グラフ集約に基づく高速な最大k-plex探索

真次　彰平†　　藤原　靖宏††　　塩川　浩昭†††

† 筑波大学大学院理工情報生命学術院　〒 305-8571 茨城県つくば市天王台 1-1-1
†† 日本電信電話株式会社　〒 243-0198 神奈川県厚木市森の里若宮 3-1
††† 筑波大学計算科学研究センター　〒 305-8571 茨城県つくば市天王台 1-1-1
E-mail: †matsugu@kde.cs.tsukuba.ac.jp, ††yasuhiro.fujiwara.kh@hco.ntt.co.jp, †††shiokawa@cs.tsukuba.ac.jp

あらまし　*k*-plex はクリークを一般化した密部分グラフのモデルであり，グラフ中の最大の *k*-plex を求める MPS (Maximum *k*-Plex Search) 問題は実世界の幅広いソーシャルネットワーク分析に応用される．しかしながら，既存の MPS 問題の解法は探索候補を枝刈りする過程で全てのノードを繰り返し走査する必要るため低速である．そこで本稿では隣接するノードをマージすることで探索空間を大幅に削減する高速な解法を提案する．実データを用いた実験により提案手法におけるマージ戦略が効率的な探索を実現し，最先端の既存手法に対して最大約 10 倍高速であることを確認した．

キーワード　k-plex，グラフ

1　はじめに

ソーシャルネットワークにおけるユーザの行動分析に用いられる技術の一つに最大密部分グラフ抽出がある．最大密部分グラフ抽出はグラフ中の密度の高い部分グラフのうち最も多くのノードからなる部分グラフを発見する技術である．最大密部分グラフ抽出における最も代表的なアプローチである最大クリーク探索は幅広い分野で応用されているが，次に示す二つの問題点がある．一つ目は実世界のソーシャルネットワークにはスケールフリー性があるため，コミュニティが多様な密度を持つ点である．二つ目は最大クリーク探索の既存手法は実際のコミュニティに対して非常に密で小さい部分グラフを同定してしまう点である．以上のことから，既存の最大クリーク探索手法は実世界のコミュニティを正確に発見することが困難である．

この問題を解決するため近年最大 *k*-plex 探索 [1] が注目を集めている [2,3]．*k*-plex はクリークを一般化した密部分グラフのモデルであり，n 個のノードからなる部分グラフの各ノードがその部分グラフ中の $n-k$ 個以上のノードと隣接するような部分グラフのことである [4]．*k*-plex は密度と大きさのバランスに優れており，最大 *k*-plex 探索は実世界の大きな密部分グラフを抽出できるため，次のようなアプリケーションに応用される．

犯罪ネットワーク分析　最大 *k*-plex 探索はソーシャルネットワークにおけるテロ組織，犯罪組織，カルト宗教のような警戒すべきコミュニティの発見に有効である．Menon らはパブリックデータセットから 9.11 アメリカ同時多発テロに関与したテロリストを検出するために複数のモデルを比較し，*k*-plex がクリークを含むいくつかのモデルと比較して最も多くのテロリストを検出できることを明らかにした [5]．また Balasundaram は犯罪ネットワークにおける麻薬取引やマネーロンダリングの検出において最大 *k*-plex 探索が最大クリーク探索より優れていることを報告している [6].

パーソナライズドソーシャルサーチ　パーソナライズドソーシャルサーチはソーシャルネットワーク上の友人関係によるコミュニティを用いて Web 検索結果をランキングする手法であり [7,8]，最大 *k*-plex 探索が用いられている．例えば Danezis らは友人による最大のコミュニティで共有される検索嗜好の事前計算に最大 *k*-plex 探索を用いた [9]．具体的にはまずソーシャルネットワークから最大 *k*-plex を抽出し，その *k*-plex における嗜好をベイズモデルにより推論する．*k*-plex が多様な密度のコミュニティを扱うことができるため，最大 *k*-plex 探索を用いたパーソナライズドソーシャルサーチは他の密部分グラフのモデルと比較して有効であることが報告されている．

グラフニューラルネットワーク (GNN)　最大 *k*-plex 探索は近年ソーシャルネットワーク解析の主要なアプローチの一つとなっている GNN の性能向上にも寄与している．GNN ではノードのコンテキストを把握するためにグラフのプーリングが重要であり，古典的にはクリークを用いたグラフプーリング手法が用いられてきた [10]．Bianchi らは最大 *k*-plex 探索を基にしたグラフプーリング手法を提案し，クリークによる手法と比較して畳み込み層の数を削減した．その結果として提案手法は GNN 上の課題である過平滑化問題を緩和することに成功し，高精度な GNN を実現した [11].

最大 *k*-plex 探索は他にも口コミによる広告戦略の策定 [12, 13] や感染症の拡大分析 [14, 15] などにも応用されている．紙面の都合上，詳細は省略する．

最大 *k*-plex 探索は多くのアプリケーションに有効であるが，グラフ中の最大 *k*-plex を見つける問題は NP-Hard であることが知られている [1]．具体的にはグラフ G 中のノード集合を V_G とすると，素朴な最大 *k*-plex 探索アルゴリズムは $O(2^{|V_G|}|V_G|)$ の時間計算量を要する．しかしながら，近年のソーシャルネットワークは巨大であるため [16]，最大 *k*-plex の探索は数日程度の計算時間を要する．

1.1 既存研究と本研究の位置付け

最大 *k*-plex 探索の高速化にはいくつかのアプローチがある．古典的なアプローチとしては *greedy branching* 法 [17]，*branch-and-cut* 法 [1]，および *branch-and-search* 法 [18] が挙げられる．これらの手法な素朴な最大 *k*-plex 探索と比較して高速だが，枝刈りに用いる上限値の計算が緩いため探索する必要のあるノード数が依然として多い．そのためこれらのアプローチでは実用的な計算時間で最大 *k*-plex を出力できない．

Gao らは近年 *branch-and-bound* (BnB) 法を提案した．その基本となるアイデアはグラフ削減アルゴリズムを用いて最大 *k*-plex になり得ないノードの無駄な探索を避けることにある．このグラフ削減アルゴリズムに基づき，BnB 法は冗長な探索を省略しつつ最大 *k*-plex を見つけられることが保証される．BnB 法は古典的なアプローチと比較して高速だが，その速度は未だ十分でないため BnB 法を高速化した手法がこれまでに提案されている [16, 19, 20]．しかしながら，既存の最先端の手法である KpLeX 法 [16] はグラフを削減するためにすべてのノードを繰り返し探索する必要があり，$O(|V_G|^2)$ の時間計算量を要するため，巨大なソーシャルネットワークに対して高速に応答できない．そのため，大規模なソーシャルネットワークに対する最大 *k*-plex 探索の更なる高速化は重要な研究課題である．

1.2 本研究の貢献

本稿ではソーシャルネットワークに対する高速な最大 *k*-plex 探索手法である，*Brunch-and-Merge* (BnM) 法を提案する．1.1 節で述べた近年の既存手法はグラフ削減アルゴリズムを実行するために $O(|V_G|^2)$ の時間計算量を要する．それに対して BnM 法はソーシャルネットワークが持つスケールフリー性に着目することで，最大 *k*-plex に含まれないノードを $O(|V_G|)$ 時間で列挙することができる．これにより BnM 法は効率的にグラフサイズを削減し，高速な計算を実現する．

上述したアイデアに基づき，BnM 法を次の 3 つのステップで設計する．(1) 最大 *k*-plex に同時に含まることのないノード集合を *safe-to-merge* ノードとして定義する（3.2 節）．(2) *safe-to-merge* ノードを効率よく探索してグラフサイズを削減する *node-merging* 法を提案する（3.3 節）．(3) *node-merging* 法を用いて最大 *k*-plex を高速に探索する BnM 法を設計する（3.4 節）. さらには，BnM 法が実世界のグラフが持つスケールフリー性を利用することで，*node-merging* 法によるグラフサイズの削減が効果的に機能することを理論的に示す．

本研究の貢献は以下の通りである．

- **効率性:** BnM 法はここ数年で提案された最大 *k*-plex 探索手法より高速である（4.1 節）．また BnM 法は最先端の既存手法と比較して時間計算量が小さいことを示す（定理 1）．
- **正確性:** BnM 法は探索の効率化のためにグラフサイズを非常に小さくするが，必ず正確な最大 *k*-plex を見つけることができる（定理 2）．
- **スケーラビリティ:** BnM 法は既存手法と比較して優れたスケーラビリティを持つ（4.3 節）．我々の実験において，BnM 法はノード数に対しておよそ線形なスケーラビリティを示した．

BnM 法は我々の知る限り大規模なソーシャルネットワークに対して効率的に最大 *k*-plex を出力できる唯一の手法であり，最先端の既存手法と比較して高速に最大 *k*-plex を探索できる．例えば 110 万ノードからなるソーシャルネットワークに対して，最先端の既存手法である KpLeX 法 [16] が最大 *k*-plex の出力に 1 分以上掛かる一方で，BnM 法は同じ解を 16 秒以内に求めることが可能であり，BnM 法は実世界の多様なアプリケーションの品質を向上させることができる．

2 事 前 準 備

まずはじめに本稿で用いる記号を定義する．本稿では重みのない連結な無向グラフ $G = (V_G, E_G)$ を考える．ここで V_G, E_G はそれぞれ G のノード集合およびエッジ集合である．ノード u の G における次数を $d_G(u)$，部分ノード集合 $V_S \subseteq V_G$ による G の誘導部分グラフを $G[V_S]$ と書く．またグラフ G からノード v および v に接続するエッジを除いたグラフを G_{-v} と書く．すなわち，$G_{-v} = G[V_G \backslash \{v\}]$ である．

最大 *k*-plex 探索はグラフ G 中の最大の *k*-plex を見つける問題である．ここで *k*-plex はクリークを一般化した密部分グラフのモデルであり，次のように定義される [21].

定義 1 (*k*-plex) グラフ G，および正の整数 k が与えられたとき，$G[V_S] \subseteq G$ が V_S 中の全てのノード $v \in V_S$ に対して $d_{G[V_S]}(v) \geqq |V_S| - k$ を満たすとき，$G[V_S]$ は *k*-plex である．

$G[V_S]$ が *k*-plex であれば, 各ノード $v \in V_S$ は少なくとも $|V_S|-k$ 個のノードと隣接する．定義 1 を用いて，最大 *k*-plex 探索は以下のように定義される。

問題定義 1 (最大 *k*-plex 探索) グラフ G，および正の整数 k が与えられたとき，最大 *k*-plex 探索は G 中の最大の *k*-plex，すなわち $|V_S|$ が最大となる $G[V_S] \subseteq G$ を見つける問題である．

$k = 1$ のとき，問題定義 1 は明らかに最大クリーク探索と等価である [22]．最大 *k*-plex 探索は NP 困難 [21, 23, 24] であるため大規模ソーシャルネットワークに対する効率的な探索手法の開発は重要である．

3 Branch-and-Mergeh (BnM) 法

本節では提案手法 BnM 法について説明する．最初に 3.1 節に BnM 法を構成する基本的なアイデアを示し，その詳細を以降の節にて説明する．紙面の都合上，補題の証明は省略する．

3.1 基本アイデア

1.1 節で示した既存手法は最大 *k*-plex に含まれないノードを削除するグラフ削減アルゴリズムを繰り返し適用することで探索空間を小さくする．しかしながら，それらのグラフ削減アルゴリズムは全てのノードを繰り返し参照する必要があるため計算コストが高い．それに対し BnM 法は最大 *k*-plex に含まれないノードを効率的に探索し，グラフサイズを小さくする．そのために，本稿ではまず共に最大 *k*-plex に含まれることのないノード集合に着目し，それらを *safe-to-merge* ノードとして定義する（3.2 節）．次に safe-to-merge ノードをグラフから効率的に計算および削除する *node-merging* 法を説明する（3.3 節）．最後に探索空間を大幅に削減することで最大 *k*-plex を高速に発見する BnM 法を提案する（3.4 節）．

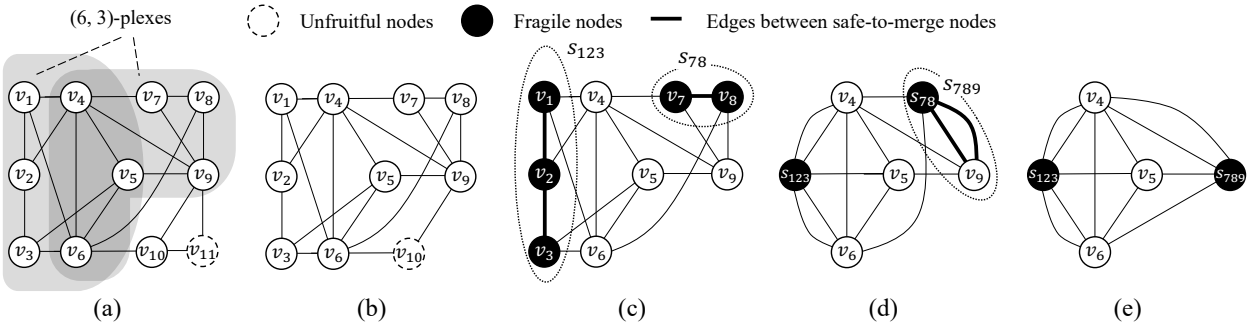


図 1: node-merging 法の動作例 （$\alpha = 6$，$k = 3$.） s_{123}, s_{78}，および s_{789} はスーパーノードを示す.

上記のアイデアを基に提案するアルゴリズムは既存手法に対して二つの優位性を持つ.（1）BnM 法は最大 *k*-plex を短時間で出力可能である．全ノードを繰り返し参照する既存のグラフ削減アルゴリズムとは異なり，BnM 法は最大 *k*-plex に含まれないノードに隣接するノードのみを走査することでグラフサイズを縮小する．ここでソーシャルネットワークはスケールフリー性を持つため [25]，次数の小さな最大 *k*-plex に含まれないノードが多く存在する．この特性を活用し，BnM 法は実世界のグラフにおいて効果的にグラフサイズを削減可能である.（2）BnM 法は常に正確な最大 *k*-plex を出力する．これは BnM 法がノードをマージする際に最大 *k*-plex に含まれないノードのみを削除するためである．そのため BnM 法は探索の品質を落とすことなく最大 *k*-plex を発見する．その結果として，BnM 法は効率的に正確な最大 *k*-plex を探索できる.

3.2 Safe-to-merge ノード

BnM 法は共に最大 *k*-plex に含まれることの無いようなノードを削除する．このようなノードの集合を safe-to-merge ノードと呼ぶ．safe-to-merge ノードを定義するために，先に定義 2，定義 3，および定義 4 を導入する.

定義 2 ((α, k)-plex) 部分グラフ $G[V_S] \subseteq G$ の大きさ $|V_S|$ が α であるとき，$G[V_S]$ は G の (α, k)-plex である.

(α, k)-plex は大きさ α の *k*-plex である．ここで (α, k)-plex は α 毎に複数存在し得ることに注意したい．例えば図 1 (a) は二つの (6, 3)-plex をもつ．具体的には $G[\{v_1, v_2, v_3, v_4, v_5, v_6\}]$ および $G[\{v_4, v_5, v_6, v_7, v_8, v_9\}]$ が定義 2 を満たす (6, 3)-plex である.

定義 2 に基づき *unfruitful* ノードを定義する．これは以下のように任意の (α, k)-plex に含まれないノード集合である.

定義 3 (Unfruitful ノード) α と k に対して, unfruitful ノード U_G は $U_G = \{v \in V_G \mid d_G(v) < \alpha - k\}$. である.

定義 3 の通り unfruitful ノード U_G は次数が $\alpha - k$ より小さなノードの集合である．定義 2 および定義 3 より，U_G は任意の (α, k)-plex に含まれない．例えば図 1 (a) において，unfruitful ノードは $d_G(v_{11}) = 2 < \alpha - k = 3$ より $U_G = \{v_{11}\}$ である.

unfruitful ノードは任意の (α, k)-plex に含まれないため，グラフ G から削除できる．しかし，ノードの削除によって隣接するノードの次数が下がるため，新しく unfruitful ノードが発生する．そこで以下のように G 中の全ての unfruitful ノードを削除したグラフを収束グラフとして定義する.

定義 4 (収束グラフ) α，および k に対して，$G^{(i)}$ を i 回目の削除後に残ったグラフとする．すなわち

$$G^{(i)} = \begin{cases} G[V_G \backslash U_G] & (i = 1) \\ G[V_{G^{(i-1)}} \backslash U_{G^{(i-1)}}] & (i > 1). \end{cases}$$

とすると，収束グラフ $\hat{G}$ を $U_{G^{(i)}} = \emptyset$ を満たす $\hat{G} = G[V_{G^{(i)}} \backslash U_{G^{(i)}}]$ である.

定義 4 に示した通り収束グラフ $\hat{G}$ には unfruitful ノードが存在しない. 例えば図 1 (a) において (6, 3)-plex に対する収束グラフを作ることを考える．unfruitful ノード U_G は $U_G = \{v_{11}\}$ であるため，U_G を削除した後には図 1 (b) のような $G^{(1)}$ を得る．定義 3 より $G^{(1)}$ はさらに unfruitful ノード $U_{G^{(1)}} = \{v_{10}\}$ を持つ．そのため $G^{(2)}$ は図 1 (c) のようになる．$G^{(2)}$ は全てのノードが次数 $\alpha - k = 3$ 以下であるため unfruitful ノードが存在しない．したがって，$G^{(2)}$ は G の収束グラフ，すなわち $\hat{G} = G^{(2)}$ である.

次に定義 2，定義 3，および定義 4 を用いて safe-to-merge ノードを定義する.

定義 5 (Safe-to-Merge Nodes) 隣接する二つのノード $u, v \in \hat{G}$ が与えられたとき，$u \in U_{\hat{G}_{-v}}$ かつ $v \in U_{\hat{G}_{-u}}$ であり，かつそのときに限り，u と v は safe-to-merge ノードである．また u と v が safe-to-merge ノードであるとき，$u \leftrightarrow_{\hat{G}} v$ と表す.

定義 5 は収束グラフ $\hat{G}$ からノード v を削除した後にノード u が unfruitful ノードになることと，$u \leftrightarrow_{hatG} v$ が成り立つことが同値であることを示す（逆も同様である）．図 1 (c) において太線は両ノードが safe-to-merge ノードであることを示す．例えば (6, 3)-plex において隣り合う二ノード v_7 と v_8 は safe-to-merge ノードである．これは G_{-v_8} においてノード v_7 の次数が 2 となるため $\alpha - k = 3$ より小さくなり，ノード v_8 も G_{-v_7} において次数が 3 より小さくなるためである．定義 5 について次の補題が成り立つ.

補題 1 $u \leftrightarrow_{\hat{G}} v$ であるノード u, v について，v がある (α, k)-plex に含まれないならば，u もその (α, k)-plex に含まれない.

補題 1 は safe-to-merge ノードである二つのノードのうち，一方のノードがある (α, k)-plex に含まれない場合，もう一方のノードもその (α, k)-plex に含まれないことを示している．逆に，一方のノードがある (α, k)-plex に属する場合，もう一方のノードも必ずその (α, k)-plex に属している．よって，(α, k)-plex に含まれないノードがあれば，それらの safe-to-merge ノードを削除

することで効率的にグラフサイズを削減できる．しかしながら，定義 5 に基づき safe-to-merge ノードを特定するためには全てのノード対について計算する必要がある．これは $O(|V_{\hat{G}}|^2)$ 時間の計算を要するため効率的でない．

3.3 Node-merging 法

本節では収束グラフから効率的に safe-to-merge ノードを見つける *node-merging* 法を説明する．まず背景となる性質について述べた後（3.3.1 節），その詳細について示す（3.3.2 節）．

3.3.1 Fragile ノード

まず safe-to-merge ノードを効率的に探索するために重要となる *fragile* ノードについて説明する．

定義 6 (Fragile ノード) 収束グラフ $\hat{G}$ が与えられたとき，fragile ノード $F_{\hat{G}}$ を以下のように定義する．

$$F_{\hat{G}} = \{v \in V_{\hat{G}} \mid d_{\hat{G}}(v) = \alpha - k\}.$$

例えば図 1 (c) において，fragile ノードは次数が $\alpha - k = 3$ であるノードの集合であるため，$F_{\hat{G}} = \{v_1, v_2, v_3, v_7, v_8\}$ となる．ノードが削除されるとその隣接ノードの次数が小さくなるため，fragile ノードは自身に隣接するノードが削除されたとき unfruitful ノードに変わる．例えば図 1 (c) 中の v_7 は v_8 を $\hat{G}$ から削除した後 unfruitful ノードになる．この逆もまた成り立つため，定義 5 より v_7 と v_8 は safe-to-merge ノードであることがわかる．一般的にこのような性質は次の補題として示すことができる．

補題 2 $G[F_{\hat{G}}]$ を fragile ノードから構成される誘導部分グラフとする．$G[F_{\hat{G}}]$ にパス $v_1 \to v_2 \to ... \to v_N$ が存在するならば，$v_i \leftrightarrow_{\hat{G}} v_{i+1}$ が $i \in \{1, 2, ..., N-1\}$ について成り立つ．

補題 2 は fragile ノードから構成される誘導部分グラフ中の連結なノード集合は safe-to-merge ノードであることを示している．例えば図 1 (c) においてノード v_1，v_2，および v_3 は $F_{\hat{G}}$ に属している．またパス $v_1 \to v_2 \to v_3$ が $G[F_{\hat{G}}]$ に存在するため，$v_1 \leftrightarrow_{\hat{G}} v_2$ かつ $v_2 \leftrightarrow_{\hat{G}} v_3$ が成り立つ．fragile ノードを用いるとこのようなパスは $O(|V_{\hat{G}}|)$ 時間で自明に探索できる．したがって補題 2 に基づき効率的に safe-to-merge ノードを見つけることができる．

3.3.2 Node-merging 法のアルゴリズム

safe-to-merge ノードを効率的に探索する手法である node-merging 法のアルゴリズムを説明する．node-merging 法はまず補題 2 に基づき safe-to-merge ノードを列挙する．次に補題 1 を用いてそれらのノードを単一のスーパーノードとしてマージすることでグラフサイズを削減する．node-merging 法のアルゴリズムを説明するために，先にスーパーノードを定義する．

定義 7 (スーパーノード) グラフ $\hat{G}$，および連結な safe-to-merge ノード $M \in G[F_{\hat{G}}]$ の集合が与えられたとき，スーパーノード s を M をマージした後のノードとする．ここで，s は $\{v \in V_{\hat{G}} \mid \forall u \in M, (u, v) \in E_{\hat{G}}\}$ に属するノードと隣接する．また s の次数は $d_{\hat{G}}(s) = \alpha - k$ とし，スーパーノードの大きさ $|s|$ を $|s| = |M|$ と定義する．M をマージすることにより，M のノードおよびそれらに接続するエッジは $\hat{G}$ から削除される．

Algorithm 1 Node-merging 法

Input: 収束グラフ $\hat{G}$;
Output: safe-to-merge ノードをマージしたグラフ G^*;

```
1:  procedure MERGE (Ĝ, α, k):
2:      G* ← Ĝ;
3:      F_G* ← {v ∈ V_G* | d_G*(v) = α − k};
4:      M ← G[F_G*] 中の連結成分;
5:      while M ≠ ∅ do
6:          スーパーノード s を V_G* に追加;
7:          for each v ∈ M do
8:              for each u ∈ N(v)\M do
9:                  エッジ (u, s) を E_G* に追加;
10:             V_G* から v を削除する;
11:         F_G* ← F_G* ∪ {s}\M;
12:         M ← ∅;
13:         for each v ∈ N_G*(s) do
14:             if v ↔_G* s then
15:                 M ← M ∪ {v};
16:         if M = ∅ then
17:             M ← G[F_G*] 中の連結成分;
18:     return G*;
```

スーパーノードは safe-to-merge ノードの集合を単一のノードとして表現する特別なノードである．またマージされる前のノードは全て fragile ノードであるため，スーパーノードの次数は常に $\alpha - k$ とする．さらには M に接続する各エッジを s と接続しなおすため，同一のエッジが複数本作られることがある．例えば図 1 (d) において $\{v_1, v_2, v_3\}$ を s_{123} にマージするとき，エッジ (v_1, v_4) および (v_2, v_4) はどちらも (s_{123}, v_4) に変換される．

Algorithm 1 に node-merging 法のアルゴリズムを示す．補題 2 よりノード $u, v \in F_{\hat{G}}$ が $G[F_{\hat{G}}]$ で連結であるとき，u, v は safe-to-merge ノードであり，Algorithm 1 はこの性質を利用する．Algorithm 1 はまず fragile ノード F_{G^*} を計算する（3 行目）．次に補題 2 に基づき $G[F_{G^*}]$ の中から連結なノード集合 M を深さ優先探索により探索する（4 行目）．その後，各連結成分 M について定義 7 に従い M を対応するスーパーノード s に置き換え（6–10 行目），s は常に fragile ノードであるため s を F_{G^*} に追加する（11 行目）．ここで，$N_{G^*}(u) = \{v \in V_{G^*} \mid (u, v) \in E_{G^*}\}$ を u の G^* における隣接ノードの集合とし，Algorithm 1 は $N_{G^*}(s)$ を探索することで連結成分 M を更新する．N_{G^*} がスーパーノード s と隣接する safe-to-merge ノードを持つならば，それらのノードを M に追加する（13–15 行目）．またそうでないならば，次の連結成分の探索へ移行する（16–17 行目）．safe-to-merge ノードが無くなったとき，Algorithm 1 は終了する（5–17 行目）．

図 1 (c)，(d)，および (e) は Algorithm 1 の実行例である．図 1 (c) に示す収束グラフ $\hat{G}$ は，$F_{\hat{G}} = \{v_1, v_2, v_3, v_7, v_8\}$ を持つ．このとき $\{v_7, v_8\}$ をマージすることにより，Algorithm 1 はスーパーノード s_{78} を図 1 (d) のように追加する．定義 7 に基づき，エッジ (v_7, v_4)，(v_7, v_9)，(v_8, v_6)，および (v_8, v_9) はそれぞれ (s_{78}, v_4)，(s_{78}, v_9)，(s_{78}, v_6)，および (s_{78}, v_9) に置き換えられる．同様に $\{v_1, v_2, v_3\}$ は s_{123} へとマージされる．ここで図 1 (d) において互いに隣接する s_{78} と v_9 はどちらも safe-to-merge ノードであるため，これらのノードを図 1 (e) のように s_{789} としてマージする．

Algorithm 2 提案手法: Branch-and-Merge (BnM) 法

Input: グラフ G, 正の整数 k;
Output: 最大の k-plex G_{max};

```
1:  Ĝ ← G, G_max ← ∅, α ← k + 1;
2:  while true do
3:      Ĝ ← Ĝ の α および k における収束グラフ;
4:      G_curr ← BRANCH (Ĝ, |V_Ĝ|, α, k)
5:      if G_curr = ∅ then
6:          return G_max;
7:      Ĝ ← G[V_Ĝ \ F_Ĝ];
8:      G_max ← G_curr;
9:      α ← α + 1
10:
11: procedure BRANCH (Ĝ, n, α, k)
12:     if n = α then
13:         return Ĝ;
14:     if n < α then
15:         return ∅;
16:     G* ←MERGE (Ĝ, α, k);
17:     G_curr ← ∅;
18:     for each v ∈ V_G* do
19:         if v is a supernode then
20:             n ← n − |v|;
21:         else
22:             n ← n − 1;
23:         G_curr ← BRANCH (G*_{−v}, n, α, k);
24:         if G_curr ≠ ∅ then
25:             return G_curr;
26:     return ∅;
```

最後に Algorithm 1 を用いて $\hat{G}$ 中の全ての safe-to-merge ノードをマージする時間計算量を示す.

補題 3 Algorithm 1 の時間計算量は $O(|F_{\hat{G}}| \cdot (\alpha - k))$ である.

3.1 節で述べた通り，実世界のソーシャルネットワークはスケールフリー性により次数の小さなノードを多く持つ．スケールフリー性とは γ をデータの次数分布の偏りを表す正の定数としたとき，次数が d であるノードの数がおおよそ $|V_G|d^{-\gamma}$ に比例するという性質である [26, 27]．fragile ノードはちょうど $d = \alpha - k$ 個のノードと隣接するため，実用上は $O(|F_{\hat{G}}|)$ は $O(|F_{\hat{G}}|) \approx O(|V_{\hat{G}}|(\alpha - k)^{-\gamma})$ と近似できる．したがって Algorithm 1 は $O(|F_{\hat{G}}| \cdot (\alpha - k)) \approx O(|V_{\hat{G}}|/(\alpha - k)^{(\gamma-1)})$ 時間でグラフサイズを削減できる．つまり Algorithm 1 は実世界のソーシャルネットワークにおいてノード数に対して劣線形時間で応答する．それに対して，1.1 節で述べた通り最先端の既存手法は $O(|V_G|^2)$ 時間でグラフ削減を行う．そのため補題 3 は Algorithm 1 は既存の最大 k-plex 探索アルゴリズムと比較して効率的にグラフを小さくできることを示している．

3.4 BnM 法

BnM 法のアルゴリズムを Algorithm 2 に示す．BnM 法は繰り返し α を増加させながら最大 k-plex を見つけるまで (α, k)-plex を探索し続ける（2–9 行目）．まず任意の k 個のノードは自明に (k, k)-plex であるため，$\alpha = k + 1$ から探索を始める（1 行目）．各イテレーションについて BnM 法は $\hat{G}$ から (α, k)-plex を見つけるために BRANCH 関数を呼び出す（4 行目）．BRANCH 関数は削減（16 行目）と分枝（18–25 行目）の 2 つのステップからなる．削減のステップでは Algorithm 1 に示した MERGE 関数を用いてグラフサイズを削減する．分枝のステップでは再帰的に BRANCH 関数を呼び出し，V_{G^*} 中のノードを一つずつ削除する．再帰計算の後に $\hat{G}$ のノード数が α であるならば，BRANCH 関数は $\hat{G}$ を (α, k)-plex として返す．そうでないならば，BRANCH 関数は何も返さない（12–15 行目）．最後に BRANCH 関数が何も返さなければ，BnM 法は $(\alpha - 1, k)$-plex を最大 k-plex として出力する (5–6 行目).

理論解析：BnM 法の効率性と正確性について理論的に解析する．本節では τ を Algorithm 2 において BRANCH 関数を呼び出した回数とする．

定理 1 BnM 法の時間計算量は $O(\tau(|V_{\hat{G}}| + |F_{\hat{G}}|(\alpha - k)))$ である.

証明 Algorithm 2 の 16 行目に示した通り，BRANCH 関数は MERGE 関数を呼び出しており，補題 3 よりその時間計算量は $O(|F_{\hat{G}}|(\alpha - k))$ である．さらに Algorithm 2 の 18–25 行目に示した通り，BnM 法 (α, k)-plex を求めるために $|V_{\hat{G}}|$ 回のイテレーションを実行する．よって BnM 法の時間計算量は $O(\tau(|V_{\hat{G}}| + |F_{\hat{G}}|(\alpha - k)))$ となる． □

最先端の既存手法はグラフ削減に $O(|V_G|^2)$ 時間を要するため，最大 k-plex の探索には $O(\tau|V_G|^2)$ 時間を要する．その一方で定理 1 に示した通り BnM 法は時間計算量を大幅に削減する．3.3 節で述べた通り，スケールフリー性によって $O(|F_{\hat{G}}|) \approx O(|V_G|/(\alpha - k)^{(\gamma-1)})$ が成り立つため，定理 1 で導いた時間計算量は $O(\tau(|V_G| + |F_{\hat{G}}|(\alpha - k))) \approx O(\tau|V_G|(1 + 1/(\alpha - k)^{(\gamma-2)}))$ と近似できる．このとき実用的には $O(1 + 1/(\alpha - k)^{(\gamma-2)}) \approx O(1)$ と見なせるため，BnM 法は $O(\tau|V_G|)$ 時間で実行可能である．ここで理論的には τ は $|V_{G^*}|$ に依存する．$|V_{G^*}|$ の値は探索の過程で動的に減少するため，τ の実際の値は $|V^*_G|$ より遥かに小さい．これにより BnM 法はノード数に対しておおよそ線形のスケーラビリティを持つ．4 節にて実世界のソーシャルネットワークに対する BnM 法の効率性とスケーラビリティを検証する．

定理 2 BnM 法は正確な最大 k-plex を出力する.

証明 BnM 法は（1）$\hat{G}$ の構築（3 行目），および（2）BRANCH 関数の呼び出し（4 行目，11–26 行目）の二つのステップからなる.（1）について Algorithm 2 の 3 行目に示した通り，定義 4 に基づき $\hat{G}$ の構築には unfruitful ノードの削除のみしか行わない．unfruitful ノードは定義 3 により (α, k)-plex に含まれないことが明らかである.（2）について Algorithm 2 の 18–25 行目に示した通り，BnM 法はノード $v \in \hat{G}$ を逐次選択し，$\hat{G}_{-v}$ から (α, k)-plex を探索する．補題 1 で示した通り，もし v がスーパーノード s に属するならば，s 中の全てのノードは $\hat{G}_{-v}$ 中の任意の (α, k)-plex に含まれない．そのため Algorithm 2 が直接分枝探索しないノード（18 行目）が存在しても，BnM 法は全ての (α, k)-plex を各イテレーションで計算できる．したがって BnM 法は正確な最大 k-plex を出力する． □

定理 2 は BnM 法が効率的な最大 k-plex の探索を実現するためにノードの集約および削除を行うが，それらが最大 k-plex の品質に影響を与えないことを示している．

表 1: 実データセットに対する実行時間

Graphs	$\|V_G\|$	$\|E_G\|$	γ	α_{max}	k	BnM	BnB	Maplex	KpLeX
4*soc-LiveMocha	4*104,103	4*2,193,083	4*1.75	19	2	16.23 (± 0.31) sec.	467.19 (± 4.42) sec.	48.96 (± 0.91) sec.	**12.01** (± 0.32) sec.
				22	3	**40.80** (± 0.55) sec.	DNF	329.74 (± 1.37) sec.	42.77 (± 0.81) sec.
				25	4	**529.31** (± 2.76) sec.	DNF	5,122.63 (± 7.72) sec.	631.10 (± 4.28) sec.
				28	5	**8,899.67** (± 8.42) sec.	DNF	DNF	DNF
4*soc-douban	4*154,908	4*327,162	4*2.13	12	2	**0.16** (± 0.01) sec.	0.81 (± 0.06) sec.	0.21 (± 0.03) sec.	0.28 (± 0.02) sec.
				14	3	**0.16** (± 0.01) sec.	0.71 (± 0.05) sec.	0.26 (± 0.03) sec.	0.31 (± 0.03) sec.
				16	4	**0.14** (± 0.01) sec.	0.59 (± 0.07) sec.	0.16 (± 0.02) sec.	0.21 (± 0.02) sec.
				17	5	0.10 (± 0.01) sec.	0.20 (± 0.03) sec.	**0.09** (± 0.01) sec.	0.16 (± 0.02) sec.
4*soc-gowalla	4*196,591	4*950,327	4*2.05	30	2	**1.02** (± 0.09) sec.	42.10 (± 1.30) sec.	1.09 (± 0.09) sec.	1.60 (± 0.09) sec.
				31	3	**1.12** (± 0.10) sec.	43.45 (± 1.46) sec.	1.17 (± 0.08) sec.	1.94 (± 0.13) sec.
				32	4	3.49 (± 0.21) sec.	90.91 (± 3.08) sec.	**3.27** (± 0.21) sec.	9.20 (± 0.82) sec.
				32	5	**28.00** (± 0.87) sec.	280.44 (± 6.45) sec.	58.54 (± 1.10) sec.	64.14 (± 3.71) sec.
4*soc-twitter-follows	4*404,719	4*713,319	4*1.18	8	2	1.53 (± 0.07) sec.	11.42 (± 0.23) sec.	**0.81** (± 0.09) sec.	0.98 (± 0.05) sec.
				9	3	1.28 (± 0.05) sec.	23.56 (± 0.36) sec.	0.75 (± 0.08) sec.	**0.72** (± 0.04) sec.
				11	4	0.61 (± 0.02) sec.	24.44 (± 0.32) sec.	**0.52** (± 0.08) sec.	0.68 (± 0.07) sec.
				13	5	**0.29** (± 0.02) sec.	28.71 (± 0.47) sec.	0.32 (± 0.03) sec.	0.44 (± 0.06) sec.
4*soc-youtube	4*495,957	4*1,936,748	4*1.88	20	2	**2.49** (± 0.18) sec.	17.23 (± 1.72) sec.	3.26 (± 0.21) sec.	4.52 (± 0.31) sec.
				21	3	19.31 (± 0.89) sec.	30.06 (± 1.96) sec.	16.25 (± 0.56) sec.	**5.81** (± 0.39) sec.
				24	4	21.31 (± 0.85) sec.	**20.12** (± 2.01) sec.	50.46 (± 0.91) sec.	23.22 (± 0.80) sec.
				26	5	**11.90** (± 0.52) sec.	25.00 (± 1.75) sec.	3,701.25 (± 20.31) sec.	63.00 (± 2.13) sec.
4*soc-delicious	4*536,108	4*1,365,961	4*2.14	23	2	0.20 (± 0.01) sec.	1.51 (± 0.15) sec.	**0.19** (± 0.05) sec.	0.89 (± 0.04) sec.
				27	3	**0.21** (± 0.02) sec.	1.79 (± 0.17) sec.	**0.21** (± 0.04) sec.	0.82 (± 0.04) sec.
				29	4	**0.14** (± 0.01) sec.	1.11 (± 0.14) sec.	0.16 (± 0.02) sec.	0.77 (± 0.04) sec.
				30	5	**0.08** (± 0.01) sec.	0.79 (± 0.11) sec.	0.16 (± 0.03) sec.	0.63 (± 0.02) sec.
4*soc-FourSquare	4*639,014	4*3,214,986	4*1.52	35	2	**621.23** (± 2.31) sec.	5,291.21 (± 24.33) sec.	798.20 (± 8.21) sec.	680.21 (± 4.12) sec.
				39	3	886.14 (± 2.91) sec.	DNF	1,302.39 (± 10.04) sec.	**430.99** (± 3.10) sec.
				42	4	910.26 (± 3.09) sec.	DNF	5,788.21 (± 25.42) sec.	**604.49** (± 4.47) sec.
				44	5	**3,490.63** (± 6.89) sec.	DNF	DNF	4,289.75 (± 21.08) sec.
4*soc-youtube-snap	4*1,134,890	4*2,987,624	4*1.72	20	2	**1.87** (± 0.19) sec.	32.80 (± 0.90) sec.	3.92 (± 0.42) sec.	3.57 (± 0.29) sec.
				21	3	9.89 (± 0.79) sec.	79.56 (± 1.16) sec.	49.95 (± 1.89) sec.	**4.99** (± 0.41) sec.
				24	4	**10.02** (± 0.90) sec.	77.37 (± 1.20) sec.	200.34 (± 3.55) sec.	10.11 (± 0.85) sec.
				26	5	**16.20** (± 1.44) sec.	101.02 (± 2.18) sec.	3,922.66 (± 17.16) sec.	79.22 (± 2.15) sec.
4*soc-lastfm	4*1,191,805	4*4,519,330	4*1.93	18	2	**4.23** (± 0.35) sec.	261.19 (± 4.74) sec.	9.88 (± 1.14) sec.	7.90 (± 0.21) sec.
				21	3	**6.96** (± 0.39) sec.	823.35 (± 6.45) sec.	62.10 (± 2.43) sec.	10.12 (± 0.30) sec.
				24	4	**32.77** (± 1.02) sec.	DNF	1,166.42 (± 8.70) sec.	42.29 (± 0.38) sec.
				27	5	**201.00** (± 1.42) sec.	DNF	DNF	312.10 (± 2.51) sec.
4*soc-pokec	4*1,632,803	4*22,301,964	4*2.88	31	2	**16.73** (± 1.31) sec.	112.42 (± 1.39) sec.	39.92 (± 1.36) sec.	21.10 (± 0.71) sec.
				32	3	**17.33** (± 1.48) sec.	151.04 (± 1.46) sec.	44.86 (± 1.82) sec.	23.54 (± 0.59) sec.
				32	4	**17.32** (± 1.29) sec.	96.34 (± 1.21) sec.	36.21 (± 1.27) sec.	22.16 (± 0.62) sec.
				34	5	**18.11** (± 1.50) sec.	77.00 (± 0.00) sec.	32.05 (± 1.14) sec.	26.35 (± 0.91) sec.
4*soc-flixster	4*2,523,386	4*7,918,801	4*1.88	38	2	34.50 (± 0.82) sec.	941.23 (± 5.12) sec.	110.79 (± 4.28) sec.	**28.1** (± 1.22) sec.
				42	3	**290.90** (± 1.74) sec.	DNF	DNF	344.80 (± 4.67) sec.
				46	4	**1,984.52** (± 7.21) sec.	DNF	DNF	2,721.11 (± 6.72) sec.
				49	5	**4,016.59** (± 13.36) sec.	DNF	DNF	DNF
4*soc-livejournal	4*4,033,137	4*27,933,062	4*2.47	214	2	**2.12** (± 0.21) sec.	17.34 (± 0.38) sec.	5.82 (± 0.27) sec.	20.21 (± 0.71) sec.
				214	3	**2.24** (± 0.20) sec.	17.23 (± 0.32) sec.	6.21 (± 0.24) sec.	19.20 (± 0.49) sec.
				214	4	**2.25** (± 0.21) sec.	17.80 (± 0.35) sec.	8.11 (± 0.27) sec.	15.70 (± 0.42) sec.
				214	5	**2.39** (± 0.22) sec.	17.60 (± 0.38) sec.	8.23 (± 0.23) sec.	13.10 (± 0.31) sec.

4 評価実験

本節では BnM 法の実行速度を実験的に評価する．

比較手法：我々は BnM 法を BnB[1] [28]，Maplex[2] [20]，および KpLeX[3] [16] の 3 つの最先端の既存手法と比較する．すべての実験は Intel Xeon Gold 6246R CPU 3.40 GHz and 128 GiB RAM を搭載したマシン上で C/C++によるシングルスレッドプログラムとして実装し，コンパイルオプションを “-O3” とした．また実験結果には 10 回の実行時間の平均値と標準偏差を載せる．

データセット：本稿では Network Repository [29] で公開されている 22 の実グラフデータセットを用いる．表 1 に詳細を示す．表の α_{max} と γ はそれぞれ最大 k-plex の大きさと次数分布の偏りを示しており，γ が大きいときグラフは低次のノードを多く持つ．紙面の都合上，小さなソーシャルネットワークに対する実験結果については省略する．

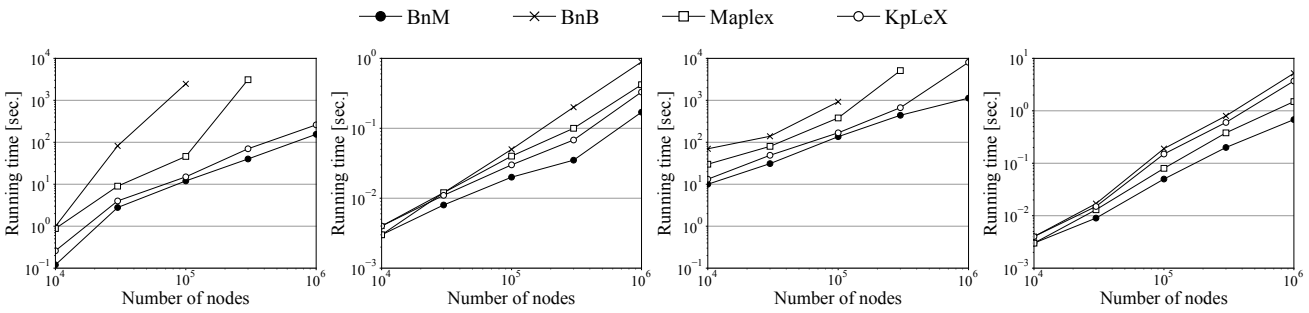
4.1 実行時間の評価

表 1 に最大 k-plex を出力するまでに要した実行時間を示す．実行に 10,000 秒以上掛かった場合は DNF とした．また既存手法 [20, 28] と同様に，k は 2 から 5 とした．BnM 法は正確な最

1：BnB のソースコード: https://github.com/JimNenu/codekplex

2：Maplex のソースコード: https://github.com/ini111/Maplex

3：KpLeX のソースコード: https://github.com/huajiang-ynu/kplex



(a) soc-lastfm (b) soc-pokec (c) soc-flixster (d) soc-livejournal

図 2: スケーラビリティ

表 2: node-merging 法の効果

Graphs	k	BnM	BnM (w/o merge)	GRR
4*soc-LiveMocha	2	**16.23** sec.	DNF	0.05%
	3	**40.8** sec.	DNF	0.03%
	4	**529.31** sec.	DNF	0.03%
	5	**8899.67** sec.	DNF	0.03%
4*soc-douban	2	**0.16** sec.	358.84 sec.	0.04%
	3	**0.16** sec.	291.14 sec.	0.05%
	4	**0.14** sec.	285.92 sec.	0.04%
	5	**0.1** sec.	202.42 sec.	0.05%
4*soc-gowalla	2	**1.02** sec.	9,537.81 sec.	0.01%
	3	**1.12** sec.	DNF	0.01%
	4	**3.49** sec.	DNF	0.04%
	5	**28** sec.	DNF	0.03%
4*soc-twitter-follows	2	**1.53** sec.	2,156.17 sec.	0.06%
	3	**1.28** sec.	7,829.11 sec.	0.01%
	4	**0.61** sec.	7,356.13 sec.	0.01%
	5	**0.29** sec.	7,001.65 sec.	0.01%
4*soc-youtube	2	**2.49** sec.	3,801.64 sec.	0.06%
	3	**19.31** sec.	7,301.28 sec.	0.25%
	4	**21.31** sec.	7,016.16 sec.	0.26%
	5	**11.9** sec.	5,792.48 sec.	0.20%
4*soc-delicious	2	**0.2** sec.	371.38 sec.	0.05%
	3	**0.21** sec.	480.72 sec.	0.04%
	4	**0.14** sec.	301.18 sec.	0.05%
	5	**0.08** sec.	266.02 sec.	0.03%
4*soc-FourSquare	2	**621.23** sec.	DNF	0.05%
	3	**886.14** sec.	DNF	0.04%
	4	**910.26** sec.	DNF	0.04%
	5	**3490.63** sec.	DNF	0.03%

大 k-plex を出力するが，BnB，Maplex，および KpLeX に対してそれぞれ最大 305.2，311.0，および 9.53 倍高速である．特に BnM 法は soc-pokec や soc-livejournal のような γ が大きいグラフに対して既存手法より非常に効率的である．定理 1 で示した通り，BnM 法は既存手法に対して時間計算量が小さいため効率的であり，これは実験結果と合致する．

BnB，Maplex，および KpLeX と異なり，最も計算コストが大きくなる $k = 5$ であっても BnM 法は多くのデータセットで最大 k-plex を発見している．既存手法のグラフ削減アルゴリズムは次数分布の偏りに関係なく全てのノードを繰り返し探索して削除するノードを計算する．しかし実世界のソーシャルネットワークには低次のノードが多く，これらのノードがスケーリフリー性により最大 k-plex になる可能性が低い．そのためグラフ削減アルゴリズムは大きな k に対して最大 k-plex の発見が困難である．それに対して 3.3 節で議論した通り，BnM 法はスケールフリー性に基づく node-merging 法を用いて，γ が大きいときには効率的に最大 k-plex に含まれないノードを削除する．これにより BnM 法は巨大なソーシャルネットワークに対しても効率的に最大 k-plex を発見できる．

4.2 node-merging 法の効果

本節では BnM 法の実行時間を node-merging 法を適用しない BnM 法と比較することで，BnM 法のアプローチの有効性を検証する．表 2 は各アルゴリズムの実行時間であり，BnM (w/o merge) は node-merging 法を用いない BnM 法を示す．また DNF は実行が 10,000 秒以内に終了しなかったことを示す．表 2 には node-merging 法によってどの程度グラフが小さくなったかを示すグラフ削減比 (GRR) も併せて示している．ここで GRR は BnM (w/o merge) に対する BnM 法が参照したエッジ数の割合として計算する．

BnM (w/o merge) と比較して BnM 法は平均して実行時間の 99.96%を削減した．表 2 から分かるように，node-merging 法は BnM (w/o merge) と比較して計算するエッジ数の 99.93%を削減している．これらの結果は node-merging 法によってグラフサイズが小さくなったことにより，BnM の探索空間が大幅に削減されていることを示す．定理 1 で用いた τ の大きさは safe-to-merge ノードをマージしたあとのグラフ G^* の大きさに依存するが，GRR は平均して 0.07% 程度であるため，τ は非常に小さいことと考えられる．したがって，node-merging 法は safe-to-merge ノードをマージすることにより BnM 法の効率性を効果的に向上させる．

4.3 スケーラビリティ

BnM のスケーラビリティについて評価する．図 2 は表 1 中の 4 つの大きなソーシャルネットワークである soc-lastfm，soc-pokec，soc-flixster，および soc-livejournal の実行時間をノード数を変えながら計測した．我々は Metropolis-Hastings Random Walk [30] を用いて元のグラフからノードを 1×10^4，3×10^4，1×10^5，3×10^5，および 1×10^6 個サンプリングし，$k = 5$ としてそれぞれのグラフに対する実行時間を評価する．なお実行に 10,000 秒以上要した場合の結果は図 2 には載せていない．

BnM 法はほとんどのグラフにおいてノード数に対してほぼ線形なスケーラビリティを示す．これらの結果はソーシャルネットワークの次数分布の偏りによるものである．定理 1 で示した通り，BnM 法は γ が十分に大きいとき $O(\tau|V_G|(1+1/(\alpha-k)^{(\gamma-2)}) \approx O(\tau|V_G|)$ 時間で実行可能である．つまり γ が大きいとき BnM 法は良いスケーラビリティを持ち，表 1 に示す通り実世

界の大規模なソーシャルネットワークは大きな γ を持つことが多い．したがって，BnM 法は大規模なソーシャルネットワークにおいてほぼ線形なスケーラビリティを実現する．

5 おわりに

本稿では，大規模ソーシャルネットワークのための効率的な最大 k-plex 探索アルゴリズムである BnM 法を提案する．BnM 法はソーシャルネットワークのスケールフリー性に着目し，safe-to-merge ノードをマージすることでグラフを縮小する．我々の実験において BnM 法は最先端の既存手法と比較して最大 k-plex を高速に発見できることを確認した．本稿を通じて高速な BnM 法を用いることで，多くのソーシャルネットワーク分析およびアプリケーションの性能向上が期待できる．

謝　辞

本研究の一部は JST さきがけ (JPMJPR2033)，JSPS 科研費 (JP22K17894), JSPS 科研費 (JP22J10972) の支援を受けたものである．

文　献

[1] B Balasundaram, Sergiy Butenko, I. Hicks, and S Sachdeva. Clique Relaxations in Social Network Analysis: The Maximum k-Plex Problem. *Operations Research*, 01 2007.

[2] Yoshiaki Okubo, Makoto Haraguchi, and Etsuji Tomita. Structural Change Pattern Mining Based on Constrained Maximal k-Plex Search. In Jean-Gabriel Ganascia, Philippe Lenca, and Jean-Marc Petit, editors, *Discovery Science*, pages 284–298, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[3] Alessio Conte, Tiziano De Matteis, Daniele De Sensi, Roberto Grossi, Andrea Marino, and Luca Versari. D2K: Scalable Community Detection in Massive Networks via Small-Diameter k-Plexes. KDD '18, page 1272–1281, New York, NY, USA, 2018. Association for Computing Machinery.

[4] Seidman, Stephen B and Foster, Brian L. A Graph-theoretic Generalization of the Clique Concept. *Journal of Mathematical sociology*, 6(1):139–154, 1978.

[5] Uffe Wiil, Nasrullah Memon, and Panagiotis Karampelas. Detecting New Trends in Terrorist Networks. pages 435–440, 08 2010.

[6] Balabhaskar Balasundaram. *Graph theoretic generalizations of clique: Optimization and extensions*. Texas A&M University, 2007.

[7] Michael G. Noll and Christoph Meinel. Web Search Personalization Via Social Bookmarking and Tagging. In Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web*, pages 367–380, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[8] David Carmel, Naama Zwerdling, Ido Guy, Shila Ofek-Koifman, Nadav Har'el, Inbal Ronen, Erel Uziel, Sivan Yogev, and Sergey Chernov. Personalized Social Search Based on the User's Social Network. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, page 1227–1236, New York, NY, USA, 2009. Association for Computing Machinery.

[9] George Danezis, Tuomas Aura, Shuo Chen, and Emre Kıcıman. How to share your favourite search results while preserving privacy and quality. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 273–290. Springer, 2010.

[10] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-Attention Graph Pooling. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3734–3743. PMLR, 09–15 Jun 2019.

[11] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral Clustering with Graph Neural Networks for Graph Pooling. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[12] Benjamin Doerr, Mahmoud Fouz, and Tobias Friedrich. Why Rumors Spread so Quickly in Social Networks. *Commun. ACM*, 55(6):70–75, jun 2012.

[13] Amirhosein Bodaghi and Jonice Oliveira. The Theater of Fake News Spreading, Who Plays Which Role? A Study on Real Graphs of Spreading on Twitter. *Expert Systems with Applications*, 189:116110, 2022.

[14] Yang Wang, D. Chakrabarti, Chenxi Wang, and C. Faloutsos. Epidemic Spreading in Real Networks: an Eigenvalue Viewpoint. In *22nd International Symposium on Reliable Distributed Systems, 2003. Proceedings.*, pages 25–34, 2003.

[15] Lin Wang and Guan-zhong Dai. Global Stability of Virus Spreading in Complex Heterogeneous Networks. *SIAM Journal on Applied Mathematics*, 68(5):1495–1502, 2008.

[16] Hua Jiang, Dongming Zhu, Zhichao Xie, Shaowen Yao, and Zhang-Hua Fu. A New Upper Bound Based on Vertex Partitioning for the Maximum K-plex Problem. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 1689–1696. International Joint Conferences on Artificial Intelligence Organization, 8 2021.

[17] Hannes Moser, Rolf Niedermeier, and Manuel Sorge. Exact Combinatorial Algorithms and Experiments for Finding Maximum k-Plexes. *Journal of combinatorial optimization*, 24(3):347–373, 2012.

[18] Mingyu Xiao, Weibo Lin, Yuanshun Dai, and Yifeng Zeng. A Fast Algorithm to Compute Maximum k-Plexes in Social Network Analysis. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[19] Peilin Chen, Hai Wan, Shaowei Cai, Jia Li, and Haicheng Chen. Local Search with Dynamic-threshold Configuration Checking and Incremental Neighborhood Updating for Maximum k-Plex Problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2343–2350, 2020.

[20] Yi Zhou, Shan Hu, Mingyu Xiao, and Zhang-Hua Fu. Improving Maximum k-Plex Solver via Second-Order Reduction and Graph Color Bounding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12453–12460, 2021.

[21] James Abello, Mauricio G. C. Resende, and Sandra Sudarsky. Massive Quasi-Clique Detection. *Proceedings of the 5th Latin American Symposium on Theoretical Informatics*, page 598–612, 2002.

[22] Immanuel M Bomze, Marco Budinich, Panos M Pardalos, and Marcello Pelillo. The Maximum Clique Problem. *Handbook of Combinatorial Optimization*, 4:1–74, 05 1999.

[23] Benjamin McClosky and Illya V Hicks. Combinatorial Algorithms for the Maximum k-Plex Problem. *Journal of Combinatorial Optimization*, 23(1):29–49, 2012.

[24] Mingyu Xiao and Shaowei Kou. Exact Algorithms for the Maximum Dissociation Set and Minimum 3-Path Vertex Cover Problems. *Theor. Comput. Sci.*, 657(PA):86–97, January 2017.

[25] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On Power-Law Relationships of the Internet Topology. *SIGCOMM Comput. Commun. Rev.*, 29(4):251–262, 08 1999.

[26] Albert-László Barabási and Réka Albert. Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512, 1999.

[27] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[28] Jian Gao, Jiejiang Chen, Minghao Yin, Rong Chen, and Yiyuan Wang. An Exact Algorithm for Maximum k-Plexes in Massive Graphs. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 1449–1455. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[29] Ryan Rossi and Nesreen Ahmed. The network data repository with interactive graph analytics and visualization. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[30] Yingan Cui, Xue Li, Junhuai Li, Huaijun Wang, and Xiaogang Chen. A Survey of Sampling Method for Social Media Embeddedness Relationship. *ACM Comput. Surv.*, 02 2022. Just Accepted.