効率的な分散グラフ処理のためのグラフ分割

藤森　俊匡† 　塩川　浩昭†† 　鬼塚　真†

† 大阪大学大学院情報科学研究科　〒 565–0871 大阪府吹田市山田丘 1 丁目 5 番
†† 筑波大学 計算科学研究センター　〒 305–8573 茨城県つくば市天王台 1 丁目 1 番 1 号
E-mail: †{fujimori.toshimasa,onizuka}@ist.osaka-u.ac.jp, ††shiokawa@cs.tsukuba.ac.jp

あらまし　実世界で扱われるグラフデータの大規模化に伴って分散グラフ処理技術の重要性が高まっている．分散グラフ処理では，グラフデータを分割し複数の計算機上に割り当ててから分析処理を行うが，このグラフ分割の品質によって分析処理の性能は大きく左右される．そこで本稿では，分散グラフ処理における分析処理を高速化するためのグラフ分割手法を提案する．本手法は，Modularity に基づくクラスタリングを拡張することで，通信発生の要因となる切断エッジ数を削減すると共に計算負荷の均衡化も達成する．また，最新の逐次割り当て手法である HDRF と組み合わせることで，さらなる通信コストの削減を実現する．我々は提案手法を既存の分散グラフ処理フレームワーク上に実装し性能比較を行った．その結果，HDRF のみを用いた場合と比較して PageRank の分析処理を最大 3.2 倍高速化できることを確認した．

キーワード　グラフ分割, グラフマイニング, 分散処理

1. はじめに

近年，情報通信技術の進歩に伴って，実世界で扱われるグラフデータの大規模化が進んでいる．例えば，Facebook の 1 日当たりの平均アクティブユーザ数は，2016 年 12 月時点で 12.3 億人であったと報告されている (注1) が，Facebook におけるユーザの友達関係は巨大なグラフデータとして表現できる．このような SNS から得られるグラフデータは，類似ユーザの検出や，類似ユーザの情報を元にした適切な広告掲示などの目的で使用される．グラフデータはこの他にも，ウェブページのリンク関係や道路ネットワーク，分子構造やユーザの商品に対する購買（評価）履歴などさまざまな分野に存在し，これらのグラフデータは今後もさらに大規模化が進むと予想される．

大規模なグラフデータに対する分析処理をより効率的に行うために，近年，数多くの分散グラフ処理フレームワークが提案されてきた [8] [10] [11] [21]．これらのフレームワークは共通して，グラフデータを分割し，得られた部分グラフを各計算機に配置してから目的の分析処理を実行する．ここで，最初のグラフ分割は，分析処理中の通信コストおよびロードバランスに大きな影響を与える重要な要素である [15]．例えば，グラフ分割によって切断された頂点またはエッジは計算機間をまたがって存在することになるが，分析処理時にはそれらを通じて計算機間で通信が発生する．したがって，グラフ分割では通信コストを抑えるために切断エッジまたは切断頂点の数を少なくすることが求められる．また，グラフ分析アルゴリズムの多くは，処理するエッジ数が多いほど計算負荷が大きくなる [18] ため，各計算機に割り当てるエッジ数の偏りが大きいと，各計算機の計算負荷の偏りも大きくなる．その場合，処理の早い計算機の待機時間が増大し非効率的であるため，グラフ分割では導出する部分グラフの内包エッジ数の偏りが小さいことが求められる．ここで本稿では，各部分グラフの内包エッジ数の偏りが小さい状態のことを等粒度性が高いと表現することとする．また，切断エッジまたは切断頂点の数が少なく，かつ等粒度性が高くなるようなグラフ分割のことを，品質の高いグラフ分割と表現することとする．

分散グラフ処理のための代表的なグラフ分割手法として，Oblivious [8] や HDRF [14] などが挙げられる．これらの手法はグラフデータの読み込み時点で頂点やエッジの割り当て先を逐次決定することにより，大規模なグラフデータを高速に分割することができる．また，これらの手法は頂点を切断してエッジをいずれかの計算機に割り当てる vertex-cut 方式のグラフ分割手法である．vertex-cut 方式は，エッジを切断して頂点をいずれかの計算機に割り当てる edge-cut 方式に比べ負荷分散に優れており [2] [8]，Oblivious と HDRF も多くの場合において最適に近い等粒度性を実現することができる [8] [14]．しかし，これらの手法は負荷分散には優れているものの通信コストは大きくなる傾向があり，それが分析処理性能を悪化させる要因となりやすい．

したがって本稿では，通信コストの削減と計算負荷の均衡化とをバランスよく両立することで，分散グラフ処理における分析処理を高速化するグラフ分割手法を提案する．提案手法では，切断エッジ数を削減するために Modularity [12] という指標を用いる．Modularity はクラスタリングの良さを評価するための指標であり，一般的にクラスタ内のエッジが密，クラスタ間のエッジ（切断エッジ）が疎であるほど高い値を示す．提案手法は，Modularity に基づくクラスタリングを拡張することで，切断エッジ数が少なく，かつ等粒度性の高いクラスタを導出することを可能とする．また，得られた edge-cut 方式のクラスタ

(注1)：https://investor.fb.com/investor-news/press-release-details/2017/Facebook-Reports-Fourth-Quarter-and-Full-Year-2016-Results/default.aspx

を vertex-cut 方式に変換することで，通信コストを削減すると共にロードバランスを最適に近づける．本稿では，提案手法を既存の分散グラフ処理フレームワークである PowerGraph [8] に組み込み，既存のグラフ分割手法を用いた場合との比較実験を行った．その結果，HDRF を用いた場合に比べ，PageRank の分析処理を最大 3.2 倍高速化できることを確認した．

本報告の構成は以下の通りである．2. 節で本稿の前提となる知識について概説する．3. 節で提案手法の詳細について説明し，4. 節にて提案手法の評価と分析を行う．5. 節で関連研究について述べ，6. 節で本報告をまとめ，今後の課題について述べる．

2. 前提知識

2.1 Modularity

提案手法では，切断エッジ数を削減するために Modularity [12] という指標を採用している．したがって，本節ではクラスタリング指標である Modularity について概説する．

Modularity はクラスタリング結果の良さを評価するための指標であり，クラスタ内のエッジが密で，かつクラスタ間のエッジが疎であるほど高い値を示すように設計されている．具体的には，実際のグラフデータにおけるクラスタ内エッジ数が，エッジをランダムに繋ぎ直してできるランダムグラフにおけるクラスタ内エッジ数より多いほど高い値を示す．グラフデータのエッジ集合を E，クラスタリングにより得られるクラスタの集合を $\mathbb{C}$，クラスタ i からクラスタ j へ接続されているエッジの集合を E_{ij} とした時，Modularity の値 Q は以下の式で定義される．

$$Q = \sum_{i \in \mathbb{C}} \left\{ \frac{|E_{ii}|}{2|E|} - \left(\frac{\sum_{j \in \mathbb{C}} |E_{ij}|}{2|E|} \right)^2 \right\}. \quad (1)$$

提案手法ではボトムアップ方式のクラスタリングを採用し，Modularity の値 Q が向上するように異なるクラスタの組をマージしていく．しかし，マージするクラスタの組を決定するために毎回式 (1) を計算するのは極めて非効率的である．したがって，提案手法ではマージするクラスタの組を決定する際に，Clauset らが導出した Modularity の変化量を求める指標 [5] を用いる．隣接するクラスタの組 i, j をマージした場合の Modularity の変化量 ΔQ_{ij} は以下のように定義される．

$$\Delta Q_{ij} = 2 \left\{ \frac{|E_{ij}|}{2|E|} - \left(\frac{\sum_{k \in \mathbb{C}} |E_{ik}|}{2|E|} \right) \left(\frac{\sum_{k \in \mathbb{C}} |E_{jk}|}{2|E|} \right) \right\}. \quad (2)$$

上式の ΔQ_{ij} が大きくなるようなクラスタの組 i, j をマージすることによって，グラフデータ全体の Modularity を向上させることができる．また，本稿では，上記の Modularity の変化量 ΔQ_{ij} だけでなく，脇田らが提案した以下の指標 [20] を用いた場合についても評価を行っている．

$$\Delta Q'_{ij} = min \left(\frac{|E_{ii}|}{|E_{jj}|}, \frac{|E_{jj}|}{|E_{ii}|} \right) \times \Delta Q_{ij}. \quad (3)$$

ここで，右辺の ΔQ_{ij} は式 (2) で示した Modularity の変化量である．一方，$min \left(\frac{|E_{ii}|}{|E_{jj}|}, \frac{|E_{jj}|}{|E_{ii}|} \right)$ はクラスタ i, j の内包エッジ数の比率であり，内包エッジ数の差が小さいほど大きな値を示す．式 (3) を用いることで，式 (2) を用いた場合よりも内包エッジ数が同程度のクラスタ同士がマージされるようになり，クラスタの内包エッジ数の極端な偏りを抑えることができる．

2.2 レプリケーションファクタとロードバランスファクタ

分散グラフ処理においては，edge-cut 方式のグラフ分割よりも vertex-cut 方式のグラフ分割を採用したほうが通信コストとロードバランスの両方において優れていると報告されている [2] [14]．したがって，提案手法では得られた edge-cut 方式のクラスタを vertex-cut 方式のクラスタに変換することで，さらなる品質の向上を実現している．本節では，vertex-cut 方式のグラフ分割における品質の評価指標について概説する．

vertex-cut 方式のグラフ分割の品質を評価するための指標として，レプリケーションファクタとロードバランスファクタがある．レプリケーションファクタ [8] は通信コストを評価するための指標であり，グラフデータの本来の頂点数に対し，実際の分析処理において計算機上に格納される頂点の複製数がどれぐらい多いかを表す．vertex-cut 方式のフレームワークでは，あるエッジを計算機に割り当てた時に，そのエッジの両端の頂点の複製を計算機上に格納する．同じ頂点と隣接するエッジを異なる計算機に割り当てると，同じ頂点の複製が異なる計算機上に格納されることになるが，このような複製間では，分析処理中にデータの整合性を保つための通信が発生する．したがって，グラフ分割によって生成される頂点の複製数が少ないほど通信コストを抑えることができる．グラフデータの頂点集合を V，頂点 $v \in V$ の複製を格納する計算機の集合を $A(v)$ とした時，レプリケーションファクタ RF は以下の式で定義される．

$$RF = \frac{1}{|V|} \sum_{v \in V} |A(v)|. \quad (4)$$

上式の RF の値が小さいほど，分析処理中の通信コストを抑えることができる．

一方，ロードバランスファクタは計算負荷の偏りを評価するための指標である．vertex-cut 方式のフレームワークでは，ロードバランスを評価するために以下のような式 [8] が用いられる．

$$\max_{m \in M} |E(m)| < \lambda \frac{|E|}{|M|}. \quad (5)$$

ここで，M は全計算機からなる集合，$E(m)$ は計算機 m に割り当てられているエッジ集合を表す．また，λ は 1 以上の小さな実数であり，グラフ分割によるタスク量の偏りをどの程度許容するかを表すパラメータである．すなわち式 (5) は，計算機に割り当てられるエッジ数の最大値を，計算機当たりの平均エッジ数の何倍まで許容するかということを表している．ここで，式 (5) から以下のような指標を導くことができる．

$$\lambda = \frac{|M|}{|E|} \max_{m \in M} |E(m)|. \quad (6)$$

本稿では，式 (6) の λ をロードバランスファクタと呼び，等粒度性を評価するための指標として用いる．

2.3 HDRF

本稿では，提案手法の edge-cut 方式から vertex-cut 方式への変換時に HDRF [14] を適用することで，単純なエッジ振り分けを行った場合よりも分析処理を高速化できることを示す．し

たがって，本節では最新のグラフ分割手法である HDRF について概説する．HDRF(High Degree (are) Replicated First) は，vertex-cut 方式の手法であり，グラフデータの読み込み時点でエッジの割り当て先を逐次決定するストリーム方式の手法である．HDRF はその名の通り，次数の高い頂点の複製を優先的に生成するよう設計されている．これは，実世界で多く見られる次数の分布確率がべき乗則に従うようなグラフデータ [7] では，次数の高い頂点（ハブ）を取り除くだけでも独立したクラスタを検出することができる [3] [6] という性質から着想を得ている．HDRF は，過去のエッジ割り当ての情報から通信コストに関するスコアとロードバランスに関するスコアを計算し，その合計値が最大となる計算機をエッジの割り当て先とする．具体的なスコアの定義は以下の通りである．エッジ (u, v) を読み込んだ時点での計算機 m に割り当てられたエッジの集合を $E(m)$ とする．また，頂点 u, v の複製が格納される計算機の集合をそれぞれ $A(u), A(v)$ とする．HDRF では，以下のように定義されるスコア S が最大となるような計算機 m をエッジ (u, v) の割り当て先とする．

$$
\begin{aligned}
S(u, v, m) &= S_{rep}(u, v, m) + S_{bal}(m) \\
S_{rep}(u, v, m) &= f(u, v, m) + f(v, u, m) \\
f(u, v, m) &= \begin{cases} 1 + \frac{d(v)}{d(u)+d(v)}, & if \quad m \in A(u) \\ 0, & otherwise \end{cases} \qquad (7) \\
S_{bal} &= \lambda \cdot \frac{maxsize - |E(m)|}{\epsilon + maxsize - minsize}
\end{aligned}
$$

ここで，$d(u), d(v)$ はそれぞれ，エッジ (u, v) が読み込まれた時点での頂点 u, v の隣接エッジ数であり，$maxsize$ と $minsize$ はそれぞれ計算機に割り当てられているエッジ数の最大値と最小値である．また，λ はロードバランスの重みを決定するためのパラメータであり，ϵ は小さな正の定数である．スコア S_{rep} は通信コストに関するスコアであり，頂点 u, v の複製がすでに格納されている計算機ほど高い値となる．すなわち，エッジ (u, v) を割り当てて新たに格納される複製が少ない計算機ほどスコアが高くなる．また，スコア f は，頂点 u, v のうち，次数の低い頂点を格納する計算機の方が高い値となる．これにより，次数の高い頂点の複製が優先的に格納されることになる．$d(u), d(v)$ は正確な次数ではないが，次数分布がべき乗則に従うようなグラフにおいては，ごく一部の次数の高い頂点がその他の多くの頂点と隣接しているために，多くの場合，エッジをランダムに読みこめば次数の高い頂点を上手に切断できると Petroni らは述べている [14]．スコア S_{bal} はロードバランスに関するスコアであり，割り当てられているエッジ数が少ない計算機ほど高い値となる．λ が 1 以下の場合，S_{bal} は 1.0 より大きな値となることはないため，頂点の複製数の削減が優先される．λ の値を大きくするほどロードバランスの向上が優先される．HDRF は最適なロードバランスを実現しつつ，ランダムな割り当て手法やその他の逐次割り当て手法よりも通信コストを削減することができる [14] が，グラフ構造全体を見る手法に比べると通信コストが高くなってしまう．

Algorithm 1 Modularity クラスタリングステップの流れ

Input: $G(V, E), k, a$
Output: $\mathbb{C}$

```
1: ℂ ← ∅
2: for each v ∈ V do
3:     c ← {v}
4:     ℂ ← ℂ + {v}
5: end for
6: ℂ_tmp ← ∅
7: while ℂ ≠ ℂ_tmp do
8:     ℂ_tmp ← ℂ
9:     for each c ∈ ℂ_tmp do
10:        ℕ ← {n | n ∈ neighbors(c) and size_constraint(c, n)}
11:        m = arg max_{n∈ℕ}{modularity_gain(c, n)}
12:        if modularity_gain(c, m) > 0 then
13:            c' ← merged_cluster(c, m)
14:            ℂ ← ℂ − {c, m} + {c'}
15:            if |ℂ| = a × k then
16:                break
17:            end if
18:        end if
19:    end for
20: end while
```

表 1　アルゴリズム 1 で使用する主な記号の定義

記号	意味
$G(V, E)$	グラフデータ．V は頂点集合，E はエッジ集合
k	最終的に得たいクラスタの数
a	終了条件を決めるパラメータ
$\mathbb{C}$	クラスタ集合
$neighbors(c)$	クラスタ c の隣接クラスタ集合
$size\_constraint(c, n)$	サイズ制約を満たすかどうかを判定する関数
$modularity\_gain(c, n)$	Modularity の変化量（式 (2) または式 (3)）
$merged\_cluster(c, n)$	クラスタ c, n をマージして得られるクラスタ

3. 提案手法

提案手法は，3 つのステップを経て計算機台数と同じ数である k 個のクラスタを導出する．1 つ目のステップは，Modularity に基づくクラスタリングにより切断エッジ数を削減する Modularity クラスタリングステップである．2 つ目のステップは，Modularity クラスタリングステップによって得られたクラスタを，等粒度性が高くなるように k 個のクラスタに割り当てていく等粒度マージステップである．3 つ目のステップは，edge-cut 方式のクラスタを vertex-cut 方式のクラスタに変換する vertex-cut 変換ステップである．各ステップの詳細について，それぞれ 3.1 節，3.2 節，3.3 節で述べる．

3.1 Modularity クラスタリングステップ

Modularity クラスタリングステップでは，2.1 節で述べた式 (2) または式 (3) を用いて，Modularity の値が向上するようにクラスタの組をマージしていく．このステップでは，最新の Modularity クラスタリング手法である塩川らの逐次集約手法 [16] を用いる．この手法は，Louvain 法 [1] と同等の品質でクラスタリングが可能であり，かつクラスタのマージを逐次行うことでマージ判定の際に参照するエッジ数を削減し，その他の既存手法よりも高速なクラスタリングを可能にする．

Algorithm 2 等粒度マージステップその 1 の流れ

Input: $\mathbb{C}, k$
Output: $\mathbb{S}$

1: $\mathbb{S} \leftarrow top\_k\_clusters(\mathbb{C}, k)$
2: $\mathbb{C} \leftarrow \mathbb{C} - \mathbb{S}$
3: $sort\_desc(\mathbb{C})$
4: **for each** $c \in \mathbb{C}$ **do**
5: $\quad s \leftarrow \arg\min_{s \in \mathbb{S}}\{|inner\_edges(s)|\}$
6: $\quad s' \leftarrow merged\_cluster(s, c)$
7: $\quad \mathbb{S} \leftarrow \mathbb{S} - \{s\} + \{s'\}$
8: **end for**

Modularity クラスタリングステップの処理の流れをアルゴリズム 1 に示す．また，主な記号の定義を表 1 に示す．このアルゴリズムは，入力としてグラフデータ $G(V, E)$ と最終的に得たいクラスタの数 k，アルゴリズムの終了条件を決めるための 1 以上のパラメータ a を受け取る．このアルゴリズムは，全ての頂点が異なるクラスタに属した状態から始まり (1 行目)，それ以上 Modularity の値が向上しなくなるか，あるいはクラスタの数が $a \times k$ になるまでクラスタのマージ処理 (2 行目〜14 行目) を行う．クラスタ c とマージするクラスタ m は，c の隣接クラスタ集合 N の中で，サイズ制約を満たし，かつ Modularity の変化量が最大となるものを選択する．ここで，サイズ制約とは，クラスタ c とクラスタ m をマージして生成されるクラスタの内包エッジ数が，平均エッジ数 $|E|/k$ より大きくなってはならないという制約のことである．この制約により，最終的なロードバランスファクタの悪化を防ぐことができる．

Modularity クラスタリングステップでは，最終的なクラスタの等粒度性を高めるために k 個よりも多いクラスタを導出するにとどめ，次のステップで k 個のクラスタへの割り当て処理を行う．

3.2 等粒度マージステップ

等粒度マージステップでは，Modularity クラスタリングステップで得られた $a \times k$ 個のクラスタを，最終的な等粒度性が高くなるように k 個のクラスタに割り当てていく．このステップでは，処理開始時点で内包エッジ数が多い上位 k 個のクラスタを種クラスタとする．そして，種クラスタの中で内包エッジ数が最少のクラスタと，その他のクラスタの中で内包エッジ数の多いクラスタとをマージしていく．こうすることで，ステップの終盤に内包エッジ数が少ないクラスタによる調整が可能となり，等粒度性を向上させることができる．本稿では，等粒度マージステップにおける処理の方法を 2 種類示す．1 つ目は，種クラスタとマージするクラスタを，内包エッジ数のみで決定する方法である．2 つ目は，種クラスタとマージするクラスタを，種クラスタの隣接クラスタから優先して選択する方法である．後者の方法は前者の方法に比べて，マージ候補のクラスタに制限があるため等粒度性が悪化しやすいが，隣接クラスタ同士をマージすることでより多くの切断エッジ数を削減することができる．

等粒度マージステップにおける 2 種類の処理の流れをそれぞれアルゴリズム 2 とアルゴリズム 3 に示す．また，主な記号の定義を表 2 に示す．等粒度マージステップは，入力として，Modularity クラスタリングステップの出力として得られたクラスタ集合 $\mathbb{C}$ と最終的に得たいクラスタの数 k を受け取る．いずれのアルゴリズムにおいても，最初に $\mathbb{C}$ 内の内包エッジ数の多いクラスタ上位 k 個を種クラスタ集合 $\mathbb{S}$ として取得する（1 行目）．以降の処理では，$\mathbb{S}$ 内の種クラスタと $\mathbb{C}$ 内のクラスタをマージしていく．まず，内包エッジ数のみを考慮したアルゴリズム 2 について説明する．このアルゴリズムでは，まずクラスタ集合 $\mathbb{C}$ を内包エッジ数で降順にソートする（3 行目）．そして，内包エッジ数の多いクラスタから順に，マージ時点で内包エッジ数が最少の種クラスタとマージを行っていく（4 行目〜8 行目）．次に，種クラスタと隣接するクラスタとのマージを優先するアルゴリズム 3 について説明する．このアルゴリズムは，大きく 1 つの処理に分けられる．1 つ目の処理は，種クラスタと隣接するクラスタとをマージする処理である（3 行目〜14 行目）．この処理では，隣接クラスタを持つ種クラスタの中で内包エッジ数が最少の種クラスタからマージを行ってい

Algorithm 3 等粒度マージステップその 2 の流れ

Input: $\mathbb{C}, k$
Output: $\mathbb{S}$

1: $\mathbb{S} \leftarrow top\_k\_clusters(\mathbb{C}, k)$
2: $\mathbb{C} \leftarrow \mathbb{C} - \mathbb{S}$
3: **while** $\mathbb{C} \neq \emptyset$ **do**
4: $\quad \mathbb{S}_{nei} \leftarrow \{s \mid s \in \mathbb{S}$ **and** $neighbors(s) \cap \mathbb{C} \neq \emptyset\}$
5: $\quad$ **if** $\mathbb{S}_{nei} = \emptyset$ **then**
6: $\quad\quad$ **break**
7: $\quad$ **end if**
8: $\quad s \leftarrow \arg\min_{s \in \mathbb{S}_{nei}}\{|inner\_edges(s)|\}$
9: $\quad \mathbb{N} \leftarrow neighbors(s) \cap \mathbb{C}$
10: $\quad m \leftarrow \arg\max_{n \in \mathbb{N}}\{|inner\_edges(n)|\}$
11: $\quad s' \leftarrow merged\_cluster(s, m)$
12: $\quad \mathbb{R} \leftarrow \mathbb{R} - \{s\} + \{s'\}$
13: $\quad \mathbb{C} \leftarrow \mathbb{C} - \{m\}$
14: **end while**
15: **for each** $c \in \mathbb{C}$ **do**
16: $\quad \mathbb{M} \leftarrow connected\_clusters(c)$
17: $\quad$ **for each** $m \in M$ **do**
18: $\quad\quad c \leftarrow merged\_cluster(c, m)$
19: $\quad$ **end for**
20: $\quad s \leftarrow \arg\min_{s \in \mathbb{S}}\{|inner\_edges(s)|\}$
21: $\quad s' \leftarrow merged\_cluster(s, c)$
22: $\quad \mathbb{S} \leftarrow \mathbb{S} - \{s\} + \{s'\}$
23: $\quad \mathbb{C} \leftarrow \mathbb{C} - \mathbb{M}$
24: **end for**

表 2　アルゴリズム 2，3 で使用する主な記号の定義

記号	意味
$\mathbb{C}$	入力クラスタ集合
k	最終的に得たいクラスタの数
$\mathbb{S}$	種クラスタの集合
$top\_k\_clusters(\mathbb{C}, k)$	$\mathbb{C}$ 内の内包エッジ数の多いクラスタ上位 k 個
$sort\_desc(\mathbb{C})$	クラスタ集合を内包エッジ数により降順にソート
$inner\_edges(c)$	クラスタ c の内包エッジ集合
$merged\_cluster(s, c)$	クラスタ s, c をマージして得られるクラスタ
$neighbors(c)$	クラスタ c の隣接クラスタ集合
$connected\_clusters(c)$	クラスタ c と連結する全クラスタの集合

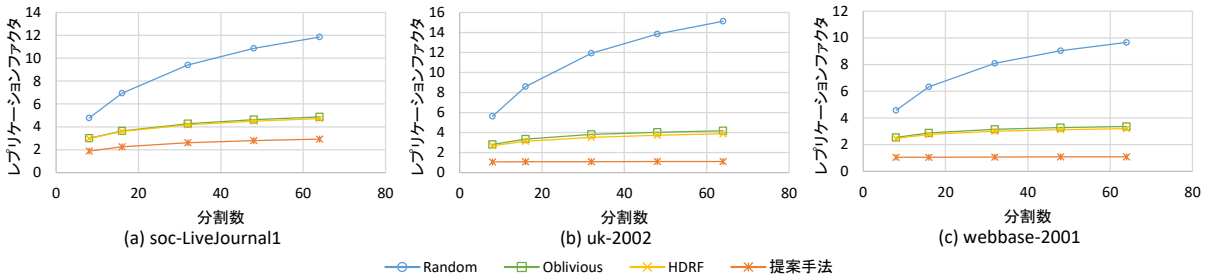

図 1　分割数を変化させた場合のレプリケーションファクタ

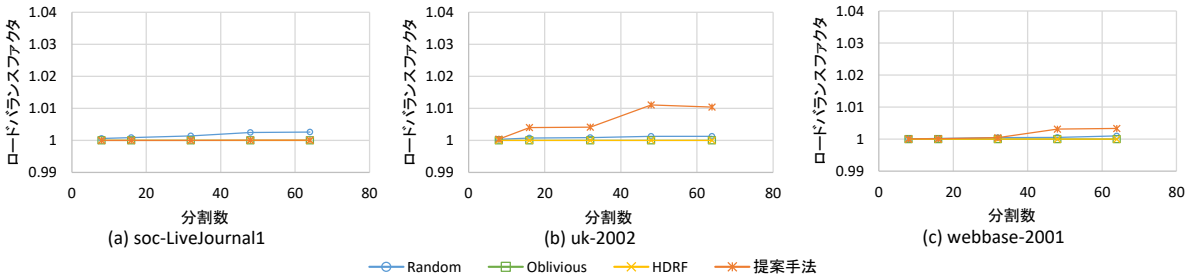

図 2　分割数を変化させた場合のロードバランスファクタ

く．2 つ目の処理は，種クラスタと，種クラスタからエッジを通じて辿りつけない非連結なクラスタとをマージする処理である（15 行目〜24 行目）．実世界で見られるグラフデータの多くは，最大の連結成分以外の連結成分に属する頂点数は少ないことから，アルゴリズム 3 では，これらの連結成分をすべて 1 つにまとめ，内包エッジ数が最少の種クラスタにマージするという処理を行っている．これにより，切断エッジ数のさらなる削減が可能である．

3.3　vertex-cut 変換ステップ

vertex-cut 変換ステップでは，前述のステップにより得られた edge-cut 方式のクラスタを vertex-cut 方式のクラスタに変換する．vertex-cut 方式への変換を行うためには，切断エッジをいずれかのクラスタに一意に割り当てる必要がある．本稿では，切断エッジの割り当て先を決定する方法を 2 種類示す．1 つ目の方法は，切断エッジの両端のクラスタのうち，内包エッジ数が少ない方に割り当てるという方法である．この単純な振り分け手法は，等粒度性を高めつつ 1 つの切断エッジによって生成される頂点の複製をたかだか 1 つに制限することができる．しかし，格納される頂点の複製数は考慮していないことから，通信コストの削減に関して改善の余地がある．2 つ目の方法は，切断エッジの割り当て先を 2.3 節で述べた HDRF [14] を用いて決定する方法である．HDRF は格納される頂点の複製数が少なくなるようにエッジを割り当てるため，通信コストをより効果的に削減できる．また，HDRF はロードバランスが向上するようにも設計されており，等粒度マージステップ終了時のクラスタの等粒度性も高いことから，最終的に得られるクラスタの等粒度性も十分高くなることが考えられる．

4.　評 価 実 験

提案手法の性能を評価するために，提案手法を既存の分散グラフ処理フレームワークである PowerGraph [8] に組み込み評価実験を行った．本実験で用いたデータセットを表 3 に示す．いずれも，ソーシャルグラフや Web グラフなどの実データをもとに形成されたグラフデータである．提案手法のパラメータ a は，多くの場合グラフデータの頂点数およびエッジ数が多いほど大きな値とする必要がある．本実験では，分割数を k とした時に，soc-LiveJournal では $a \times k = 8,000$, uk-2002 では $a \times k = 16,000$，webbase-2001 では $a \times k = 160,000$ となるように a を設定した．

表 3　比較実験で用いたデータセット

データ名	$\|V\|$	$\|E\|$	Modularity
soc-LiveJournal1 [22]	4,847,571	68,993,773	0.721
uk-2002 [23]	18,520,486	298,113,762	0.986
webbase-2001 [23]	118,142,155	1,019,903,190	0.976

本実験では，実験環境として Amazon EC2 を使用した．使用したインスタンスは r3.2xlarge の linux インスタンスである．CPU は Intel(R) Xeon(R) CPU E5-2670 v2 であり，クロック数は 2.50GHz，コア数は 4 である．メモリは 60GB である．また，インスタンス間のデータ伝送速度はおよそ 1.03Gbps であり，hdparm を-t オプションをつけ実行することで得られたディスク読み込み速度はおよそ 103MB/sec であった．提案手法の実装には C++を用いた．コンパイルに使用した g++のバージョンは 4.8.1 であり，最適化オプションとして-O3 を使用した．また，PowerGraph も C++で実装されており，上記と同様の条件でコンパイルを行った．

本実験では，2.2 節で述べた評価指標と実際に分析処理を実行した場合の分析処理時間による評価を行う．まず，4.1 節で既存手法との比較結果を示す．次に，4.2 節で各ステップの処理方法を変えた複数の提案手法の比較結果を示す．

4.1　既存手法との比較

本節では，既存手法と提案手法との比較結果を示す．既存手法として，Random，Oblivious [8]，HDRF [14] の 3 つの手

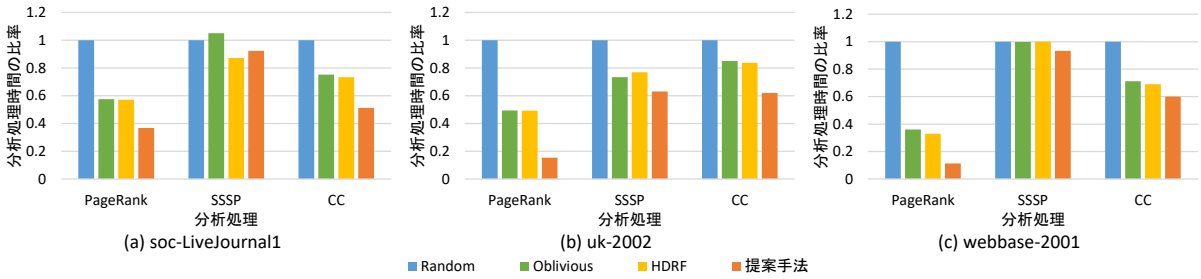

図 3　計算機台数を 64 台とした場合の Random に対する分析処理時間の比率

法を用いた．Random は，エッジの割り当て先をハッシュ値を用いてランダムに決定する最も単純な手法である．Oblivious は，エッジ割り当てにより格納される頂点の複製数が最少であり，かつ割り当てられているエッジ数が最少の計算機にエッジを割り当てる手法である．Oblivious に次数の高い頂点の複製を優先的に格納するように変更を加えた手法が HDRF である．HDRF のパラメータ λ は文献 [14] に従い 1 とした．既存手法はいずれも PowerGraph に実装されているプログラムを使用した．提案手法に関しては，既存手法との比較を明瞭にするために，本節では Modularity クラスタリングステップにおいて等粒度性を考慮した Modularity 変化量を用い，等粒度マージステップで内包エッジ数だけを考慮してクラスタのマージを行い，vertex-cut 変換ステップにおいて HDRF を用いる手法についての結果のみ示す．

4.1.1　グラフ分割指標による評価

図 1 に分割数を変化させた場合の各手法のレプリケーションファクタを示す．図 1 から，すべてのグラフデータおよび分割数において，既存手法と比べ提案手法の方がレプリケーションファクタを削減できていることが分かる．本実験では，HDRF に対して，レプリケーションファクタを soc-LiveJournal1 では最大 38.0%，uk-2002 では最大 71.5%，webbase-2001 では最大 65.9%削減することに成功した．表 3 と合わせて，Modularity の値が大きいグラフデータほどレプリケーションファクタの削減率が大きいことが分かる．これは，Modularity の値が大きいグラフデータほど，Modularity クラスタリングステップでより多くの切断エッジを削減することができたためであると考えられる．

また，図 2 に分割数を変化させた場合の各手法のロードバランスファクタを示す．図 2 から，uk-2002, webbase-2001 においては Random と比べロードバランスが悪化しているものの，ロードバランスファクタの悪化を 1%ほどに抑えられていることが分かる．

4.1.2　分析処理時間による評価

本実験では，PowerGraph を用いて実際にグラフ分析アルゴリズムを実行し，分析処理に要した時間を計測した．本実験で用いたグラフ分析アルゴリズムは以下の 3 つである．

1)　PageRank [13]: ウェブページの重要度を計算するために考えられた分析処理

2)　SSSP (single-source shortest path): ある 1 つの頂点から他の全ての頂点への最短距離を求める分析処理

表 4　webbase-2001 に対する計算機あたりの平均通信バイト量

	PageRank	SSSP	CC
HDRF	16.2GB	688.2MB	1207.5MB
提案手法	1.07GB	131.5MB	176.0MB

3)　CC (Connected Component): グラフデータ内から頂点が連結された部分グラフを抽出する分析処理．

図 3 に結果を示す．縦軸は Random を用いた場合の分析処理時間に対する比率である．図 3 から，ほぼすべての分析処理およびグラフデータにおいて，既存手法と比べ提案手法の方が分析処理時間を高速化できていることが分かる．特に，PageRank においては，既存手法に対しレプリケーションファクタをより削減できているグラフデータほど高速化の効果が大きいことが分かる．しかし，soc-LiveJournal1 を用いて SSSP を実行した場合は，HDRF を用いた場合と比べて分析処理時間が長くなっている．この結果に関しては，HDRF を用いた場合の分析処理時間が 3.4 秒，提案手法を用いた場合の分析処理時間が 3.6 秒と非常に短いため，分析処理中のその他のオーバヘッドの影響が大きくなり，提案手法による高速化の効果がでなかったためであると考えられる．

分析処理によって提案手法の高速化の効果が異なるのは，各分析処理の通信量が異なるためであると考えられる．グラフデータとして webbase-2001 を用いた場合の各分析処理中の計算機あたりの通信バイト量を表 4 に示す．表 4 から，分析処理によって分析処理中の通信バイト量が大きく異なることが分かる．また，図 3 と合わせて，通信バイト量の差が大きい分析処理ほど提案手法の高速化の効果が大きいことが分かる．soc-LiveJournal1 および uk-2002 を用いた場合においても，同様の傾向の結果となった．本実験では，HDRF に対して，PageRank の分析処理を最大 3.2 倍，SSSP の分析処理を最大 1.2 倍，CC の分析処理を最大 1.3 倍高速化することに成功した．

4.2　各ステップの性能評価

本稿では，提案手法の各ステップにおける処理の方法を 2 種類ずつ示した．本稿で示した各ステップの処理方法をまとめると，以下の通りである．

- Modularity クラスタリングステップ：通常の Modularity 変化量を用いるか (**normal**)，等粒度性を考慮した Modularity 変化量を用いるか (**balanced**)
- 等粒度マージステップ：クラスタの内包エッジ数のみを

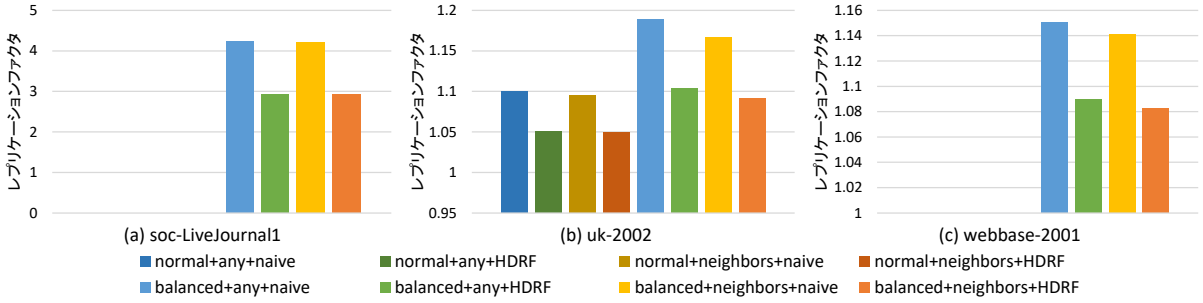

図 4　各提案手法のレプリケーションファクタ ((a)(c) の normal は計測不可)

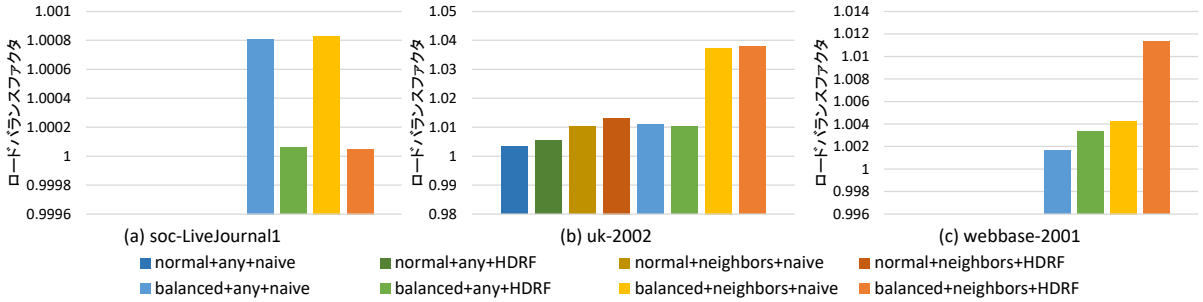

図 5　各提案手法のロードバランスファクタ ((a)(c) の normal は計測不可)

考慮してマージするか (**any**)，隣接クラスタのマージを優先するか (**neighbors**)

- vertex-cut 変換ステップ：単純な振り分け手法を用いるか (**naive**)，HDRF を用いるか (**HDRF**)

以降，説明の簡略化のために，各ステップでどの処理方法を用いるかを略称を用いて記述することとする．例えば，Modularity クラスタリングステップでは等粒度性を考慮した Moduarity 変化量を用い，等粒度マージステップではクラスタの内包エッジ数のみを考慮し，vertex-cut 変換ステップでは HDRF を使用する提案手法は，balanced+any+HDRF と記述する．

各ステップの処理方法を変えた複数の提案手法におけるレプリケーションファクタおよびロードバランスファクタを，それぞれ図 4，図 5 に示す．分割数は 64 である．soc-LiveJournal1 および webbase-2001 の図における Modularity クラスタリングステップで normal を用いた場合の結果に関しては，長時間経過してもグラフ分割処理が終了しなかったために結果を記載していない．この原因として，Modularity クラスタリングステップにおけるサイズ制約の影響が考えられる．Modularity クラスタリングステップでは，内包エッジ数が計算機あたりの平均エッジ数を超過するようなクラスタのマージは行わないという制約を設けている．soc-LiveJournal1 および webbase-2001 では，normal を用いた場合は内包エッジ数の多いクラスタが生成されてしまい，サイズ制約を満たすクラスタの組を見つけるのに膨大な時間を要するのだと考えられる．図 4 から，vertex-cut 変換ステップにおいて naive を用いた手法に比べ，HDRF を用いた手法の方がレプリケーションファクタを削減できていることが分かる．また，naive と HDRF ほどの差はないものの，等粒度マージステップにおいて any を用いた手法に比べ，neighbors を用いた手法の方がレプリケーションファクタを削減できていることが分かる．一方，図 5 から，any を用いた手法に比べて neighbors を用いた手法の方がロードバランスファクタが増大していることが分かる．Modularity クラスタリングステップの比較では，balanced を用いる手法よりも normal を用いる手法の方がレプリケーションファクタを削減できていることが分かる．そして，ロードバランスファクタに関しても，normal を用いる手法の方がロードバランスの悪化を抑制できていることが分かる．これは，等粒度性を考慮しない Moduarity 変化量を用いる場合でも，サイズ制約を設けることによって内包エッジ数の偏りを抑制できているためと考えられる．

図 6 に，各ステップの処理方法を変えた複数の提案手法を用いた場合の分析処理時間を示す．計算機台数は 64 台であり，用いた分析処理は PageRank である．図 6 から，vertex-cut 変換ステップにおいて，naive を用いる手法に比べ HDRF を用いる手法の方が分析処理を高速化できていることが分かる．これは，HDRF を用いた手法の方が通信コストを削減できるからだと考えられる．一方，レプリケーションファクタが同程度となっている手法同士を比較すると，ロードバランスファクタが増大するような場合であっても分析処理時間が早くなっている場合があるのが分かる．この原因として，頂点単位で更新処理を行うという分散グラフ処理における効率化の影響が考えらえる．分散グラフ処理では，頂点単位で更新処理を並列に実行し，値が収束した頂点から更新処理を終えることで効率化を図っている．そのため，分析処理途中に各計算機上で更新処理を行う頂点の数にばらつきが生じる場合がある．提案手法ではいずれの組み合わせにおいてもロードバランスの悪化を抑制できていることから，更新処理を終了した頂点のばらつきによる計算負荷の偏りの影響が大きくなるのだと考えられる．

5. 関 連 研 究

著名なグラフ分割ライブラリとして METIS [9] が挙げられ

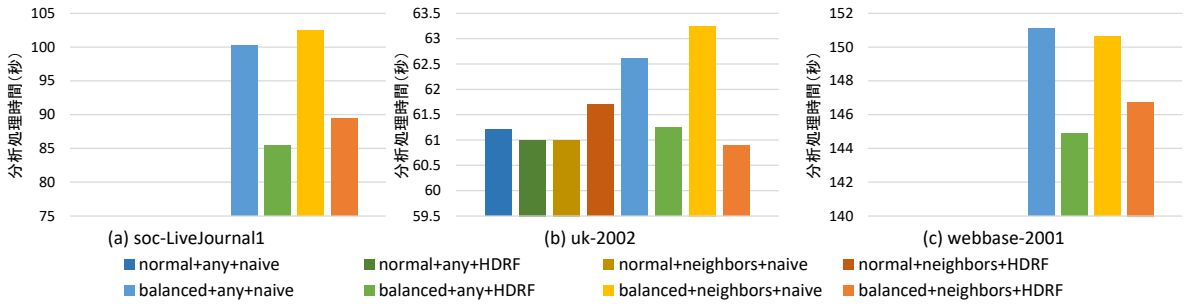

図 6　各提案手法の分析処理時間 ((a)(c) の normal は計測不可)

る．METIS は，グラフデータを圧縮するフェーズ，圧縮されたグラフデータを分割するフェーズ，分割の品質を向上させながらグラフデータを元のサイズに戻していくフェーズの 3 つのフェーズによってグラフデータを分割する．METIS は品質の高いグラフ分割が可能であるものの，大規模なグラフデータの分割には膨大な時間がかかるため，分散グラフ処理では用いられていない．

分散グラフ処理では，Oblivious [8] や HDRF [14] のようなストリーミンググラフ分割手法 [17] が多く用いられている．Fennel [19] は edge-cut 方式の手法であり，割り当てにより切断されるエッジ数が少なく，かつ内包する頂点数が少ないような部分グラフに頂点を割り当てる．Ginger [4] は，次数の高い頂点と隣接するエッジは vertex-cut 方式により別の部分グラフに割り当て，次数の低い頂点は edge-cut 方式の Fennel により割り当て先を決定することで，HDRF と同様に次数の高い頂点の複製を優先的に格納する．

6. おわりに

本稿では，通信コストの削減と計算負荷の均衡化とをバランスよく両立し，分散グラフ処理における分析処理を高速化できるグラフ分割手法を提案した．提案手法は，Modularity に基づくクラスタリングを拡張することで，切断エッジ数が少なく，かつ等粒度性の高いクラスタを導出することを可能とする．また，得られた edge-cut 方式のクラスタを vertex-cut 方式のクラスタに変換することでさらなる品質の向上を実現する．本稿では，提案手法を既存の分散グラフ処理フレームワークである PowerGraph に組み込み評価実験を行った．実験の結果，HDRF のみを利用した場合に比べ，PageRank の分析処理を最大 3.2 倍高速化することに成功した．今後の課題として，グラフデータの構造によって，各ステップにおける最適な処理方法を自動で決定する方法についての検討が考えられる．

文　献

[1] V. D. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre, “Fast unfolding of communities in large networks,” Journal of Statistical Mechanics: Theory and Experiment, 2008.
[2] F. Bourse, M. Lelarge and Milan Vojnovic, “Balanced graph edge partition,” Proceedings of KDD, 2014.
[3] D. S. Callaway, M. E. Newman, S. H. Strogatz and D. J. Watts, “Network robustness and fragility: Percolation on random graphs.”, Phys. Rev. Lett., 2000.
[4] Y. C. R. Chen, J. Shi and H. Chen, ”Powerlyra:Differentiated graph computation and partitioning on skewed graphs.”, Proceedings of SIGOPS ,2015.
[5] A. Clauset, M. E. J. Newman and C. Moore, “Finding community structure in very large networks,” Phys. Rev. E, 2004.
[6] R. Cohen, K. Erez, D. Ben-Avraham and S. Havlin, “Breakdown of the internet under intentional attack.”, Phys. Rev. Lett., 2001.
[7] M. Faloutsos, P. Faloutsos and Christos Faloutsos, “On power-law relationships of the Internet topology,” Proceedings of SIGCOMM, 1999.
[8] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson and C. Guestrin, “PowerGraph: distributed graph-parallel computation on natural graphs,” Proceedings of OSDI, 2012.
[9] G. Karypis and V. Kumar, “Multilevel k-way partitioning scheme for irregular graphs,” Parallel and Distributed Computing, 1998.
[10] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola and J. M. Hellerstein, ”Distributed GraphLab: a framework for machine learning and data mining in the cloud,” PVLDB, 2012.
[11] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser and G. Czajkowski, “Pregel: a system for large-scale graph processing,” Proceedings of SIGMOD, 2010.
[12] M. E. J. Newman and M. Girvan. “Finding and evaluating community structure in networks,” Phys. Rev. E, 2004.
[13] L. Page, S. Brin, R. Motwani and T. Winograd, “The pagerank citation ranking: Bringing order to the web,” Technical Report, 1999.
[14] F. Petroni, Leonardo Querzoni, K. Daudjee, S. Kamali and G. Iacoboni. “HDRF: Stream-Based Partitioning for Power-Law Graphs,” Proceedings of CIKM, 2015.
[15] S. Salihoglu and J. Widom, “GPS: a graph processing system.”, Proceedings of SSDBM, 2013.
[16] H. Shiokawa, Y. Fujiwara, and M. Onizuka, “Fast algorithm for modularity-based graph clustering,” Proceedings of AAAI, 2013.
[17] I. Stanton and G. Kliot, “Streaming graph partitioning for large distributed graphs,” Proceedings of KDD, 2012.
[18] S. Suri and S. Vassilvitskii, “Counting triangles and the curse of the last reducer,” Proceedings of WWW, 2011.
[19] C. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic, “Fennel: Streaming graph partitioning for massive scale graphs,” Proceedings of ACM, 2014.
[20] K. Wakita and T. Tsurumi, “Finding community structure in mega-scale social networks,” Proceedings of WWW, 2007.
[21] R. S. Xin, J. E. Gonzalez, M. J. Franklin and I. Stoica, “GraphX: a resilient distributed graph system on Spark,” Proceeding of GRADES, 2013.
[22] Stanford Large Network Dataset Collection. http://snap.stanford.edu/data/
[23] Laboratory for Web Algorithmics. http://law.di.unimi.it.