Vol.2020-AL-177 No.3
2020/3/16

同型性の利用によるデータ研磨アルゴリズムの高速化手法

宇野 毅明[1,a)]

概要：データ研磨は, データのゆらぎを除去することで中小規模の構造を明確化し, マイニングアルゴリズムの効率や精度を高める手法である. 例えば, ネットワーククラスタリングの場合, グラフの密な部分をクリークにし, 疎な部分を独立集合とすることで, クラスタ構造を明確にする. 通常のクラスタリングが, 比較的大きなクラスタを見つけるのが上手であるのに対して, データ研磨によるクラスタリングは, 小さくてまとまりの良いクラスタを網羅的に, かつ独立性高く, 適切な個数で見つけることができる. データ研磨は, 大規模な実データでも数十の反復で収束に至ることがほとんどであり, 高速性が期待できる一方で, グラフが密になると計算時間が長大になってしまうという弱点も持っている. 本稿では, 密なグラフに対するデータ研磨アルゴリズムの速度向上を目指し, 同型性を用いた高速化手法を提案する. グラフの中から隣接頂点集合が同じである頂点の対を見つけ, それを縮約することにより高速化を行う. 計算実験の結果により, ランダムに生成した人工データ, 実データの両方で高速化がなされることを示す.

キーワード：データ解析，クリーク列挙，クラスタリング, 同型性, 高速化

Speeding up Data Polishing Algorithm by the use of Isomorphism

Abstract: Data polishing is a methodology to increase the efficiency and accuracy of data mining algorithms by removing the ambiguities of the data to clarify the small to middle size structures of the data. For example, data polishing clarifies the cluster structures in the given network data by changing a dense subgraph to a clique, and a sparse subgraph to an empty subgraph on the other hand. Clustering algorithms with the use of data polishing can find small concentrated clusters, relatively more completely, with keeping the independency of each cluster from the others so that we get proper number of clusters, while ordinary clustering algorithms are good at finding few relatively big clusters. Data polishig algorithm usually terminates in few iterations such as 10 time 100. On the other hand, it may take long time when the input similarity graph is relatively dense. In this paper, we aim to speed up data polishing algorithm for dense graphs, and propose a new algorithm that utilizes the automorphisms in the data. It shrinks the vertices into one that have automorphism, so that the number of vertices in the graph will be decreased. Computational experiments show that the new algorithm reduces the computation time drastically for real world data and data generated by a generation model.

Keywords: data analysis, clique enumeration, clustering, ishomorphism, speedup

1. Introduction

クラスタリングは, 教師無し学習における基本的かつ中心的な手法である. 科学分野から産業・商業応用まで幅広く使われており, 理論面の研究も進んでいる. クラスタリングは, データを分けることであるが, 同時に似たものが集まったグループを見つけることにもある. 両者は異なるものであるが, 同時に混在した目的として説明されることが多く, これまでの研究なり産業応用なりで, 両者の違いを区別してモデル化なりデータ解析の方針なりを説明している文献は少ないように見える.

一般的にだが, 自然科学や商業での顧客分析など, データの大きな構造が知りたいときには, 「分ける」目的が主体となることが多いように思われる. また, 自動推薦や広告など, Web サービス分野や商業分野での顧客, ユーザの分類には, ユーザがなんらかのクラスタに所属していないと属性付与ができないため, 「分ける」アプローチが重要と

[1] 国立情報学研究所
2-1-2, Hitotsubashi, Chiyoda, Tokyo 101–8430, Japan
[a)] uno@nii.ac.jp

なる. 一方で, 新聞記事や SNS のクラスタからトピックを見つける, 顧客グループやコミュニティの特徴を捉えたいような場合は, 似た記事, 似た顧客だけが集まっていたほうが共通性が浮き上がりやすいため, 「似たものが集まったグループ」のアプローチが重要となる. 同じマーケッティングでも, 担当者が顧客を深く理解したい, 新商品開発や店舗レイアウト変更のために, 顧客の典型的な嗜好や動きが知りたい, というときなどに力を発揮する. 違う見方をすれば, 人間がクラスタに関して知識を, 自らが考えることによって得たいと思う場合には, 似たものが集まったグループを用いるのが適切であり, コンピュータによる自動化で何かを行いたい場合には, 分けるアプローチのほうが適していることが多い, という見方もできる. もちろん, これはすべてではなく, 特徴の自動抽出をしたいときなどは似たものが集まったグループ型が適切となる.

既存のクラスタリングアルゴリズムの研究では, 多くが「分ける」問題を扱っている. 著名なアルゴリズム, k-means[5] や Girvan-Newman[1], グラフカット [3] などもこのタイプである. 一方, 似たものが集まったグループを見つけ, クラスタに所属しない要素があっても良い, とするタイプのクラスタリングは, むしろマイニングアルゴリズムとして研究されていることが多く, 例えばコミュニティマイニング, ランダムウォークを用いたコミュニティ発見, 列挙アルゴリズムを用いたパターンマイニング的アルゴリズムがその代表である. 「分ける」問題に対するアルゴリズムは, 少数のクラスタに分ける場合においては, 比較的質の高いクラスタが得られることが多く, 科学分野から産業応用まで盛んに用いられている. 一方で「似たもののグループ」を見つけるものに対しては, アルゴリズムの研究は多いものの, 実用上満足のいく解が得られていないためか, 利用の範囲は非常に限定的である. 実用上障害となる点は, 解の再現性がない, 結果が安定しない, 網羅性がない, 解が多すぎて扱えない, などである.

筆者らは「似たもののグループ」を見つける問題に対して, データ研磨というアルゴリズムを提案し, 安定性, 再現性, 網羅性, 解の数に対する問題を一気に解決した [6], [7], [8], [9]. データ研磨は質の面でも高いクラスタを出力するため, 新聞記事や SNS のトピック分類や顧客自動分類のための良い特徴を見つけるためにも使われている [2], [4]. 新聞記事のクラスタを直接観察してみると, 既存手法のクラスタリングが, 粒度も中身もばらばらのクラスタを見つけているのに対し, データ研磨のアルゴリズムは中身が非常にそろった, 適切な大きさのクラスタを見つけていることがわかる. 一般にこのようなクラスタは既存手法が求めるクラスタよりもサイズが小さく数が多いため, このようなクラスタをマイクロクラスタと呼ぶ. また, グラフクラスタリングのために設計したデータ研磨アルゴリズムをグラフ研磨とよぶ.

一方で, グラフ研磨は類似度グラフを扱うように設計されているため, 類似度グラフが密である場合に計算時間が長くなるという弱点を持つ. 一般に巨大なデータは疎であることが多く, また, マイクロクラスタを見つけたいデータは, 要素間の類似度が小さく, クラスタ数が多くなるようなデータであるので, 速度の問題は頻発しない. しかし, データの一部が密であり, そこに対する計算時間が増大してしまう場合は十分にありうる. そこで本稿では, グラフ研磨のアルゴリズムの結果に影響を与えずに計算時間を高速化する手法を提案する. 大規模な類似度グラフにグラフ研磨を適用すると, グラフの中で互いに同型であるような頂点が増えてくる. 提案手法は, この同型な頂点を縮約することで, 計算時間を短縮するものである.

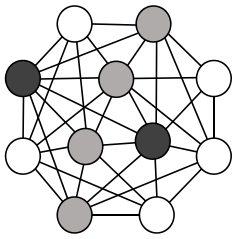

図 1 ある程度大きな密構造 (擬クリーク) に含まれる頂点の組には多くの頂点が共通して隣接する

グラフ研磨はグラフから不明瞭な部分を解消し, すべてのクラスタが極大クリークに対応するようにグラフを変形していく. そのため, 最終的に, 1 つのクラスタに含まれる頂点はほぼすべてが同型な頂点となる. 通常, ノイズが多い, 整ってない入力に対しては, データの圧縮など同型性を利用したものは効率が悪いことが多いが, グラフ研磨の上記のメカニズムにより, 反復が進むにつ入れて同型な頂点が増えて, 計算が高速化されると期待される. 本稿では, 実データとランダムデータで, この効果がどの程度であるかを検証する.

2. 記法

グラフの**クリーク**は，そのグラフの頂点部分集合で，全ての頂点間に枝がある物である．クリークは通常部分グラフとして定義されることが多いが，ここでは頂点集合で定義していることを注意しておく．他のクリークに含まれないクリークを**極大クリーク**という．

G の頂点 v に対し，v と枝で結ばれている頂点 u は，v に**隣接する**といい，そのような u を v の**近傍**という．v の近傍の集合を $N(v)$ と表記する．v の**次数** $d(v)$ は，v に隣接する頂点の数，つまり $|N(v)|$ である．$N[v]$ は $N(v) \cup \{v\}$ のことであり，**閉近傍**という．頂点 w が u と v の両方に隣接する頂点を**共通近傍**とよぶ.

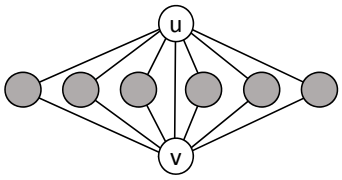

図 2 2つの頂点は6つの共通近傍を持っている

3. グラフクラスタリングに対するデータ研磨アルゴリズム

データ研磨は, データの不明瞭な部分を修正して明確にすることにより, データが内包する局所的な部分構造を明らかにするデータ解析手法である.

グラフクラスタリングでは，通常，局所的に枝が密である部分をグループ（クラスタ）であると考える．しかし，多くの実データでは密な部分の境界が曖昧であり，取りようによっていくらでも類似するグループ構造が見つかってしまう．逆に，全てのグループがクリークになっていて，グループに含まれない部分は疎になっているのであれば，境界がはっきりしており，明確に構造を発見できる．つまり，密度の高い部分はクリークに，そうでない部分には枝がない，というグラフが，グループ構造が明確になっているグラフと考えることができる.

このような「明確」なグラフは，同じクラスタに含まれる頂点の間には必ず枝があり，同じクラスタに含まれない頂点の間には枝がないようなグラフ，と考えることができる．そこで, グラフ G の2つの頂点 v と u が同じクラスタに含まれそうであれば枝があり, そうでないならば枝がないようなグラフ G' を作ることにより, クラスタ構造を明確化しようというのが, データ研磨の狙いである. v と u が同一のクラスタに含まれるかどうかは, 以下の仮説によって推測する.

近傍類似性仮説 v と u が同一のクラスタに含まれるなら, v の閉近傍の集合 $N[v]$ と u の閉近傍の集合 $N[u]$ は類似している. (類似度が θ 以上である)

u と v が同一のクラスタに含まれるならば, u と v はそのクラスタに対応する密な部分グラフに含まれるので, その密な部分グラフの頂点は u と v 両方との間に枝を持つ可能性が高く, 共通隣人の数が増える (図 1). すると, u の閉近傍集合と v の閉近傍集合の類似度が高くなる, ということである. グラフのすべての頂点対 v と u に対して, その閉近傍集合の類似度を $\mathrm{sim}(u,v)$ で表記する. sim は, 例えば閉近傍集合の共通部分 $N[u]\cap N[v]$ や Jaccard 係数 $\frac{N[u]\cap N[v]}{N[u]\cup N[v]}$ で定義する. ここで行うことは, u と v が同一のクラスタに含まれそうであれば枝が張られているグラフ G' を, 近傍類似性仮説から, $\mathrm{sim}(u,v)\geq\theta$ であれば u,v 間に枝があるグラフで定義することである. 2つの頂点が多くの共通近傍を持つ場合を, 図 2 に示した.

グラフ研磨は, この操作を反復的に行う, つまり G' にも適用してさらに新しいグラフを得て, そのグラフにも適用し, ということを続けて, グラフを明確化する. この反復は, 最終的にグラフが変化しなくなるまで, あるいは一定数の反復を行っても変化がなくならず, 収束の見込みがなくなるまで続ける. このデータ研磨アルゴリズム, 特にグラフ研磨と呼ぶ, は以下のように記述できる.

ALGORITHM GraphPolishing ($G=(V,E)$, sim(), θ, τ)
1. $G' := (V, \{(u,v) \mid \mathrm{sim}(u,v)\geq\theta\})$
2. **if** $G'=G$ or $i=\tau$ **then output** G; **exit**
3. $G := G'$
4. $i := i+1$; **go to** 1

著者らの計算実験 [6], [9] で, データ研磨は大きさが小さく類似性が高いものが集まったクラスタを適度な量見つける目的には非常に高い効果が示されている.

4. グラフ研磨アルゴリズムの高速化

前節で示したグラフ研磨アルゴリズムは, 入力グラフが疎, 特に次数分布がべき乗則に従っていれば, その反復の計算時間はグラフの枝数の線形になることが示されている [6], [9]. しかし, グラフが密になると, 1 反復の計算時間は線形より長くなり, 長大な計算時間を必要とするケースも出てくる. 本節では, 密な場合の計算時間を短縮する新しいアルゴリズムを提案する.

今, 頂点 u と u' が同型で, $N[u]=N[u']$ が成り立つとしよう. このとき, 任意の頂点 $v\neq u,u'$ に対して,

$$\mathrm{sim}(u,v)\geq\theta \;\leftrightarrow\; \mathrm{sim}(u',v)\geq\theta$$

が成り立つ. つまり, G' において, u と v が隣接していれば, またその時に限り, u' と v は隣接する. つまり, G' においても, $N[u]=N[u']$ が成り立つ. つまり, グラフ研磨の反復を繰り返す中で, 一度でも $N[u]=N[u']$ が成り立てば, 以後すべての反復において $N[u]=N[u']$ が成り立ち, 最終的に出力されたグラフにおいて, u と u' は隣接することがわかる. よって, G から u を除去したグラフ $G\setminus u$ を考える. すると, 任意の頂点 $v\neq u,u'$ に対して, $u\notin N[v]$ であれば, $G\setminus u$ での $|N[v]|$ は, G での $|N[v]|$ と等しいことがわかり, $u\in N[v]$ であれば, G での $|N[v]|-1$ と等しいことがわかる. また, 任意の頂点対 $v,v'\neq u,u'$ に対して, $u\notin N[v]\cap N[v']$ であれば, $G\setminus u$ での $|N[v]\cap N[v']|$ は, G での $|N[v]\cap N[v']|$ と等しく, $u\in N[v]\cap N[v']$ であれば, G での $|N[v]\cap N[v']|-1$ と等しい. よって, グラフ研磨アルゴリズムの 3. での計算は, 縮約したグラフ $G\setminus u$ を用いて行うことができることがわかる.

G の頂点 u に対して, u と同型な頂点の数, つまり

$N[u] = N[u']$ が成り立つ頂点 u' の数を, $h(u)$ と表記する. また, 頂点同型群 U を U の任意の頂点対 $u, v \in U$ に対して $N[u] = N[v]$ が成り立つ極大な頂点集合とする. G の縮約グラフ $G^*(G)$ を, G の各頂点同型群から 1 つ頂点を選び, それ以外の頂点を消去して得られるグラフとする. このとき, 任意の頂点 v に対して, G での $|N[v]|$ は, $G^*(G)$ での $\sum_{u \in N[v]} h(u)$ と等しいことがわかる. また, 任意の頂点対 v, v' に対して, G での $|N[v] \cap N[v']|$ は, $\sum_{u \in N[v] \cap N[v']} h(u)$ と等しいことがわかる. このことから, 以下のアルゴリズムを得る.

ALGORITHM GraphPolishing ($G = (V, E)$, sim(), θ, τ)
1. set $h(v) := 1$ for all $v \in V$
2. $G' := (V, \{(u, v) \mid \mathrm{sim}(u, v) \geq \theta\})$,
   by compuusing sim with using $h()$
3. **if** $G' = G$ or $i = \tau$ **then output** G; **exit**
4. $G := G^*(G')$
5. $i := i + 1$; **go to** 2

グラフ研磨で得られるグラフには, 少数の極大クリークと孤立点しか存在しないこと, 2 つ以上のクリークに含まれる頂点は少数であることが実験的にわかっている [6], [9]. 特に, このアルゴリズムが扱うある程度の密構造を持つグラフであれば, 極大クリークの大きさは大きくなるはずで, つまり極大クリークの数はグラフの頂点数に比べて小さくなるはずである. よって, グラフ研磨の反復が進むにつれて, グラフ G は, 少数の極大クリークからなるグラフに近づき, そのため同型な頂点が増えていくと考えられる. そのため, 上記の改良アルゴリズムは, 反復が進むにつれ, 一反復の計算時間が減少していくものと期待される.

5. 計算機実験

この節では, 前節で提案したアルゴリズムによる速度の改善がどの程度であるのか, 実データと人工的に作成したランダムグラフを用いて検証を行う.

まず, ランダムグラフの作成方法を記述する. グラフ研磨は, クラスタが埋もれているであろうグラフに対して適用することが多いだろうと考えられるため, クラスタが埋め込まれたランダムグラフを作成する. パラメータとして, グラフの頂点数 n, クラスタサイズ s, クラスタ数 c, 確率 p, q, q' を用意する. まず, n 個の頂点を用意する. そして, s 個の頂点をランダムに選び, その頂点間に確率 p で枝を張る. この作業を c 回繰り返す. 各 s 個の頂点がクラスタに対応し, s が誘導する部分グラフに密度が p のグラフを足しこむことになる.

できたグラフにおける頂点 v の次数を $d(v)$ とする. 次に, 各頂点の次数の比をなるべく変えないようにランダムな枝を足しこむ. 頂点 v の次数の期待値が $d(v) \times q$ となるようなランダムグラフを作り, これをもとのグラフに足しこむ. ランダムグラフの作り方は, 各頂点 v から $d(v) \times q/2$ 本の枝をランダムに選んだ頂点 u に張る. u は, v でないすべての頂点の中から, $d(u)$ に比例する確率で選ぶ.

さらに, 孤立した頂点ができることを防ぐため, 各頂点の次数の期待値が $d(v) \times q'$ の平均値となるように作ったランダムグラフを同じように足しこむ. 上記のグラフと同じように, 各頂点 v から $d(v) \times q'/2$ 本の枝をランダムに選んだ頂点 u に張る. u は, v でないすべての頂点の中から, 一様ランダムに選ぶ.

このグラフを用いて, グラフの頂点数, クラスタの大きさ, ノイズの付加率の変化に伴う計算時間の変化を検証した. 使用した計算機の CPU は Ryzen Threadripper 1990X 3.4GHz で, RAM は 64GB, OS は Linux を用いたが, マルチコア計算は用いず, RAM も 1GB 程度しか使用していない.

まず, グラフの構造の変化に対する高速化の効果を検証する. 1 つ目の実験は, グラフの規模に対する高速化の検証である. $p = 0.5$, $q, q' = 0.25$ を固定し, 頂点数 n を 2,500, 5,000, 10,000, 20,000, 40,000, 80,000, 160,000, 320,000, 640,000 と変化させ, 同時にクリーク数を n/s にしたものと, $2n/s$ にしたものを用意する. 前者は各頂点が平均一回クラスタに所属し, 後者は 2 回である. このことを, 被覆率が 1, および 2 であるという. s は $s = 20, s = 80$ としたものを用意する. 結果は表 1 に示す.

次に, クリークのサイズの変化による高速化の効果を検証する. $p = 0.5$, $q, q' = 0.25$, 頂点数 n を $10,000$ に固定し, クリークの大きさ s を $10, 20, 40, 80, 160, 320, 640, 1,280$ に変化させたグラフを用意する. クリーク数 c は n/s , $2n/s$ にする. 結果は表 2 に示す.

ノイズによる高速化の検証も行う. 頂点数 n を $100,000$ に, クリークの大きさ s を $20, 80$, クリーク数 c は n/s, $2n/s$ にする. そして, $q, q' = 0.25$ として, $p = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1$ に変化させたグラフを作成する. 同様に, $p = 0.5, q' = 0.25$ として, $q = 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.5, 0.7, 1$ に変化させたグラフ, $p = 0.5, q = 0.25$ として, $q' = 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.5, 0.7, 1$ に変化させたグラフを用意する. 結果は表 3,4,5 に示す.

計算結果を見ると, グラフの大きさが大きくなると, またクラスタのサイズが大きくなると, 高速化の効果が高くなっていることが観察される. 所属するクラスタの集合が同じである頂点は, 反復が進むにつれて閉近傍集合が等しくなっていくため, クラスタサイズが大きい, あるいはクラスタの数が大きく, 多くの頂点は単一のクラスタのみに所属する場合に, 高速化がされるであろうと考えられるため, 自然な結果が得られた.

クラスタ内枝密度パラメータ p の変化に対しては, それほど大きな影響は出ていない. これは, 一度反復を行うと, どの p においてもある程度クラスタが明確化された, つまり各クラスタの密度が高いほぼ同質なグラフが得られると考えられ, そのために計算時間の開きにそれほどの差異がないものと考えられる. $p = 1$ のときのみ, 高速化の影響が大きいが, これは最初から枝密度が高く同型な頂点が容易に発見されるためと考えられる.

クラスタ外ノイズ枝のパラメータに関しては, パラメータの変化に対する計算時間の変化が認められなかった. これは, 最初の反復で, ほぼすべてのクラスタ外の枝が取り除かれ, 結果ほぼ等しいグラフが得られているのではないかと推測される.

図 1 は, 入力したグラフの枝数が, 反復を経るごとにどのように変化するかを図示したグラフである. 1 本の線が 1 つのデータの実行に対する枝数の推移を表しており, 横軸が反復数, 縦軸がグラフの枝数を, オリジナルのアルゴリズムが最終的に出力したグラフの枝数で割ったものを記している. つまり, 各線の左の端の高さは, その実行が入力したグラフが, 最終反復の枝数の何倍の枝を含むかであり, 右に行くにしたがって, 高さは, 第 2 反復, 第 3 反復のそれを表している. 赤い線（一本の水平線に収れんする線達）がオリジナルのデータ研磨の実行結果に対応し, 残りの青い線が高速化したものに対応する. 赤い線が収れんした一本の水平線の高さが, 1, つまり出力したグラフの枝数の高さに対応する. これを見ると, ほぼすべての実行で, 枝数は急速に出力グラフの枝数に収束していることがわかる. 高速化の影響が倍から数倍におよぶこともわかる. 高速化の影響が大きい時は, 入力グラフに対して出力グラフが大きいときに対応し, 入力グラフが出力グラフに対してはるかに大きいときには, 入力グラフの処理が計算時間の大半を支配してしまうため, 以後の反復に効果を発揮する今回の高速化手法は効果を発揮しない. これは, 多くのデータに対して, 高速化の効果が入力データが本質的に含有するクラスタの数と大きさのみに依存することを示している.

続いて, ツイッターデータによる実データでの高速化の効果の検証を行う. ツイッターの解析, とくに時系列的なデータの解析では, ときどきバースト的にツイートが増え, 計算時間が非常に長くなる. バーストが起きるときは, 何かの情報が爆発的に広がることが多く, 基本的には同じような文面の投稿が大量に発生するため, 巨大なクラスタが少数発生する. データ研磨はこのような巨大なクラスタの計算にコストがかかってしまうため, 本稿の改良で, このようなバースト時の計算が改善できるかどうかを検証する. 対象としたツイートは, ある企業が炎上した時, そのピーク時 1 時間の, その企業名を含むツイートおよそ 3000 件と, 東日本大震災が発生した直後, 2011 年 3 月 11 日 15:00 から 15:30 のすべてのツイートおよそ 60 万件の 2 つのデータを用意した. すべてのツイートは mecab により分かち書きされ, 単語の集合に変換され, その Jaccard 係数により類似度グラフを作成した. 表には, 類似度グラフを作成するときの閾値を変化させた時の, 実行時間の変化を記録している. なお, 東日本大震災のデータは巨大すぎて計算が終わらないため, 単語集合が 2 以下のツイート, 頻度が 5%以上の単語は計算対象から除外した. 結果を見てみると, それぞれの閾値で十数倍程度の高速化が行われていることがわかる. これは, バーストの原因となる類似する多数のツイートが巨大なクラスタを形成し, これが計算コストを上昇させていると考えられ, 今回の高速化手法はこの部分のコストを圧縮できたために, その効果が高いと考えられる.

6. まとめ

本稿では，データ研磨アルゴリズムを高速化する, グラフの部分的な同型性を利用してグラフを縮約する手法について提案した. 計算機実験の結果, マイクロクラスタリングを行う動機のある, クラスタの大きさが数十であるようなランダムデータに対して, 数倍の高速化が観察された. また現実のツイッターデータにおいて, バーストが発生してデータが一時的に巨大になる場合など, 巨大なクラスタが生成されているときに特に効果を発揮し, 最大 10 数倍の高速化を達成することを示した. データが巨大になった時により効果を発揮することは, 実用性の上で大きな長所となる.

データ研磨はある意味でデータをまんべんなく全体的に処理するため, 高速化の余地が小さい. 今後はモデルの改良も含めた, さらなる高速化の研究が必要となるだろう.

謝辞

この研究は科学技術振興機構 CREST, 課題番号 JPMJCR1401 の補助を受けている.

参考文献

[1] M. Girvan and M. E. J. Newman: Community Structure in Social and Biological Networks, Proc. Natl. Acad. Sci. USA **99**, pp. 7821–7826 (2002)
[2] Takako Hashimoto, Takeaki Uno, Tetsuji Kuboyama, Kilho Shin, Dave Shepard: Time Series Topic Transition Based on Micro-Clustering, BigComp 2019: 1-8 (2019)
[3] G. Karypis, V. Kumar: METIS–Unstructured Graph Partitioning and Sparse Matrix Ordering System, version 2.0. (1995).
[4] 中原孝信, 丸橋弘明, 羽室行信, 宇野毅明: グラフ研磨を利用した顧客クラスタリングによる多様性を考慮した特徴抽出オペレーションズ・リサーチ 64(2) 102 - 109 2019 年 2 月
[5] H. Steinhaus: Sur la division des corps materiels en parties, Bull. Acad. Polon. Sci. 4 (12), pp. 801–804, (1957) (French).
[6] Takeaki Uno, Hiroki Maegawa, Takanobu Nakahara, Yukinobu Hamuro, Ryo Yoshinaka, and Makoto Tatsuta:

表 1 頂点数の増加に伴う計算時間の増加

頂点数	2,500	5,000	10,000	20,000	40,000	80,000	160,000	320,000	640,000
クリークサイズ 20, 被覆率 1, オリジナル	0.293	0.524	0.878	2.509	6.405	14.728	33.934	85.670	269.538
クリークサイズ 20, 被覆率 1, 高速化	0.192	0.326	0.674	1.620	3.168	8.052	21.539	56.571	148.689
クリークサイズ 80, 被覆率 1, オリジナル	1.312	2.095	4.258	10.288	24.714	56.784	133.442	351.613	1169.507
クリークサイズ 80, 被覆率 1, 高速化	0.664	1.243	2.697	5.951	13.519	32.361	74.622	194.402	607.887
クリークサイズ 20, 被覆率 2, オリジナル	0.254	0.380	0.754	1.699	3.465	8.899	24.655	62.934	186.014
クリークサイズ 20, 被覆率 2, 高速化	0.154	0.299	0.680	1.365	2.694	6.885	16.554	43.110	128.456
クリークサイズ 80, 被覆率 2, オリジナル	1.530	1.863	3.816	9.589	19.166	44.113	103.491	241.819	861.507
クリークサイズ 80, 被覆率 2, 高速化	1.088	1.644	3.389	7.306	17.326	36.440	84.263	191.595	709.671

表 2 クリークサイズの増加に伴う計算時間の増加

クリークサイズ	10	20	40	80	160	320	640	1,280
被覆率 1, オリジナル	9.886	18.396	27.169	74.785	198.240	673.287	2920.004	8582.597
被覆率 1, 高速化	6.052	10.329	18.458	42.721	109.951	317.004	938.748	2684.510
被覆率 2, オリジナル	8.262	13.671	21.406	57.746	183.083	618.671	2118.771	7330.756
被覆率 2, 高速化	8.830	9.509	16.621	47.436	160.785	540.700	1815.461	6443.509

表 3 クリーク内の枝密度 p の変化に伴う計算時間の変化

p	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
クリークサイズ 20, 被覆率 1, オリジナル	6.316	10.14	10.98	10.12	18.39	17.253	16.575	20.19	20.94	24.42
クリークサイズ 20, 被覆率 1, 高速化	3.416	3.838	5.056	6.455	10.42	16.302	12.81	15.78	19.89	12.33
クリークサイズ 80, 被覆率 1, オリジナル	6.116	16.93	20.33	35.07	75.61	103.09	194.56	229.88	181.91	209.58
クリークサイズ 80, 被覆率 1, 高速化	4.822	8.382	11.41	19.83	43.02	59.68	102.43	117.56	98.77	103.01
クリークサイズ 20, 被覆率 2, オリジナル	7.238	6.158	7.930	8.832	13.49	18.88	20.76	27.22	35.30	34.35
クリークサイズ 20, 被覆率 2, 高速化	5.168	5.827	6.034	6.779	9.385	13.36	17.74	22.72	28.50	31.08
クリークサイズ 80, 被覆率 2, オリジナル	4.318	10.10	20.95	35.79	59.90	91.78	141.00	208.10	228.49	267.77
クリークサイズ 80, 被覆率 2, 高速化	4.339	10.19	18.06	31.92	45.67	78.38	118.18	176.54	187.55	227.08

表 4 次数倍化型ノイズ枝の変化に伴う計算時間の変化

q	0.05	0.1	0.15	0.2	0.25	0.3	0.5	0.7	1
クリークサイズ 20, 被覆率 1, オリジナル	18.267	18.181	18.510	18.140	18.619	18.482	18.138	18.193	18.623
クリークサイズ 20, 被覆率 1, 高速化	10.545	10.265	10.362	10.430	10.342	10.479	10.380	10.407	10.430
クリークサイズ 80, 被覆率 1, オリジナル	73.118	74.647	74.229	74.343	74.776	73.384	75.345	75.736	73.548
クリークサイズ 80, 被覆率 1, 高速化	42.726	43.105	42.691	43.353	43.189	43.055	42.999	43.093	42.680
クリークサイズ 20, 被覆率 2, オリジナル	13.703	13.959	13.799	13.974	13.638	14.046	13.931	13.479	13.681
クリークサイズ 20, 被覆率 2, 高速化	9.358	9.632	9.696	9.408	9.839	9.584	9.707	9.286	9.920
クリークサイズ 80, 被覆率 2, オリジナル	61.843	58.500	60.531	58.782	62.381	57.711	57.231	56.867	57.817
クリークサイズ 80, 被覆率 2, 高速化	46.391	49.864	47.885	47.426	44.772	50.048	44.956	43.093	49.681

表 5 定数追加型ノイズ枝の変化に伴う計算時間の変化

q'	0.05	0.1	0.15	0.2	0.25	0.3	0.5	0.7	1
クリークサイズ 20, 被覆率 1, オリジナル	16.814	20.277	21.197	18.495	18.155	16.112	15.739	13.657	13.909
クリークサイズ 20, 被覆率 1, 高速化	11.443	11.798	11.049	11.958	10.319	9.734	8.726	9.394	7.424
クリークサイズ 80, 被覆率 1, オリジナル	84.594	100.694	101.526	73.006	73.934	66.991	52.685	39.002	38.709
クリークサイズ 80, 被覆率 1, 高速化	52.711	53.409	53.212	43.497	42.586	37.697	32.969	28.345	31.537
クリークサイズ 20, 被覆率 2, オリジナル	11.574	13.184	12.981	12.061	13.858	10.916	10.181	10.319	9.919
クリークサイズ 20, 被覆率 2, 高速化	9.535	9.452	9.576	9.062	9.634	8.447	7.951	8.314	8.620
クリークサイズ 80, 被覆率 2, オリジナル	60.954	61.890	56.083	56.093	58.699	55.842	61.671	75.021	88.310
クリークサイズ 80, 被覆率 2, 高速化	45.549	45.176	45.220	46.320	49.807	50.921	61.693	69.250	92.348

表 6 企業 A の炎上時と東日本大震災地震直後のツイートでの類似度グラフ閾値の変化に対する算時間の変化

閾値	0.1	0.15	0.2	0.25	0.3	0.4	0.5	0.7	1
炎上, オリジナル	209.774	284.499	221.441	232.835	187.142	69.653	43.887	9.432	5.105
炎上, 高速化	40.392	35.533	24.949	20.944	17.296	6.802	3.729	1.469	0.705
東日本大震災, オリジナル	>80000	43720.29	24565.69	20599.28	9477.51	4608.88	3518.30	738.89	554.28
東日本大震災, 高速化	8513.79	3762.25	2215.53	1334.25	635.57	317.70	261.32	101.41	83.77

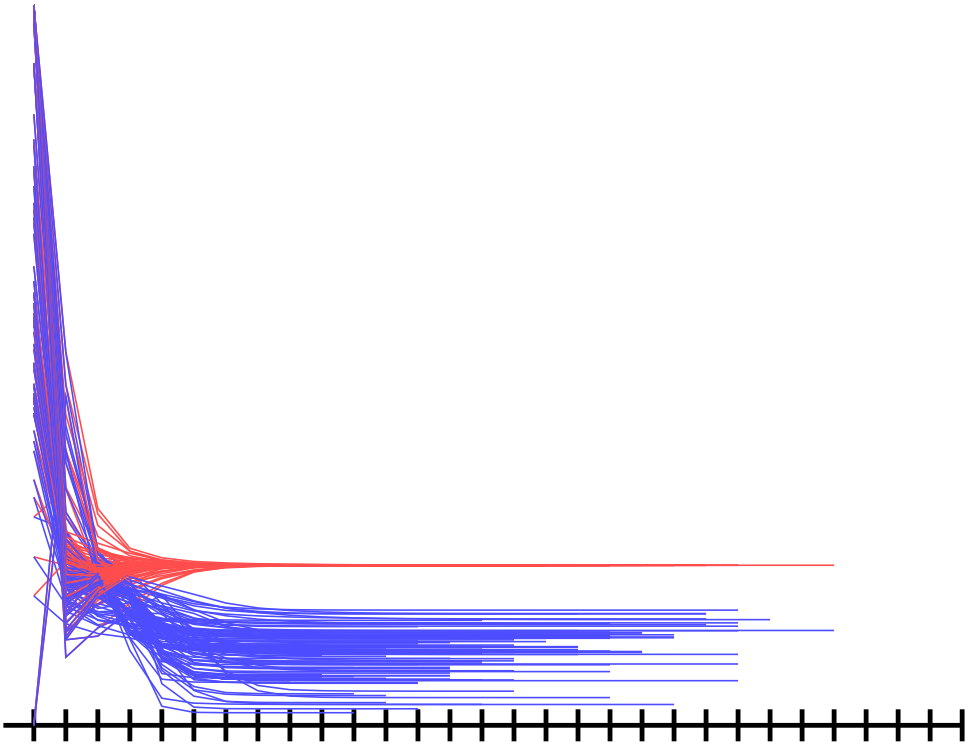

図 3 アルゴリズムの収束の様子. 横軸は反復回数, オリジナルのアルゴリズムが縦軸は出力したグラフの枝数に対する, k 回目の反復で入力したグラフの枝数. 中央の水平な一本の線の高さが出力グラフの枝数

Micro-Clustering by Data Polishing, IEEE Big Data 2017 (2017).
[7] 宇野毅明, 岩崎幸子, 中原孝信, 中元政一, 羽室行信: 乱数シード依存のクラスタリング手法の安定化に対するアプローチ, 人工知能基本問題研究会 105, pp. 58–62 (2018).
[8] 宇野毅明, 岩崎幸子, 中原孝信, 中元政一, 羽室行信: データ研磨によるコンセンサスクラスタリングの精緻化, 人工知能基本問題研究会 106, pp. 43–50 (2018).
[9] 宇野毅明, 中原孝信, 前川浩基, 羽室行信: データ研磨によるクリーク列挙クラスタリング, 第 146 回情報処理学会アルゴリズム研究会 146, pp. 1–8 (2014).