大規模ネットワークに対する
実用的クラスタ発見アルゴリズムの開発

宇野 毅明
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: uno@nii.jp

抄録: データマイニングやネットワーク・言語学モデルの分野に，グラフの中からある種の特徴を持った頂点集合 (クラスタと呼ばれる) を抽出する問題がある．特徴の定義により多種の問題が存在するが，ここでは，頂点集合がクリーク，あるいはクリークに類するものを見つけ出す問題を考える．クリークとは，頂点集合 S で，任意の S の頂点間に枝が張られているようなものをいう．本稿では，巨大で疎なグラフの極大なクリーク，あるいはなるべく大きな，重み付きクリークに類する頂点集合を高速かつ多量に見つけるアルゴリズムを提案する．

A Practical Fast Algorithm for Finding Clusters of Huge Networks Takeaki Uno

National Institute or Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, JAPAN, e-mail: uno@nii.jp

Abstract: In the areas of data minings, computer networks, and computational linguistics, problems of finding vertex sets with some properties (called "clusters") of a given graph have been studied. There have been studied various of such problems. In this paper, we consider cliques as the clusters. A clique is a vertex set such that there is an edge between any two vertices of the vertex set. We propose a practical fast algorithm for finding many maximal cliques, and a fast heulistic algorithm for finding many quassi maximal cliques. These algorithms are fast enough for huge graphs.

1 はじめに

グラフ $G = (V, E)$ に対して，V の部分集合 S で，任意の 2 つの頂点 $v_1, v_2 \in V$ の間に枝がある，つまり $(v_1, v_2) \in E$ であるようなものを G の**クリーク**と呼ぶ．G が 2 つの V_1, V_2 により構成される 2 部グラフである場合，任意の 2 つの頂点 $v_1 \in V_1, v_2 \in V_2$ に対して $(v_1, v_2) \in E$ であるようなものを，G の**2 部クリーク**，あるいは単にクリークと呼ぶ．頂点集合 S が誘導するグラフが，クリークから数本枝を取り除いて得られるグラフであるとき，S を**準クリーク**と呼ぶことにする．クリークは，グラフ論の中でも基礎的なものであり，最適化問題を含め多くの研究がなされている．

クリークは，他の分野での応用も多い．データマイニングやネットワーク・計算言語学モデルの分野では，データセットの対応，コンピューターネットワーク，異種言語の意味

的な対応を表すグラフから，ある特徴を持った頂点集合，あるいは部分グラフ (クラスタと呼ばれる) を抽出し，対象となるデータを分析しようという試みが多く行われてきている．相澤 [5], 中渡瀬 [4], 村田 [2, 1, 3] の研究では，ある特徴を持った頂点集合として 2 部クリークを用い，2 部グラフ (あるいは N 部グラフ) の極大クリーク，あるいは準クリークを 1 つ，あるいは数多く見つけることにより分析を行っている．クリーク・準クリークは，相関の大きいデータの集合を表すなど，自然なモデル化に基づいて導き出されるものだからである．

ある種のクリーク・準クリークを見つける問題は，部分グラフ族に関する最適化問題とみなすことができる．頂点集合 V から実数への関数 f を，V がクリークに近ければ大きく，また求める特徴を備えているほど大きくなるような関数として定義すると，

最大化　$f(S)$
制約条件　$S \subseteq V$

という最適化問題の最適解，あるいは準最適解を見つける，という問題になる．この種の問題は，数理計画，特に組合せ最適化の分野や，離散アルゴリズムの分野で多くの研究がなされている．特にメタヒューリスティックは良く研究されており，多くの効率良いアルゴリズムが提案されている．しかし，その主眼はあくまでも最適解，あるいは精度の良い解を如何に見つけるかという点に注がれており，準最適解を大量に見つけるという観点からの研究はメインストリームではなかった．準最適解を大量に見つけるという観点に沿って，過去の組合せ最適化・離散アルゴリズムの技術がどのように生かされるかに注目し，実際に高速なアルゴリズムを構築したので報告する．これらの結果は，相澤・中渡瀬・村田らの研究に生かすべく，現在検討中である．

2　クリークの応用事例

ここで，相澤 [5], 中渡瀬 [4], 村田 [2, 1, 3] の研究を大まかに説明しよう．

村田 [2, 1, 3] の研究は，ウェブネットワークの中から，共通の事柄について書かれたページ群，つまり共通の事柄に関心を持つコミュニティーを機械的に探し出そうというものである．村田はウェブページのリンク構造に着目し，共通の事柄に関心を持つものは，共通のウェブページにリンクを張るだろうと考えた．つまり，ページ集合 S_1 と S_2 があり，S_1 のページから S_2 のページへ密にリンクが張られていれば，それは S_2 には共通の事柄が述べられており，S_1 はそれらに関心のあるコミュニティーである可能性が高い，というものである．

中渡瀬 [4] の研究は，いくつかの単語が並んでできている複数語に注目し，類義語を機械的に抽出しようというものである．単語の集合を 2 つ用意し，それを 2 つの頂点集合と考える．単語 A の次に単語 B をつなげた単語 (複数語) が存在する場合，頂点 A から B への枝を作る．こうして得られるグラフのクリークは，ある種の似た意味を持つ単語のグループになっていると考えられる，というものである．このようなグループをたくさん見つけることにより，類義語群を機械的に発見しようというものである．

相澤 [5] の研究は，いくつかの文書の関連性 2 つの言語の対約辞書や，論文データベースを用いて，論文と著者，言語 1 の言葉と言語 2 の言葉の対応を表す 2 部グラフを作り，そこから情報量が大きな（枝が密であることと相関のある）部分グラフを見つけることにより，類似性のあるグループを見つけようというものである．このようなグループを大量にみつけることにより，機械的に類似性のある論文を分類し，意味からの検索などが容易に行えるようにしようというものである．

以上の 3 つの論文ともに，ある種の，枝が密である部分グラフを見つける問題を解いているが，これといった，ある種完璧な目的関数が定義されているわけではなく，むしろモデルや目的関数はある種のあいまいさを含んでいる．このような状況では，目的関数を最適にするものを 1 つ見つけるよりは，目的関数をある程度大きくするものを大量に見つけ，その中からまたある種のフィルタリングの操作を行うなどして，より必要なものを選び出していくほうが，より現実的な分析を行うためには好都合であると想定される．

3 問題の定式化とアルゴリズム

ここで，本研究で扱った問題を詳しく定義する．一つは極大クリークを求める問題であり，もう一つは，頂点重み $w : V \to R$，枝重み $e : E \to R$ に対して定義される以下の最大化問題の準最適解を求める問題である．

最大化　$\sum_{v \in S} w(v) + \sum_{u,v \in S} w((u, v))$
制約条件　$S \subseteq V$

準最適解とは，目的関数値が最適解と近いもののことをさすが，この場合は，局所最適解などのメタヒューリスティック等の方法によって得られる解の総称とする．相澤・中渡瀬・村田の研究で用いられていたグラフは 2 部グラフであり，この節で扱っているグラフは一般のグラフであるので，ここの議論を直接に彼らの問題に適用することはできない．しかし，簡単な修正により 2 部グラフを扱うアルゴリズムに変更ができること，実際の計算オーダー，実験的な計算時間もあまり変化しないことがあるので，ここでは，説明が簡単である一般のグラフのバージョンを解説する．

極大クリークは，貪欲法 (局所探索) により，簡単に求めることができる．しかし，グラフが大きくなると，単純な方法で貪欲解を求めるのでは非常に時間がかかることが多い．例えば，以下のアルゴリズムを考えよう．入力するグラフは，隣接リストの形で渡すものとする．

Basic_Maximal_clique (V, E)
(BM1) $S = \emptyset$
(BM2) $V \setminus S$ の頂点で，S の全ての頂点に隣接するものがあれば S に加え，(BM2) に戻る
(BM3) S を出力し，終了する

このアルゴリズムは (BM2) の 1 回の実行に $O(|E|-|S|)$ 時間を消費し，全体では $O(|S|(|E|-|S|))$ の時間を消費する．ただしここで，S は出力された極大クリークとする．グラフが

疎であれば，$|E| - |S|^2 = \Theta(|E|)$ とみなしてよいので，このアルゴリズムの実行時間は $\Theta(|E||S|)$ となる．相澤・中渡瀬・村田の研究で用いられたグラフは，とても疎であり，$|S|$ はせいぜい 30 であった．しかし $|E|$ は 100,000 から 100,000,000 程度と大きく，よってこの方法で極大クリークを求めると，最近のパーソナルコンピューターでも 10 秒程度の時間がかかることになる．相澤の研究では，準最適解を何千何万と求めているので，この方法では時間がかかりすぎる．

さて，このような状況では，どのようにアルゴリズムの改善をするべきであろうか．有効な方法のひとつは，教科書的ではあるが，グラフが疎であることを利用して，各反復で前回の反復の計算結果を再利用することである．

今，S のすべての頂点に隣接する頂点の集合を $U(S)$ としよう．(BM2) で行っていることは，$U(S) = \emptyset$ であれば終了し，そうでなければ，$U(S)$ のある頂点 v を S に追加することである．つまり，$U(S)$ が保持されていれば，(BM2) は高速に実行できるのである．

$U(S)$ を保持するためには，S に頂点 v を追加したときに，$U(S)$ を $U(S \cup \{v\})$ に更新する，という作業を行わなくてはならない．

$$\begin{aligned} U(S \cup \{v\}) &= \{u|^{\forall} w \in S \cup \{v\}, (u,w) \in E\} \\ &= \{u|u \in U(S), (u,v) \in E\} \end{aligned}$$

であることから，その作業は $U(S)$ の頂点で v に隣接するものだけを取り出せばよい．この計算は $O(\Delta)$ 時間 (Δ はグラフの最大次数) であり，これによりアルゴリズムの計算時間は $O(|S|\Delta)$ まで小さくなる．相澤・村田・中渡瀬の例などのようにグラフが疎であれば，Δ はせいぜい 100 であるので，枝数が 10 万であれば，計算時間は 1/1000 ほどに短縮される．これならば 1000 個の極大クリークを見つけるにも 10 秒ほどしかかからない．

さて，次に重み w が与えられたとき，以下の最大化問題の準最適解を求める問題に対するアルゴリズムを考えよう．

最大化　$\sum_{v \in S} w(v) + \sum_{u,v \in S} w((u,v))$
制約条件　$S \subseteq V$

この問題は最大クリーク問題を特殊ケースとして含むので，NP-hard である．そこで，メタヒューリスティック解法の一つである局所探索によるアルゴリズムを構築した．クリーク，あるいはそれに類する問題に対して効果的であると言われている解法は他にもいくつかあるが，解を数多く生成する，という観点からは，局所最適解を数多く生成できる局所探索法，あるいは多スタート局所探索法が良いと考えた．

頂点集合 S に対して，その近傍集合 $N(S)$ を

$$N(S) = \{X|^{\exists} v, X \cup \{v\} = S \ \text{ or } \ X \setminus \{v\}\}$$

と定義し，$N(S)$ の要素を S の近傍と呼ぶ．そして，以下の局所探索アルゴリズムを考える．

Local_Search (V,E)
(LS1) $S = \emptyset$
(LS2) S の近傍 S' で，S より目的関数値が大きいものが存在したら，$S = S'$ とし，(LS2) に戻る
(LS3) S を出力し，終了する

研究の目的は,「なるべく大きなクリーク，あるいは準クリークを短時間で見つける」ということであったので，目的関数値は「S が大きなクリーク，あるいは準クリークであると大きくなる」関数にすべきである．また，アルゴリズムを高速にするためには，S の近傍の目的関数値が容易に計算できる必要がある．さらに欲を言えば，今後他の応用にも使えるよう，他の目的関数にも対応できるような形になっていると良い．以上を踏まえて，本研究では以下の関数を用いた．

目的関数　$f(S) = \sum_{v \in S} w_V(v) + \sum_{u,v \in S} w_E((v,u)) + g(|S|)$

ただし，w_V は頂点重み，w_E は枝重みを表す関数であり，$g(|S|)$ は S の大きさに関する任意の関数である．任意の頂点・枝に対する重みを 1 と設定すれば，$\sum_{v \in S} w_V(v)$ は $|S|$ と等しくなり，

$$c(S) = (|S| - 1)|S|/2 - \sum_{u,v \in S} w_E((v,u))$$

は，枝のない S の頂点対の数を表す．$-c(S)$ が大きいほど S はクリークに近くなるので，$g(|S|)$ をうまく設定すれば，なるべく大きく，クリークに近いものほど値が大きくなるように目的関数を設定できる．

枝や頂点に重要度がある場合など，目的により枝重み，頂点重みを変更して，他のモデルに対応することができる．また，各枝 e に容量 $c(e)$ が与えられているとき，枝 e の重みを $-2c(e)$，頂点 v の重みを v に接続する枝の容量の総和，$g(|S|) = 0$ と設定すると，目的関数 $f(S)$ は

$$\begin{aligned} f(S) &= \sum_{v \in S, (u,v) \in E} w_E((u,v)) - 2 \sum_{u,v \in S} w_E((u,v)) \\ &= \sum_{v \in S, u \notin S, (u,v) \in E} w_E((u,v)) \end{aligned}$$

となり，S のカットの容量を表すことになる．

さらに，これらの関数に対して，$g(|S|)$ を適当に設定することにより,「ある大きさ以上 (以下) のものを求めたい」といった要求にも対応できるようになる．

次に，頂点集合 S に対して近傍の中で目的関数値が大きいものを高速で見つける方法について解説する．頂点集合 S と頂点 v に対して，

$$f^+(S,v) = \alpha w_V(v) + \beta \sum_{u \in S} w_E((u,v))$$

とする．S の近傍 S' に対して，

$$f(S') - g(|S'|) = \begin{cases} f(S) - g(|S|) + f^+(S,v) & \text{if } S' = S \cup \{v\} \\ f(S) - g(|S|) - f^+(S,v) & \text{if } S' = S \setminus \{v\} \end{cases}$$

が成り立つ．S に属さない頂点 v で $f^+(S,v)$ を最大にするものを z^+，S に属する頂点 v で $f^+(S,v)$ を最小にするものを z^- とすると，S の近傍の中で目的関数値を最大にするものは，$S \cup \{z^+\}$ か $S \setminus \{z^-\}$ のどちらかである．よって，S に属さない頂点で $f^+(S,v)$ を最大にする頂点を求めるヒープ H^+ と，S に属する頂点で $f^+(S,v)$ を最小にする頂点を求めるヒープ H^- の 2 つがあれば，S の近傍で S より目的関数値が小さいものを求める操作は $\mathrm{O}(\log |V|)$ 時間で行える．

では次に，局所探索の 1 反復が終了し，解を S から S' に変更したとき，H^+ と H^- に対する変更がどの程度必要か見てみよう．

S' が S に頂点 w を加えて，あるいは取り除いて得られた，つまり $S' = S \cup \{w\}$ か $S' = S \setminus \{w\}$ であるとする．頂点 v が $(v,w) \notin E$ であれば，

$$\sum_{u \in S'} w_E((u,v)) = \sum_{u \in S} w_E((u,v))$$

であるので，

$$f^+(S',v) = f^+(S,v)$$

が成り立つ．また，$(v,w) \in E$ であれば，

$$\sum_{u \in S'} w_E((u,v)) = w_E(v,w) + \sum_{u \in S} w_E((u,v))$$

であるので，$S' = S \cup \{w\}$ ならば

$$f^+(S',v) = f^+(S,v) + w_E(v,w)$$

$S' = S \setminus \{w\}$ ならば

$$f^+(S',v) = f^+(S,v) - w_E(v,w)$$

が成り立つ．よって，$f^+(S,v) \neq f^+(S',v)$ となる頂点 v は必ず w に隣接していることがわかる．H^+ と H^- の更新は，これら w に接続する頂点と w 自身についてのみ行えばよいので，$\mathrm{O}((deg(w)+1)\log |V|)$ 時間で行える．ただし $deg(w)$ は w の次数である．

グラフの頂点数を 1000 万，最大次数を 100 としても，$(deg(w)+1)\log |V|$ は 2500 程度であるので，1 秒間に何万回かの反復ができると予測される．

4 計算実験

本節では，上記の 2 つのアルゴリズムを実際に計算機上に実装し，計算実験を行った結果を報告する．両者ともに C を用いて実装し，Pentium III 500MHz の PC で実験した．OS は Linux，コンパイラは gcc である．今回の実験では，2 部クリークではなく，通常の

クリークを求めるアルゴリズムを実装した．2 部グラフ $G = (V = V_1 \cup V_2, E)$ の場合，V_1 自体や，$v \in V_1$ と v に隣接する頂点全てからなる集合が，極大クリークとなり，このようなものに対しては計算時間や反復数が異常に大きくなる．これらを排除する必要があるか，また排除する基準が不明瞭であるため，今回は純粋にアルゴリズム自体の性能を検証する目的からも，一般のクリークで計算を行った．

実験に用いたグラフはランダムグラフである．2 部グラフではなく，一般のグラフである．準クリークを求める局所探索には，頂点数が 10 万，次数がおよそ 100 のグラフと，頂点数が 100 万，次数がおよそ 10 のグラフの 2 種類について実験した．極大クリークを求めるアルゴリズムについては，頂点数が 1 万，次数がおよそ 1000 のグラフと，頂点数が 1000，次数がおよそ 300 のグラフの 2 種類について実験した．これは，あまり大規模で疎なランダムグラフだと，極大クリークの大きさが 2,3 程度になってしまい，実験の意味がなくなるからである．一回の実行で局所的な情報しか参照しないので，グラフは 1 つに固定し，各頂点について，その頂点のみからなる頂点集合を初期解として，実行した．つまり，アルゴリズムを頂点数回実行した総実行時間と総反復数を計測した．結果は以下の通りである．

極大クリーク

頂点数	枝数	平均次数	総計算時間	総反復数	1 回の 平均反復数	10000 反復の 平均計算時間
1 万	1000 万	1000	29 sec	46002	4.6	6.3 sec
1000	30 万	300	0.78 sec	6111	6.1 sec	

準クリーク局所探索

頂点数	枝数	平均次数	総計算時間	総反復数	1 回の 平均反復数	10000 反復の 平均計算時間
10 万	1000 万	100	462 sec	2,387,413	23	1.9 sec
100 万	1000 万	10	269 sec	7,987,721	8.0	0.34sec

どの例においても，1 反復の計算時間は 1/1000 秒以下である．反復回数もそれほど大きくならない (せいぜい 30 程度) ので，大規模で疎なグラフに対しても十分高速な計算ができることが実証された．

また，今回の実験では，最適解との目的関数値乖離，つまり誤差については評価を行わなかった．これは，そもそもこのような巨大なグラフでは最適解を求めるのは不可能であること，また，研究の出発点が「巨大なグラフでどのようなことができるか」というものであったことによる．

相澤 [5] では，頂点数が 1 万，枝数が 10 万程度のグラフで，準クリークを 1000-10000 個程度求めるのに 1 時間程度の時間をかけていた．これと比べれば，似たような準クリークを求めるのにかかる時間は 1 分以内に短縮されると予想され，成果が期待できる．

5　まとめ

本稿では，大規模かつ疎なグラフの極大クリークと，頂点重みと枝重みと頂点数の関数をの和をなるべく小さくする頂点集合を求める高速な局所探索法のアルゴリズムを提案した．計算実験により，枝数が 100 万を越えるような巨大な疎グラフでも 1 反復の計算時間は 1/1000 程度であることが検証された．

この問題はデータマイニング，計算言語学などの分野の問題に現れるなるべく大きなクリーク，あるいはクリークに近い頂点集合を大量に求める問題に対して応用できるので，今後は実際のシステムとして，このアルゴリズムを応用することを目指す．

6　謝辞

本稿の作成に協力いただいた相澤・村田両氏に感謝いたします．

参考文献

[1] 村田 剛志, “参照の共起性に基づく Web コミュニティの発見,” 人工知能学会誌, Vol.16, No.3, pp. 316-323, 2001.

[2] Tsuyoshi Murata, “Discovery of Web Communities Based on the Co-occurrence of References,” Proceedings of the Third International Conference on Discovery Science (DS2000), Lecture Notes in Artificial Intelligence 1967, pp.65-75, Springer, 2000.

[3] Tsuyoshi Murata, “Finding Related Web Pages Based on Connectivity Information from a Search Engine,” Poster Proceedings of the Tenth International World Wide Web Conference (WWW10), pp.18-19, 2001.

[4] 中渡瀬 秀一, “複数語からの類義語抽出法,” 情報処理学会研究会報告, 情報学基礎 FI-66-6, pp. 39-46, 2002.

[5] Akiko Aizawa, A Method of Cluster-Based Indexing of Textual Data. Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002), pp. 1-7, 2002.