増加に伴い，低い値に変化している．また，k-means(GPU) は，クラスタの最大サイズの変化に対して，HAC(GPU) と同様の変化をしている．Modularity の値は，クラスタの最大サイズの変化に関わらず，固定である．これは，Modularity のクラスタリング結果の最大のクラスタのサイズが各グラフの 1/8 以下であり，クラスタの最大サイズの制限によってクラスタリング結果が変化していないためである．一方，Modularity 以外の各手法は，クラスタの最大サイズが増加するのに伴い，クラスタリング結果のクラスタ数が増加し，performance が低下する傾向が存在する．

データセットの 2 グラフに対して，5 つの分割手法を適用した際のクラスタリング結果に inter-cluster conductance を適用し

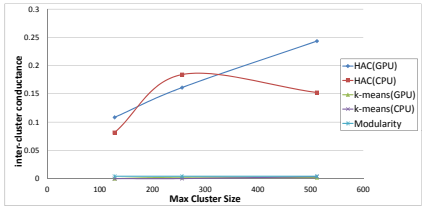

図 7　Number of Node 1024，inter-cluster conductance

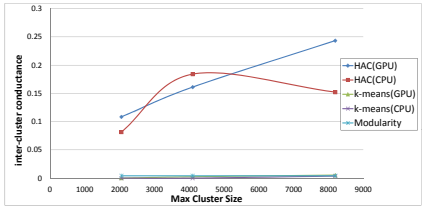

図 8　Number of Node 16384，inter-cluster conductance

た結果をノード数 1024 の場合を図 7，ノード数 16384 の場合を図 8 に示す．HAC(CPU)，HAC(GPU) 以外の各手法の結果は，低い値である．原因として，Modularity は，クラスタ間の隣接関係に基づいて，結合するクラスタを決定してるが，停止において，Modularity の増減のみに着目するため，疎なクラスタがクラスタリング結果に残りやすいことが原因と考えられる．k-means(GPU)，k-means(CPU) は，ノード間でのハッシュの比較ではなく，クラスタを基準としてクラスタリングを行っていることが原因であると考えられる．

データセットの各グラフに対して 5 つの分割手法を用いて分割を行った際のクラスタのと実行時間のグラフを図 9 に示す．グラフデータの分割処理の実行時間の比較では，Modularity と比較して，HAC(GPU)，k-means(GPU) の実行時間は，それぞれ短時間である．また，GPU を用いた場合と CPU を用いた場合の比較において，HAC(CPU)，k-means(CPU) それぞれと比較して，HAC(GPU)，k-means(GPU) は，短時間である．特に，k-means(GPU) は，非常に短時間で処理を停止しており，高速にグラフデータ分割が可能であることを示している．実行時間の分析では，クラスタリングの停止までに類似度の算出とクラスタの結合を行うループについて，HAC(CPU) は，平均 24.67 回実行したのに対して，HAC(GPU) は，平均 254 回であった．ループ回数は大きく異なるが，ループ処理の多くは，クラスタ数が減少してから発生しているため，実行時間にあまり差が発生しなかったと考えられる．また，GPU を用いた場合と CPU を用いた場合の比較において，k-means(CPU) と比較して，k-means(CPU) は，短時間である．処理に GPU を用いた場合と CPU を用いた場合の比較より，提案した各手法は，GPU を用いて処理が高速化可能である．

4.5 考　　察

データセットに対するグラフデータ分割結果のまとめとして，Modularity と比較して，提案した各手法は，Silhouette coefficient において低い値である．coverage，performance は，評価値とクラスタ数との相関が高く，クラスタの最大サイズに依存して，同程度の精度を得られる可能性がある．inter-cluster conductance

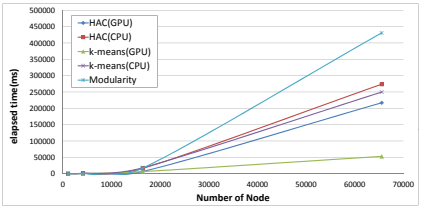

図 9　データセットの各グラフに分割手法を適用した際の実行時間

では，Modularity と比較して，k-means(GPU) が同程度，あるいは低い値であるのに対して，HAC(GPU) が高い値である．Modularity と HAC(GPU) の実行時間の比較では，HAC(GPU) は，短時間で処理が実行できているため，精度面の課題について，結合するクラスタの決定方法の変更や，類似度を算出する 2 クラスタを隣接したクラスタのみにするなどの改良が考えられる．また，CPU を用いた場合と比較し，GPU を用いた分割では，類似度の算出とクラスタの結合を行うループを多数実行しており，実行時間が長くなっていると考えられる．そのため，クラスタ数の減少が小さくなった時点で，他のクラスタリング手法に切り替えるなどの工夫を行うことで分割処理の高速化が可能であると考えられる．Modularity と k-means(GPU) の実行時間の比較では，k-means(GPU) は，非常に短時間で処理が実行できているが，精度面での改善が考えられる．データセットの各グラフへの分割処理において，k-means(GPU) は，非常に短時間で処理を実行できており，クラスタのハッシュ値が収束するまでに要するループ回数と実行時間の関連が非常に強いことが確認できた．Modularity のクラスタリング精度について，Modularity クラスタリングが Modularity の増減のみに着目して，結合するクラスタを決定しているため，クラスタの最大サイズを大きくしても，一部の指標の値が改善しない場合が存在することが確認できた．

5. おわりに

グラフデータの分割手法として，既存の 2 つのクラスタリング手法に基づいた手法を提案した．各手法は，分割後の各クラスタに対して GPU を用いて効率的に処理することを目標に，クラスタの最大サイズを任意に変更可能な手法である．凝集法では，クラスタの探索を並行して行うことで，クラスタリング結果が局所的最適解となることを抑制している．また，k-means 手法では，クラスタに割り当てられたノードがクラスタの最大サイズを超える場合，類似度が等しい別のクラスタに割り当てを行う．これにより，特徴量が類似するクラスタを複数作成し，クラスタの最大サイズ以上のクラスタ内のノードを複数のクラスタに分割して配置する．提案した手法は，実験より，既存のクラスタリング手法と比較し，クラスタリングの精度は，多少劣っているが，高速にクラスタリングが可能な手法であることを示した．そのため，提案手法は，グラフデータの分割を行う場合に有効であると考えられる．

今後の課題として，クラスタのハッシュ値を最小値から最頻値に変更する，クラスタリング時にノード間の距離を考慮する，重み付きのグラフに対応することなどがあげられる．また，分割後のデータについて分析手法を適用することなどがあげられる．

文　献

[1] Engineering graph clustering: Models and experimental evaluation. *J. Exp. Algorithmics*, Vol. 12, pp. 1.1:1–1.1:26, June 2008.

[2] Ulrik Brandes and Thomas Erlebach. *Network Analysis: Methodological Foundations (Lecture Notes in Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[3] A. Broder. On the Resemblance and Containment of Documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, SEQUENCES '97, pp. 21–, Washington, DC, USA, 1997. IEEE Computer Society.

[4] Gregory Buehrer and Kumar Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, WSDM '08, pp. 95–106, New York, NY, USA, 2008. ACM.

[5] Stéphan Clémençon, Hector De Arazoza, Fabrice Rossi, and Viet-Chi Tran. Visual mining of epidemic networks. In *Proceedings of the 11th international conference on Artificial neural networks conference on Advances in computational intelligence - Volume Part II*, IWANN'11, pp. 276–283, Berlin, Heidelberg, 2011. Springer-Verlag.

[6] Munmun De Choudhury, Yu-Ru Lin, Hari Sundaram, K. Selcuk Candan, Lexing Xie, and Aisling Kelliher. How Does the Data Sampling Strategy Impact the Discovery of Information Diffusion in Social Media? In *Proceedings of the 4th International AAAI Conference on Weblogs and Social Media*, May 2010.

[7] Wenfei Fan. Graph pattern matching revised for social network analysis. In *Proceedings of the 15th International Conference on Database Theory*, ICDT '12, pp. 8–21, New York, NY, USA, 2012. ACM.

[8] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity Search in High Dimensions via Hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pp. 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[9] Michael Isard and Yuan Yu. Distributed data-parallel computing using a high-level programming language. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, SIGMOD '09, pp. 987–994, New York, NY, USA, 2009. ACM.

[10] Toralf Kirsten, Lars Kolb, Michael Hartung, Anika Gross, Hanna Köpcke, and Erhard Rahm. Data Partitioning for Parallel Entity Matching. In *8th International Workshop on Quality in Databases*, 2010.

[11] Andreas Noack and Randolf Rotta. Multi-level Algorithms for Modularity Clustering. In *Proceedings of the 8th International Symposium on Experimental Algorithms*, SEA '09, pp. 257–268, Berlin, Heidelberg, 2009. Springer-Verlag.

[12] Peter Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.*, Vol. 20, No. 1, pp. 53–65, November 1987.

[13] Venu Satuluri and Srinivasan Parthasarathy. Scalable graph clustering using stochastic flows: applications to community discovery. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pp. 737–746, New York, NY, USA, 2009. ACM.

[14] Venu Satuluri, Srinivasan Parthasarathy, and Yiye Ruan. Local graph sparsification for scalable clustering. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, pp. 721–732, New York, NY, USA, 2011. ACM.

[15] S. M. van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, The Netherlands, 2000.

[16] Ren Wu, Bin Zhang, and Meichun Hsu. Clustering billions of data points using GPUs. In *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*, UCHPC-MAW '09, pp. 1–6, New York, NY, USA, 2009. ACM.