極大クリーク列挙を用いた 高速なkクリークコミュニティのオンライン探索

矢野　洋祐†　　照山　順一††　　吉田　悠一††

† 東京大学情報理工学系研究科　〒 113–8656 東京都文京区本郷 7–3–1
†† 国立情報学研究所　〒 101–8430 東京都千代田区一ツ橋 2–1–2
E-mail: †yyano@is.s.u-tokyo.ac.jp, ††{jteruyama,yyoshida}@nii.ac.jp

あらまし　現実世界のグラフにはコミュニティという密な部分構造が存在することが知られており，その検出はネットワークを解析する上で重要である．しかし，巨大なグラフにおいて全てのコミュニティを検出することは困難なため，近年になって指定された頂点を含むようなコミュニティを探索する問題が提唱された．k クリークコミュニティは頂点が複数のグループに属することを許すクリークに基づいたコミュニティモデルである．本論文では，クエリ頂点を含む k クリークコミュニティを列挙する問題に対して新しいアルゴリズムを提案する．提案手法は，必要な極大クリークのみを注意深く列挙しながらそれらの隣接関係を計算することにより高速なコミュニティを列挙を実現する．実験により，提案手法が既存手法より高速に動作することを示した．

キーワード　グラフ，コミュニティ検出，極大クリーク列挙

1. はじめに

ネットワークにおいて，内部で密に繋がっており外部との結合は疎であるような部分構造をコミュニティと呼ぶ．ソーシャルやウェブのネットワークにはしばしばこのような構造が見られることが知られており，コミュニティの検出は実ネットワークの解析において重要な役割を果たすと考えられている．これまで応用に応じて様々なコミュニティ[1,10,11] が提唱されている．多くの定義では各頂点が最大 1 つのコミュニティに属するモデルとなっているが，近年では 1 つの頂点が複数のコミュニティに属しうることが明らかになって来ている [1,12].

頂点が複数のコミュニティに属することを許可するコミュニティ(overlapping community) のモデルとしては，クリークパーコレーション，あるいは k クリークコミュニティ[1] と呼ばれるものがよく研究されている．k クリークコミュニティのモデルでは，大きさ k のクリークをコミュニティの構成要素と考え，$k-1$ 頂点を共有する k クリーク同士は同じコミュニティに属しているものみなして統合したものをコミュニティとする．k クリークコミュニティの特徴として重複を許すことの他に，k の値を調整することによって異なる粒度のコミュニティを得ることが出来ることが挙げられる．

本論文では k クリークコミュニティに関連して，Cui らによって 2013 年に提唱された次の問題 [13] を扱う．

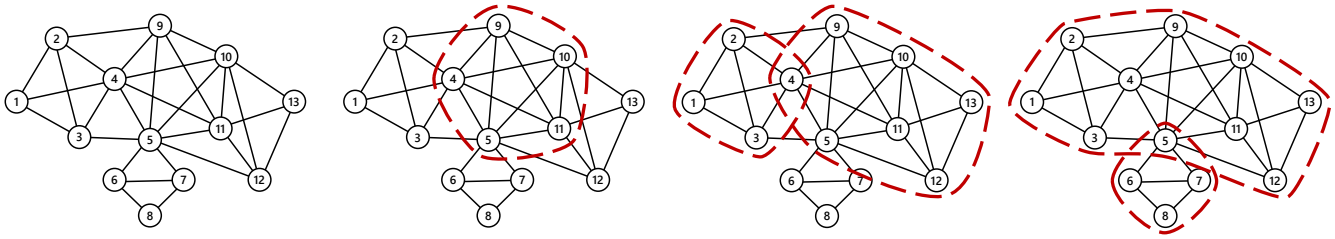
問題 1 (Overlapping Community Search (OCS))．与えられたグラフ $G=(V,E)$ とクエリ頂点 $q \in V$，パラメータ k に対して頂点 q を含む k クリークコミュニティを全て求めよ．

グラフ全体のコミュニティの列挙ではなく，ある頂点の周りの k クリークコミュニティのみを考える利点は以下の通りである．

- 大規模なデータに対しても適用できる．近年現実のネットワークが大規模化しており，それら全体のコミュニティを見つけるには大変なコストがかかるが，クエリ頂点の周りの探索問題を考えることによって，いくつかの頂点に関してコミュニティを見つけたい場合に計算時間を大幅に短縮することが出来る．
- グラフ全体に対して同じパラメータ k を使うのではなく，クエリ頂点に応じた k の値を設定することができる．k クリークコミュニティでは k の値を小さくしたときに特に密な部分で非常に大きなコミュニティが検出されてしまったり，逆に大きくした時に比較的疎な部分で全くコミュニティが検出されないということが起こる場合があるが，クエリ形式の問題を考えることによってクエリ毎に k の値を変えることができるようになり，適切な粒度のコミュニティを得ることが容易となる．
- 動的グラフへの対応が容易である．動的グラフ上で全体のコミュニティを維持するためには辺の削除や追加に対応したアルゴリズムを設計する必要があるが，クエリ形式の問題では関係する部分のみ再びクエリを発行しなおせばよい．

したがって，大規模なネットワーク上でのコミュニティ検出を考える際にはこのような設定は現実的な選択肢となる．しかしながら Cui らの論文 [13] の貢献は重複構造を有するコミュニティの探索に関する一般的な枠組みを提唱したことであり，k クリークコミュニティの探索アルゴリズムに関しては単純なものを使用している．精度保証の無い近似を入れることによって高速なコミュニティの発見を達成しているが，厳密アルゴリズムの実行時間には問題があった．

そこで本論文では，クエリ頂点の周りの k クリークコミュニティ探索する問題に対して，より効率的なアルゴリズムを提案する．提案手法は，多くの場合冗長な k クリークを探索する代わりに極大クリークのバックトラッキング探索 [16–18] を元に



(a) 元のグラフ　(b) 5 クリークコミュニティ　(c) 4 クリークコミュニティ　(d) 3 クリークコミュニティ

図 1　グラフと k クリークコミュニティの例.

したシンプルなアルゴリズムである．極大クリーク探索のアルゴリズムは元々グラフ全体のクリークを列挙するものであるが，アルゴリズムの実行時間がグラフ全体の大きさに影響を受けないようにするためにはクエリ頂点の周りの k クリークコミュニティに含まれる極大クリークのみを上手く列挙する必要がある．提案手法では素集合データ構造を用いて極大クリーク同士の隣接関係を注意深く管理することにより必要な極大クリークのみの探索を実現する．

更に我々は様々な現実のネットワークを用いて計算機実験を行った．提案手法と既存手法に関してクエリごとの実行時間の詳細に比較し，全体として実行時間が改善していることを示した．特にウェブグラフでは数十倍から数百倍の高速化を達成した．

2. 関 連 研 究

2.1 重複構造を有するコミュニティの検出

Palla らは現実のネットワークが重複したコミュニティを持つことを指摘し，k クリークコミュニティを探索する手法を提案した [1]．生物学ネットワーク上の k クリークコミュニティを検出するために CFinder [2] というソフトウェアが開発されており，また近年にもクリークグラフの全域木を考えて不必要なクリークの隣接判定を減らし，高速に k クリークコミュニティを列挙する手法が提案されている [3]．クエリ頂点の周りのコミュニティのみの列挙を考えるという問題設定は近年 Cui らによって提案され [13]，k トラスコミュニティ[4] やその他のいくつかのモデル [14] に対しても拡張されている．k トラス [4,15] は k クリークコミュニティに比べて繋がりの緩いコミュニティのモデルであり，それぞれ応用範囲が異なる．

2.2 極大クリーク列挙

k クリークや極大クリークを列挙する問題は非常によく研究されている．一般のグラフにおいて k クリークが存在するか判定する問題は NP 完全であることが知られている [5]．クリーク検出の実用的な手法としては，局所探索手法が広く用いられている [6,7]．グラフ G の任意の部分グラフが d 以下の次数の頂点を含む時，そのような最小の d を degeneracy あるいは core number と呼ぶ．Degeneracy が小さいグラフにおいては k クリークや極大クリークが効率的に列挙できることが知られている [8,18]．現実世界のネットワークは degeneracy が小さい傾向にある [9] という研究結果もあり，これらのアルゴリズムの実用的な効率性の裏付けの 1 つとなっている．k クリークコミュニティの検出には，クエリ頂点の周りのクリークのみでなく，クリーク同士の隣接関係に応じて探索する頂点を増やさなければならず，そのような問題を考えた既存研究はこれまでにはなかった．

3. 準　　備

3.1 用語・記法

$G=(V,E)$ を単純無向グラフとする．G の頂点数と辺数をそれぞれ n,m で表す．頂点 $v\in V$ に隣接する頂点の集合を $N_G(v)$ と表し，頂点集合 $S\subseteqq V$ に対して $N_G[S]$ を $S\cup\bigcup_{v\in S}N_G(v)$ と定義する．文脈から明らかなとき，単に $N(v),N[S]$ と記す．頂点集合 S が誘導する部分グラフを $G[S]$ と表記する．

頂点数 k のクリークを k **クリーク**と呼ぶ．2 つの k クリークが $k-1$ 頂点を共有しているとき**隣接している**という．k クリークの集合が**連結である**とは，集合中のどの 2 つの k クリーク同士も，隣接する k クリークをたどって到達できることをいう．ここで k クリークコミュニティは次のように定義される．

定義 1 (k クリークコミュニティ)．k **クリークコミュニティ**とは，グラフ上の連結な k クリークの集合であって極大なものをいう．

図 1 に k クリークコミュニティの例を示した．点線で囲まれている部分が 1 つの k クリークコミュニティを表している．

3.2 クリークグラフ

k クリークコミュニティに関する議論のため，**クリークグラフ**という道具を導入する．グラフ $G=(V,E)$ のクリークグラフ $K(G)=(V_C,E_C,w)$ を以下のように定義する．$K(G)$ は頂点集合を V_C，辺集合を E_C，コスト関数を $w:E_C\to\mathbb{N}$ とするような辺重み付き無向グラフであり，それぞれの頂点は G 中の極大クリークと一対一対応する．クリークグラフ上の頂点 $v\in V_C$ に対応する G 上のクリークを $C(v)\subseteqq V$ と表記する．クリークグラフ上の 2 頂点 $u,v\in V_C$ に関して，$C(u)$ と $C(v)$ が少なくとも 1 つ頂点を共有するときのみ辺 (u,v) が E_C に属しているとし，辺 (u,v) の重みは 2 頂点が共有する頂点の数とする．

クリークグラフに関して，(1) 大きさ k 以上の極大クリーク内のどの k 点も k クリークをなすこと，(2)$w(u,v)\geqq k-1$ なる $u,v\in V_C$ について隣接するような 2 つの k クリーク

Algorithm 1 BronKerboschPivot(R, P, X)

Input: $R, P, X \subseteqq V$

1: **if** $|R \cup P| < k$ **then return**
2: **if** $P \cup X = \emptyset$ **then** ReportMaximalClique(R).
3: $|P \cap N(u)|$ を最大化するピボット $u \in P \cup X$ を選択.
4: **for each** $v \in P \setminus N(u)$ **do**
5: BronKerboschPivot$(R \cup \{v\}, P \cap N(v), X \cap N(v))$.
6: $P \leftarrow P \setminus \{v\}$.
7: $X \leftarrow X \cup \{v\}$.

$A \subseteqq C(u), B \subseteqq C(v)$ が存在することに注意すると，次のことがわかる．グラフ G 上の k クリークコミュニティは，$K(G)$ において大きさ k 以上の極大クリークと重み $k-1$ 以上の辺のみを考えた場合の連結成分に対応する．したがって，k クリークコミュニティを見つけるためには極大クリークとそれらが共有する頂点の数を考えればよい．

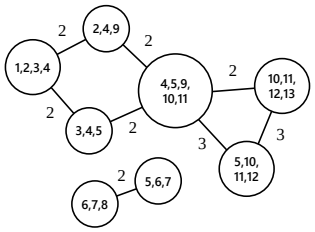


図 2　図 1 に対応するクリークグラフ（重み 1 の辺は省略されている）．

3.3 極大クリーク列挙

クリーク列挙は k クリークコミュニティ探索において重要な役割を果たす．我々の手法では k クリークの列挙の代わりに極大クリーク列挙を，Bron-Kerbosch アルゴリズム [16–18] を用いて行う．

Algorithm 1 に Bron-Kerbosch アルゴリズムの擬似コードを示した．BronKerboschPivot(R, P, X) は，G の部分グラフ $G[R \cup P \cup X]$ 上で，R に属する頂点を全て含み，P に属する頂点をいくつか含み，X に属する頂点を 1 つも含まない極大クリークを列挙するルーチンである．R は極大クリークの部分グラフとして既に選んだ頂点集合であり，P と X は R 中の頂点全てに隣接している頂点集合の分割になっている．つまり，$P \cup X$ は現在のクリーク R に追加する頂点の候補であると考えることが出来る．P から頂点を 1 つずつ選び R に追加したのち（4 行目），P と X を R 中の全頂点と隣接するよう適切に更新して再帰的にルーチンを呼び出し（5 行目），$P \cup X$ が空であれば R を極大クリークとして報告する（2 行目）というのが基本的な動作になっている．P から頂点を選ぶ際に既に候補として選んだ頂点を X に追加していくことで，重複して極大クリークを探索することを防いでいる．

実際のアルゴリズムでは，それぞれの再帰呼び出しの際に P に属する頂点全てを選ぶ必要はなく，ある頂点 $u \in P \cup X$ に対して $P \setminus N(u)$ に属する頂点を R に追加する候補として選べばよい．なぜなら，R を含む極大クリークは必ず $(P \cup X) \setminus N(u)$ 中の頂点を少なくとも 1 つは含むからである．もし R と $N(u)$ の部分集合のみを含む極大クリークがあったとすると，それらは全て u に隣接しているため u を追加してもクリークであるが，これは極大性に矛盾する．このような頂点 u はピボットと呼ばれる．

ピボットとして $|P \setminus N(u)|$ を最小にするような u を選ぶことによって理論上，また実用上も実行時間を大幅に短くできることが知られている [17, 18]．また今回のタスクにおいては大きさ k 以上の極大クリークのみを考えれば良いため，$|R \cup P|$ が k より小さい場合には枝刈りを行っている（1 行目）．

Algorithm 2 SearchCommunity(G, q, k)

Input: $G = (V, E)$, $q \in V$, $k \in \mathbb{N}$.

1: $Q \leftarrow \{q\}$, $\overline{Q} \leftarrow \emptyset$.
2: $\mathcal{C} \leftarrow \emptyset$, $\mathcal{A} \leftarrow \emptyset$.
3: $P \leftarrow V$, $X \leftarrow \emptyset$.
4: **while** Q が空でない **do**
5: $v \leftarrow Q$ 中の頂点を 1 つ選択.
6: $Q \leftarrow Q \setminus \{v\}$, $\overline{Q} \leftarrow \overline{Q} \cup \{v\}$.
7: BronKerboschPivot$(\{v\}, P \cap N(v), X \cap N(v))$.
8: $P \leftarrow P \setminus \{v\}$, $X \leftarrow X \cup \{v\}$.
9: **return** SynthesizeActiveCliques$(\mathcal{A})$.

4. 提 案 手 法

本節ではクエリ頂点を含む k クリークコミュニティを，極大クリーク列挙を用いて高速に探索するアルゴリズムを提案する．まず手法の概要を述べ，その後それぞれの操作の詳細を説明する．提案手法の正当性と計算量についても議論する．

4.1 手 法 概 要

ここでは提案手法の概要を述べる．グラフ $G = (V, E)$ とクエリ頂点 $q \in V$，パラメータ k を入力とする．我々の手法では，まず頂点 q を含むサイズ k 以上の極大クリークを列挙する．その後，「列挙したクリークに含まれる頂点」を含む極大クリークを必要に応じて列挙し，それらの隣接関係を判定しながら適切にクリーク同士をマージしていくという操作を繰り返して k クリークコミュニティを探索する．提案手法では，k クリークの代わりに多くの場合より数の少ない極大クリークを考えることで効率的に探索を行うが，必要な極大クリークのみをローカルに列挙するには注意深くアルゴリズムを設計する必要がある．

Algorithm 2 に提案手法のアルゴリズムの大枠を示した．Q はこれから極大クリークを探索するべき頂点の集合を，$\overline{Q}$ は既に処理した頂点の集合を表し，それぞれクエリ頂点のみから成る集合 $\{q\}$ と空集合で初期化する．クエリ頂点 q を含む k クリークコミュニティに含まれることがわかった極大クリーク C をアクティブであると呼ぶことにする．$\mathcal{C}$ はアルゴリズム中で発見された極大クリークの集合であり，$\mathcal{A}$ はその中でアクティブであるとわかった極大クリークの集合を表す．これらの集合を用いて新しく見つかった極大クリークがアクティブであるかどうか判定する．

Algorithm 3 ReportMaximalClique(C)

1: $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\}$.
2: $\mathcal{N} \leftarrow \{N \in \mathcal{C} \mid N$ は C と $k-1$ 点以上共有している $\}$.
3: **if** $q \in C$ or $\mathcal{A} \cap \mathcal{N} \neq \emptyset$ **then** active $\leftarrow$ **true**.
4: **else** active $\leftarrow$ **false**.
5: **for each** $N \in \mathcal{N}$ **do**
6: **if** active and $N \notin \mathcal{A}$ **then**
7: $\mathcal{L} \leftarrow$ list(N).
8: $\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{L}$.
9: **for each** $X \in \mathcal{L}$ **do**
10: **for each** $v \in X$ **do**
11: **if** $v \notin \overline{Q}$ **then** $Q \leftarrow Q \cup \{v\}$.
12: union(C, N).

Q が空でない間，Q から頂点を取り出し（その頂点を v とする），v を含む極大クリークを列挙する（Algorthm 1）．この際，BronKerboschPivot における P と X を適切に設定することによって全体を通して重複なく極大クリークが列挙される．$\mathcal{C}$，$\mathcal{A}$，Q は，BronKerboschPivot 中で極大クリークを発見し ReportMaximalClique を呼び出した際に更新される (Algorithm 3)．この更新処理がアルゴリズムの重要な部分である．Q が空になった後，それまでに発見したアクティブな極大クリークを統合し (Algorithm 4)，k クリークコミュニティを得る．

4.2 素集合データ構造によるコミュニティの管理

どの極大クリークが同じコミュニティに属しているかを管理するために，大きさ k 以上の極大クリークを頂点とする（一般的な）素集合データ構造 [19] を用いる．それぞれの集合は同じ k クリークコミュニティを構成する極大クリークの集合を表し，クリーク同士の隣接関係が判明した場合にそれらが属するコミュニティをマージしながら管理する．

通常の素集合データ構造は，要素 e を受け取り e の属する集合の ID を返す find 操作と，要素 e と e' を受け取りそれらが属する集合をマージする union 操作をサポートする．素集合森を用いたデータ構造では，これらの操作はならし時間 $O(\alpha(n))$ で行うことが出来る [19]．我々の手法ではそれに加えて，要素 e を受け取り e の属する集合を返す list 操作が必要となるが，この操作は e の属する集合に対応する木を根から走査することにより，出力に線形な時間で行うことが出来る．ReportMaximalClique と SynthesizeActiveCliques のアルゴリズム中では，union 操作と list 操作をそれぞれ union，list で表している．

4.3 極大クリーク発見時の処理

Bron-Kerbosch アルゴリズム内で頂点数 k 以上の極大クリーク C を発見した際には，ReportMaximalClique を呼び出し $\mathcal{C}$，$\mathcal{A}$，Q を更新する (Algorithm 3)．まず，これまで発見した極大クリークの集合である $\mathcal{C}$ には単に C を要素として追加すれば良い．次に C がアクティブなクリークであるか，すなわち，現時点でクエリ頂点 q を含むとわかっているようなコミュニティを構成するクリークであるかを判定する．$\mathcal{N} \subseteq \mathcal{C}$ を C と $k-1$ 点以上共有するクリークの集合であるとする（$\mathcal{N}$ は C を含むことに注意する）．C が q を含むときは明らかに C はアクティブである．また，$\mathcal{N} \cap \mathcal{A} \neq \emptyset$ であるとき，つまり C と $k-1$ 頂点以上共有するようなアクティブなクリークが存在するときも C はアクティブになる（3 行目）．それぞれの頂点に対してその頂点が属している極大クリーク（のうち既に発見したもの）のリストを保持しておくことで $\mathcal{N}$ を効率的に計算することが出来る．C に含まれる頂点に関してそのリストを走査してどのクリークが何回現れたかを数え，$k-1$ 回以上現れたものが $\mathcal{N}$ に属しているとすればよい．

Algorithm 4 SynthesizeActiveCliques($\mathcal{A}$)

Input: $\mathcal{A}$
1: $\mathcal{K} \leftarrow \emptyset$.
2: **for each** $A \in \mathcal{A}$ **do**
3: $\mathcal{L} \leftarrow$ list(A).
4: **if** $\mathcal{L}$ がまだ処理されていない **then**
5: $\mathcal{K} \leftarrow \mathcal{K} \cup \{\bigcup_{C \in \mathcal{L}} C\}$.
6: **return** $\mathcal{K}$.

クリーク C がアクティブであることがわかった場合，$\mathcal{N}$ に属するクリークと，更にそれらと同じコミュニティに属するクリークも全てアクティブであることがわかる．したがって，$N \in \mathcal{N}$ に対して以下の操作を行う．まず C がアクティブかつ N がアクティブでなかった場合，list 操作を N に対して行い N と同じコミュニティに属するクリークを列挙する．そして，これらのクリークに属する頂点のうちまだ極大クリークを探索していない頂点を Q に追加する（11 行目）．その後，C と N の間で union 操作を行いクリーク同士の隣接関係を保持している素集合データ構造を更新する (12 行目)．

4.4 アクティブなクリークの統合

最後に SynthesizeActiveCliques でアクティブな極大クリークを統合し k クリークコミュニティを返す (Algorithm 4)．これは単純にアクティブなクリークに対して list 操作を適用し，得られたクリークに含まれる頂点を統合すればよい．

4.5 正当性と計算量

BronKerboschPivot(R, P, X) が R 中の全ての頂点と P 中のいくつかの頂点を含み，X 中の頂点を含まないような大きさ k 以上の極大クリークのみを全て列挙することは [17] のアルゴリズムの正当性より直ちに導かれる．このことを用いて，提案手法の正当性を簡単に示すことが出来る．

定理 1. SearchCommunity(G, q, k) は G 中の頂点 q を含む k クリークコミュニティを全て列挙する．

Proof. 提案手法が q を含む k クリークコミュニティに含まれる（大きさ k 以上の）極大クリークを全て発見する（アクティブにする）ことを示せば良い．SearchCommunity がトップレベルで頂点 v を Q から取り出して BronKerboschPivot を呼び出す際には，v を含む全ての極大クリークを列挙し，それぞれについて互いが $k-1$ 点以上を共有しているかどうかを判定している．ここである極大クリーク C であって頂点 q を含む k クリークコミュニティに含まれるものがアクティブにならなかったと仮定すると，C に含まれる頂点全てが Q に追加されなかっ

たことになるが，それは C と $k-1$ 点以上共有する極大クリークがアクティブにならなかったことを意味する．これを繰り返して極大クリークを辿って行くと k クリークコミュニティの定義より頂点 q を含む極大クリークに到達でき，q が Q に追加されなかったことが導けるが，q はアルゴリズムの最初で Q に追加されている．これは矛盾であるため，C が発見されなかったという仮定が誤りであることがわかる．したがって提案手法は q を含む k クリークコミュニティに含まれる極大クリークを全てアクティブにする． □

また，提案手法の時間計算量に関して次のことが言える．

定理 2. n 頂点のグラフ上での BronKerboschPivot の時間計算量を $T(n)$ とする．SearchCommunity 内で列挙された極大クリークを $\mathcal{C}$ で表し，$N = \bigcup_{C\in\mathcal{C}} N[C]$ とおくと，SearchCommunity(G,q,k) の時間計算量は $O(T(|N|) + |\mathcal{C}|\sum_{C\in\mathcal{C}}|C| + |\mathcal{C}|^2\alpha(|\mathcal{C}|))$ でおさえられる．ただし $\alpha(x)$ は逆アッカーマン関数である．

部分グラフ $G[N]$ の極大クリーク列挙にかかる時間は全体で $T(|N|)$ である．極大クリークが発見された際にはまず集合 $\mathcal{N}$ を求めるために，そのクリークの全頂点に対して頂点が属している極大クリークのリストを走査する（ReportMaximalClique の 2 行目）．また隣接している極大クリークに対して，union 操作を最大 $|\mathcal{C}|^2$ 回行う（ReportMaximalClique の 12 行目）．前者は $O(|\mathcal{C}|\sum_{C\in\mathcal{C}}|C|)$ 時間，後者は $O(|\mathcal{C}|^2\alpha(|\mathcal{C}|))$ 時間で計算できる．7 行目の list 操作と 11 行目で Q に頂点を追加する操作は，1 つのクリークが高々1 度しかアクティブにならないことを考えると全体で $O(\sum_{C\in\mathcal{C}}|C|)$ 時間しかかからないためボトルネックにはならない（$|\mathcal{C}| \leqq \sum_{C\in\mathcal{C}}|C|$ であることに注意せよ）．最悪の場合の計算量は $T(n) = O(n3^{n/3})$，$|\mathcal{C}| = O(3^{n/3})$ であることが Tomita らにより示されている [17].

4.6 頂点選択の順番

SearchCommunity において頂点を Q から取り出す順番，また BronKerboschPivot において頂点を探索する順番には自由度がある．実際，頂点選択の順番は極大クリーク列挙の性能に少なからず影響を与えることが知られており，[18] では適応的次数（既に選択した頂点を除いたグラフでの次数）の小さい順に頂点を選ぶ手法を提案しており，この手法では時間計算量が degeneracy という値を使って抑えられることが示されている．

しかしながら我々の提案手法に関しては，適応的次数順を用いてもあまり性能に変化が見られないことが予備実験の結果からわかった．全体のクリークを列挙する場合と異なり Q に頂点が徐々に追加されることによって，しばしば適応的次数の高いを選ばざるを得なくなることが原因である考えられる．本論文の実験では，Q に追加した順番に頂点を選択するという戦略を採用した．

5. 実　　験

本節では，提案手法の性能を検証するために行った計算機実験の結果を示す．

表 1　実験データセット

データセット	種類	$\|V\|$	$\|E\|$
Epinions	Social	75,888	405,740
NotreDame	Web	325,729	1,090,108
Google	Web	875,713	4,322,051
Skitter	Computer	1,696,415	11,095,298
LiveJournal	Social	4,847,571	42,851,237

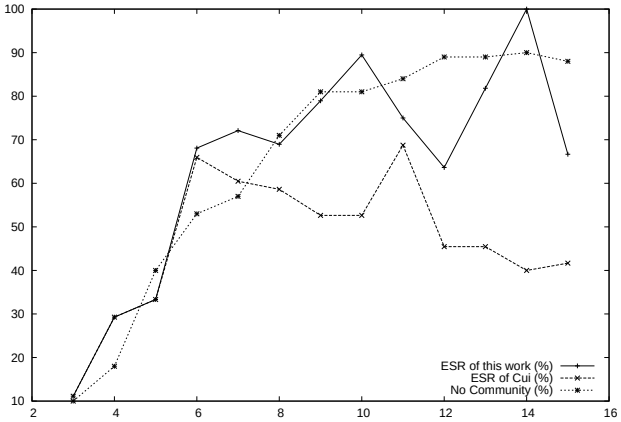


図 4　パラメータ k による実効成功率の変化

5.1 実 験 概 要

実験は Linux サーバ（CPU : Intel Xeon X5675，メモリ : 288GB）上で行った．提案手法は C++で実装した．また Cui らによる既存手法（Cui と表記する）[13] も我々が C++によって再実装を行った．既存手法の再実装に関しては，[13] で言及されている k クリークコミュニティに特化した最適化を含めて行った．

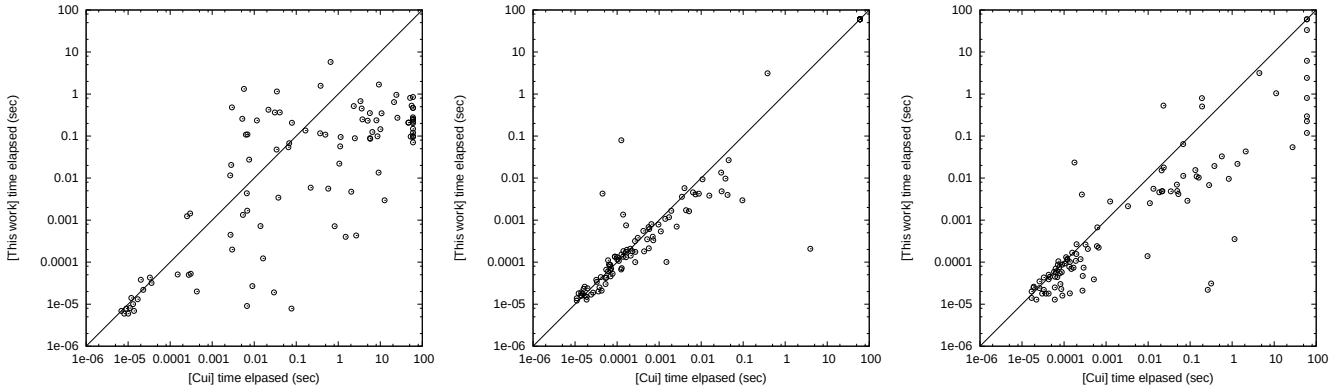
実験に使用したデータセットは表 1 に示した．これらのデータは Stanford Large Network Dataset Collection(注1) で公開されているものである．以下の実験ではこれらのグラフを単純無向グラフに変換して使用した．

5.2 実行時間の比較

まず提案手法と既存手法の実行時間を比較する実験を行った．各データセットと k の値ごとに，次数が $k-1$ 以上の頂点の中からクエリ頂点を一様ランダムに 100 点選び，クエリに応答するまでの時間を測定した．60 秒でクエリに答えられなかった場合はタイムアウト扱いとし，60 秒でクエリに答えたとみなして平均クエリ実行時間を求めた．また k の値によってはほとんどのクエリ頂点がコミュニティに含まれないという場合もあることを考慮し，少なくとも 1 つのコミュニティを返すクエリのうち，タイムアウトせずに答えを返した割合を調べた．この割合を実効成功率 (Effective Success Ratio) と呼ぶことにする．各々のデータセットについて，k の値はコミュニティを返すクエリの割合と出力の大きさを考慮して定めた．

表 2 に実験の結果を示した．NotreDame，Google，LiveJournal のデータセットでは，平均実行時間・実効成功率共に，提案手法が既存手法の性能を上回っていることが読み取れる．特に

(注1)：http://snap.stanford.edu/index.html



Google ($k = 7$)　　Skitter ($k = 10$)　　LiveJournal ($k = 10$)

図 3　提案手法と既存手法 (Cui [13]) の比較

表 2　提案手法と Cui [13] の実行時間の比較．RT，TO，ESR はそれぞれ平均実行時間，実行に 60 秒以上かかりタイムアウトしたクエリ数，実効成功率を表す．NC はどの k クリークコミュニティにも含まれていなかったクエリ頂点の数を表す．

Dataset	k	提案手法			Cui [13]			NC
		RT (秒)	TO	ESR	RT (秒)	TO	ESR	
Epinions	8	10.201	17	19.0%	10.223	17	19.0%	79
	9	11.516	19	20.8%	10.945	18	25.0%	76
	10	9.396	15	28.6%	10.376	17	19.0%	79
NotreDame	5	1.174	0	100.0%	13.212	18	70.1%	38
	6	0.258	0	100.0%	12.580	19	58.7%	54
	7	0.073	0	100.0%	13.873	23	39.5%	62
Google	5	0.470	0	100.0%	13.114	16	80.7%	17
	6	0.381	0	100.0%	9.118	11	84.5%	29
	7	0.113	0	100.0%	12.308	12	84.2%	24
Skitter	8	9.014	15	11.8%	9.032	15	11.8%	83
	9	7.800	13	0.0%	7.861	13	0.0%	87
	10	5.401	9	0.0%	5.447	9	0.0%	91
LiveJournal	8	5.728	9	69.0%	8.125	12	58.6%	71
	9	3.027	4	78.9%	5.788	9	52.6%	81
	10	1.595	2	89.4%	5.916	9	52.6%	81

NotreDame と Google のデータセットでは，提案手法は 100 クエリ全てに対して時間内に答えることができた．したがってこれらのネットワークに対するクエリでは，出力の大きさ等ではなくクリークの列挙とその隣接性の判定がボトルネックになっており，提案手法はその問題を解決できていることがわかる．Epinions と Skitter のネットワークにおいては，両手法とも実効成功率が低くなっている．これらのネットワークには非常に大きいコミュニティが存在すると考えられ，今回のタスクに関して難しいデータセットであったと考えられる．

さらに，いくつかのデータセットに関して各々のクエリの実行時間を比較した（図 3）．それぞれの点が 1 つのクエリを表し，横軸が Cui のクエリ時間，縦軸が提案手法のクエリ時間である．両対数プロットであることに注意する．つまりある点を見た時にそれが対角線上にあれば両手法が同じ時間でクエリに答えたということであり，右下にあるほど提案手法が優れており左上にあるほど既存手法が優れていることを示す．Google, LiveJournal に関しては，全てのクエリで提案手法が優れているわけではないものの，多くのクエリで数倍から 10000 倍程度クエリ時間の差があることが読み取れる．Skitter では多くのクエリ頂点がコミュニティに含まれないため差は小さいが，多くのクエリの実行速度は同程度から 10 倍程度となっている．

5.3　パラメータ k による性能変化

ここではパラメータ k がどのように性能に影響するかを議論する．表 2 の結果からは，全体的に k の値が大きい方が提案手法が優れている傾向にあることが読み取れる．更に我々は LiveJournal のデータセットを用いて，k の値を 3 から 15 まで変化させた時の実効成功率の推移を調べた（図 4）．$k \leq 6$ の範囲では提案手法も既存手法もほぼ同じ結果となっており，実効成功率は比較的低い．これは k が小さい範囲ではコミュニティの基準が緩くなり，それぞれのコミュニティが非常に大きくなってしまうからであると考えられる．しかしそれより大きい k の値では，提案手法は成功率が高くなる傾向にある一方，

Cui では徐々に成功率は下がっていく傾向にある．k が大きくなるにつれてコミュニティとそれを構成する極大クリークの数は小さくなっていくため，提案手法の成功率が高くなるのは自然である．既存手法の成功率が低下していく理由は，k が大きくなるほどクリークの探索に時間がかかるようになることと，k クリーク自体の数も増える可能性があるからであると考察する．提案手法は k の値が大きいほどコミュニティを高速に見つけられるという性質があるため，実際にアルゴリズムをコミュニティ検出に使用する際に，コミュニティのサイズとクエリ実行時間を考慮しながら徐々に k の値を小さくしていくという使い方が可能であると考える．

6. おわりに

本論文では，ある頂点の周りの k クリークコミュニティを探索する問題に対して実用的に高速なアルゴリズムを提案した．提案手法は極大クリークの列挙アルゴリズム [16–18] に基づいており，極大クリーク同士の接続関係を素集合データ構造で管理しながら，クエリ頂点を含む k クリークコミュニティに属する極大クリークとその周りのみを探索することで効率的にコミュニティを発見する．実世界のネットワークを用いた計算機実験により，我々の手法がほとんどのクエリに対して実行時間で既存手法 [13] を上回ることを示した．提案手法は，既存手法の適用が難しかった大規模かつ密なネットワークでのコミュニティ解析を可能にし，ソーシャルやウェブネットワーク上での種々のタスクに応用されることが期待される．

謝　辞

本研究は JST，ERATO，河原林巨大グラフプロジェクトの助成によるものである．ここに記して謝意を表す．

文　献

[1] Gergely Palla, Imre Derényi, Illés J. Farkas and Tamás Vicsek, “Uncovering the overlapping community structure of complex networks in nature and society,” Nature 435(7043):814–818, 2005.

[2] Balázs Adamcsek, Gergely Palla, Illés J. Farkas, Imre Derényi and Tamás Vicsek, “CFinder: Locating cliques and overlapping modules in biological networks,” Bioinformatics 22:1021-1023, 2006.

[3] Fergal Reid, Aaron McDaid and Neil Hurley, “Percolation computation in complex networks,” In ASONAM, 274–281, 2012.

[4] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian and Jeffrey Xu Yu, “Querying k-truss community in large and dynamic graphs,” In SIGMOD, 1311–1322, 2014.

[5] Michael R. Garey and David S. Johnson, “Computers and intractability; A guide to the theory of NP-Completeness”, 1990.

[6] Roberto Battiti and Marco Protasi, “Reactive local search for the maximum clique problem”, Algorithmica 29(4):610–637, 2001.

[7] Wayne Pullan and Holger H. Hoos, “Dynamic local search for the maximum clique problem,” Journal of Artificial Intelligence Research 25(1):159–185, 2006.

[8] Norishige Chiba and Takao Nishizeki, “Arboricity and subgraph listing algorithms,” SIAM Journal on Computing 14(1):210–223, 1985.

[9] Maksim Kitsak, Lazaros K. Gallos, Shlomo Havlin, Fredrik Liljeros, Lev Muchnik, H. Eugene Stanley and Hernán A. Makse, “Identification of influential spreaders in complex networks,” Nature Physics 6(11):888–893, 2010.

[10] Mark E. J. Newman, “Modularity and community structure in networks,” Proceedings of the National Academy of Sciences 103(23):8577–8582, 2006.

[11] Vladimir Batagelj and Matjaz Zaversnik, “An $O(m)$ Algorithm for Cores Decomposition of Networks,” CoRR cs.DS/0310049, 2003.

[12] Yong-Yeol Ahn, James P. Bagrow, and Sune Lehmann, “Link communities reveal multiscale complexity in networks,” Nature 466(7307):761–764, 2010.

[13] Wanyun Cui, Yanghua Xiao, Haixun Wang, Yiqi Lu and Wei Wang, “Online search of overlapping communities,” In SIGMOD, 277–288, 2013.

[14] Wanyun Cui, Yanghua Xiao, Haixun Wang and Wei Wang, “Local search of communities in large graphs,” In SIGMOD, 991–1002, 2014.

[15] Jonathan Cohen, “Trusses: Cohesive subgraphs for social network analysis,” National Security Agency Technical Report, 2008.

[16] Coen Bron and Joep Kerbosch, “Algorithm 457: finding all cliques of an undirected graph,” Communications of the ACM 16:575–577, 1973.

[17] Etsuji Tomita, Akira Tanaka and Haruhisa Takahashi, “The worst-case time complexity for generating all maximal cliques and computational experiments,” Theoretical Computer Science 363(1):28–42, 2006.

[18] David Eppstein and Darren Strash, “Listing all maximal cliques in large sparse real-world graphs,” In SEA, 364–375, 2011.

[19] Robert E. Tarjan, “Efficiency of a Good But Not Linear Set Union Algorithm,” Journal of the ACM, 22(2): 215–225, 1975.