高速な最大 k-Plex 探索アルゴリズムの提案

真次 彰平† 塩川 浩昭††,†††

† 筑波大学システム情報工学研究科 〒305–8573 茨城県つくば市天王台 1-1-1
†† 筑波大学計算科学研究センター 〒305–8577 茨城県つくば市天王台 1-1-1
††† JST さきがけ

E-mail: † matsugu@kde.cs.tsukuba.ac.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る．グラフ中の最大の k-plex を求める最大 k-plex 探索アルゴリズムは，実世界の大規模なコミュニティの発見に大きく寄与する．しかしながら，最大 k-plex 探索に関する既存手法は同型のグラフ構造に対して繰り返しグラフ縮小を実行するため低速である．そこで本稿では局所的なグラフ構造に対してグラフ縮小結果をメモ化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メモ化が最も効果的なノードの探索順序を特定し，提案手法が既存手法と比較して最大 33 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

Twitter や Facebook といったソーシャルネットワークの普及に伴い膨大なデータが生成されており，それらのデータを分析する技術が注目されている．特に，データエンティティをノード，それらの関係をエッジとして表現するグラフ分析は，ネットワーク構造を持つ大規模なデータから有用な情報を抽出するための重要な分野である．

多くのグラフ分析アルゴリズム [1, 2, 3]では，エッジが密である部分グラフ，即ち密部分グラフの抽出が重要とされている．とりわけ，頑健な密部分グラフのモデルの一つに，クリーク [4]がある． クリークはノード集合内の任意の 2 ノード間にエッジが存在するような部分グラフのことをいう．グラフ中のクリークの抽出技術 [5]は，実世界のコミュニティの発見に大きく貢献する．

しかしながら、クリークは部分グラフの構造への制約が強く，大きなコミュニティを抽出するには不適である [6]. そのため，より制約の緩い，クリークの一般化にあたる概念として k-plex [6]が近年用いられている．k-plex は部分グラフ中のエッジの欠損に基づいて構造の評価を行うモデルである．具体的には，s個のノードから構成された部分グラフHが k-plex であるとき，部分グラフHの各ノード中のs − k個以上のノードグラフと接続するという条件を満たす．k-plex は部分グラフ中に多少のエッジの欠損を許容するため，クリークと比較して柔軟な構造を抽出できる．

例えば図 1 (a)はクリークであるが図 1 (b), (c)はクリークではない．なぜなら，図 1 (b)はエッジ(v_6, v_8)，図 1 (c)はエッジ$(v_{11}, v_{13}), (v_{11}, v_{15}), (v_{13}, v_{14})$が存在していないためである．このとき，図 1 (b)は最低次数が 3 であるため 2-plex，図 1 (c)は最低次数が 2 であるため 3-plex である．

k-plex に対する既存の探索問題のひとつに，最大 k-plex 抽出問題 [7]がある．最大 k-plex 抽出問題は，グラフG，および整数kが与えられたとき，グラフ中の最もノード数の多い k-plex を探索する問題である．Gao らが提案した最大 k-plex 抽出問題に対する最先端の既存手法 [8]は、枝刈り探索を用いて探索空間を動的に削減し，高速化を実現した．しかしながら，ノード数 60 万程度のグラフに対して 1 時間以内に解を出力できず，依然として実行速度が遅い．

そこで本稿では，高速な最大 k-plex 抽出手法の提案を目的とする．既存手法では動的枝刈りによる効率化を採用しているが，探索の過程でグラフ全体のノードとエッジに対して繰り返し枝刈り処理を行う必要がある．これは同一のノード，あるいは同型のグラフ構造に対して枝刈り処理および枝刈りの可否判定処理を何度も実行するため非効率であり，グラフが大規模になった場合には膨大な計算時間を要する．そのため本稿では，枝刈り処理自体に用いる計算時間を短縮することで，計算時間の効率化を図る。具体的には，同型のグラフ構造に対しての枝刈りパタンが常に等しくなることに着目して，枝刈り処理に要する計算時間を削減する．

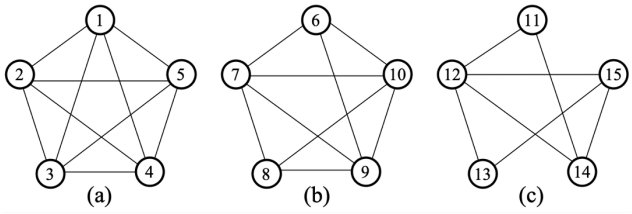

図 1 k-plex の例

本研究の貢献は以下の 3 つである．

- **枝刈り処理の重複除去による処理回数の削減**：提案手法は同一のノードに対する枝刈り処理の判定処理の回数を削減し(3.2.2 節)，既存手法 [8]と比較して最大 93%の処理が削減されることを実証した(4.3 節).
- **高速な探索アルゴリズムの設計**：提案手法は枝刈り処理を効率化した探索アルゴリズムを設計し(3.2.4 節)，実験により既存手法 [8]と比較して同様の結果を最大 33 倍高速に得られることを示した(4.2 節)
- **探索順序の改善による高速化**：上記の高速化アルゴリズムは，探索の過程においてノードを選択する順序により性能が大きく変化する．そのため，我々はノードの選択順序に関するいくつかのモデルを提案し(3.2.3 節)，実験によりそれらの特徴を比較した．(4.3 節)

提案手法を用いることで, 最大 33 倍の高速化を実現しただけでなく，既存手法では実行不可能であった数百万ノード程度の規模のグラフに対して現実的な時間で最大 k-plex を抽出することが可能となった．これには，DBLP のような共著関係を表すグラフや, Wikipedia のハイパーリンクによるグラフがこの規模に該当し，実世界において我々の提案手法はより多くの場面において適用可能であることを示唆している．

表 1　本稿で用いる記号とその定義

記号	定義
G	計算対象のグラフ
V	Gに含まれるノードの集合
E	Gに含まれるエッジの集合
$d_G(v)$	ノードvのGにおける次数
$N_G(v)$	ノードvのGにおける隣接ノードの集合
S	計算過程の k-plex
$G[S]$	SによるGの誘導部分グラフ
d_{avg}	Gの平均次数
LB	計算過程の解の下限値
$*R_{G,k}$	$*$-reduction により除去されるノードの集合

最後に, 本研究の応用について説明する. k-plex の検出は, 従来では実世界の大規模なグラフに対して満足に実行できなかった．しかしながら，提案手法を用いることでグラフ中の最大の k-plex の高速な検出が可能となる．そのため，クラスタリングやコミュニティ検索といった高度な密部分グラフ分析技術の基盤として，提案手法のアルゴリズムが貢献することが期待される．

本稿の構成は以下の通りである．2 節では前提となる知識および既存手法について説明し，3 節では提案手法を示す．また 4 節では既存手法と提案手法の評価実験の結果を示し，5 節では関連研究を紹介する．最後に，6 節にて本稿のまとめを述べる．

2. 前提知識

本研究に関する基本事項について説明する．本研究で対象とするグラフは単純で連結な重みなし無向グラフ$G(V,E)$である．ここで，Vはノード集合，Eはエッジ集合である．表 1 に本稿で用いる主な記号とその定義を示す.

2.1. 問題設定

本節では本稿で取り扱う最大 k-plex 抽出問題 [7]を説明する．まず，k-plex [6]について定義する.

定義 1　(k-plex)

グラフ$G(V,E)$，および整数 kについて，ノード集合Sが k-plex であるとは，Sの任意のノード$v \in S$について，S中のvと隣接していないノードの数がk個未満であることをいう．即ち，Sが k-plex $\Leftrightarrow \forall v \in S, d_{G[S]}(v) \geq |S| - k$である.

k-plex は Abello らによってクリークの一般化として提案された密部分グラフのモデルである．k-plex はクリークのアイデアを継承しつつ，柔軟な密部分グラフを検出する.

次に, グラフ中の最も大きな k-plex を見つける問題である，最大 k-plex 抽出問題について説明する.

問題定義　(最大 k-plex 抽出問題)

グラフ$G(V,E)$，整数kが与えられたとき，最大 k-plex 抽出問題は，G中の最大の k-plex を探索する問題である.

最大 k-plex 抽出問題は, $k = 1$のとき最大クリーク問題に一致することから，最大クリーク問題を包含する難解な問題であるといえる．また，最大 k-plex 抽出問題は NP 困難であることが示されている [7].

2.2. 既存手法

本節では既存手法 [8]における最大 k-plex 抽出問題の解法について解説する．既存手法は素朴な深さ優先探索を基本とした 4 つの枝刈りアルゴリズムを提案し，それらの枝刈りアルゴリズムを状況に応じてサブルーチンとして呼び出すことで計算の効率化を図った.

Algorithm 1: vertex-reduction(G, LB, k)

Input: $G = (V,E)$, 整数LB, k
Output: 削減されたグラフG'
1. $Q \leftarrow \emptyset$;
2. **foreach** $v \in V$ **do**
3. **if** $d_G(v) + k \leq LB$ **then** Q.push(v);
4. **while** $Q \neq \emptyset$ **do**
5. $v \leftarrow Q.pop(v)$;
6. **foreach** $u \in N_G(v)$ **do**
7. **if** $d_G(u) + k - 1 \leq LB$ **then** Q.push(u);
8. G.remove(v);
9. **return** G;

Algorithm 2: v-reduction(G, S, LB, k, v)

Input: $G = (V,E)$, 整数LB, k,k-plex $S \subseteq V, v \in S$
Output: 削減されたグラフG'
1. $r_{G,S}(u)$を$\forall u \in (N_G(v) \setminus S) \cup U$について求める;
2. $Q \leftarrow \emptyset$;
3. **foreach** $u \in N_G(v) \setminus S$ **do**
4. $c_{G,S}(u,v)$を求める;
5. **if** $c_{G,S}(u,v) \leq LB - |S| + 1$ **then** Q.push(u);
6. **while** $Q \neq \emptyset$ **do**
7. $u \leftarrow Q.pop(v)$;
8. $H \leftarrow \left(N_G(u) \cap N_G(v)\right) \setminus S$;
9. **if** $u \in V$ **then** G.remove(u);
10. **foreach** w$\in H$ **do**
11. $c_{G,S}(w,v)$を求める;
12. **if** $c_{G,S}(w,v) \leq LB - |S| + 1$ **then** Q.push(w);
13. **return** G;

本節では，まず既存手法の基となる，最大 k-plex 探索における諸定理について順に述べ，次に枝刈り探索アルゴリズムを示す. なお, 各定理は既存手法 [8]にて証明されているため，ここでは省略する.

2.2.1. 準備としての諸定理

本節では既存手法 [8]のアルゴリズム設計の元となる定理について順に説明する.

定理 1

グラフ$G(V,E)$，整数k, LBが与えられたとき，次を満たすノード$v \in V$はLBより大きな k-plex に含まれない.

$$d_G(v) + k \leq LB.$$

定理 1 は k-plex の定義より，次数が小さなノードは明らかに探索候補に含まれないことを示している．次の定理を示す前に，次の定義 2，3 を導入する.

定義 2

グラフ$G(V,E)$, k-plex $S \subseteq V$およびノード$u \in V$について，関数$r_{G,S}(u)$を次のように定義する.

$$r_{G,S}(u) = k - |S \setminus (N_G(u) \cup \{u\})| - 1.$$

関数$r_{G,S}(u)$はノード集合$S \cup \{u\}$を含む k-plex に含まれるuに隣接しないノード数の上限値である．即ち，定義 2 は$S \cup \{u\}$がuに隣接しないノードを高々$r_{G,S}(u)$個までしか追加できないということを表す.

定義 3

グラフ$G(V,E)$，k-plex $S \subseteq V$について，関数$c_{G,S}(u,v)$を次のように定義する.

$$c_{G,S}(u,v) = \min\{d_{G'}(u) + r_{G,S}(u) + 1, |V'|\} + r_{G,S}(v).$$

Algorithm 3: k-reduction(G,S,k)

Input: $G=(V,E)$, 整数 k, k-plex S
Output: 削減されたグラフG'
1. **foreach** $u \in S$ s.t. $|S \setminus N_G(u)|+1=k$ **do**
2. G.remove$(\forall v \in V \setminus (N_G(u) \cup S))$;
3. **foreach** $u \in V \setminus S$ s.t. $|S \setminus N_G(u)|+1>k$ **do**
4. G.remove(u);
5. **return** G;

ここで, $G'(V',E')=G[(N_G(v) \setminus S) \cup \{v\}], v \in V, u \in V' \setminus \{v\}$である.

関数$c_{G,S}(u,v)$はノード集合$S \cup \{u,v\}$を含む k-plex に含まれるuに隣接しないノード数の上限値である. 定義 2, 3 を用いて, 定理 2, 3 が導ける.

定理 2

グラフ$G(V,E)$, 整数k, k-plex $S \subseteq V$, およびノード$v \in V, u \in N_G(v) \setminus S$が与えられたとき, $S \cup \{u,v\}$を含む最大の k-plex の大きさは$c_{G,S}(u,v)+|S \setminus \{v\}|$以下である.

定理 2 はSにノードvを追加するときにそれ以外のノードuに関する k-plex の上限値を示すものである.

定理 3

グラフ$G(V,E)$, 整数k, k-plex $S \subseteq V$が与えられたとき, 次のいずれかの条件を満たすノード$v \in V \setminus S$は S を含む k-plex に含まれない.

1. $r_{G,S}(u)=0$となるノード$u \in S \setminus N_G(v)$が存在する.
2. $r_{G,S}(v)=0$である.

定理 3 は k-plex の制約により, Sに追加すると即座に k-plex の定義に違反するノードを示す. 条件 1 について, これ以上隣接しないノードを追加できないノード$u \in S$が存在するとき, uに隣接しない$V \setminus S$中の全てのノードはSを含む k-plex に含まれない. また条件 2 について, ノードv自身がS中に隣接しないノードをk個含むならば, vは明らかにSを含む k-plex に含まれない.

2.2.2. グラフ除去アルゴリズム

既存手法は, これらの定理を用いてノード除去を行いながら探索を行う. まず, 定理 1 から定理 3 を用いて設計される除去アルゴリズムについて順に説明する.

(1) vertex-reduction: 定理 1 に基づいた除去アルゴリズムである, vertex-reduction を Algorithm 1 に示す. vertex-reduction ではノードが除去されることによって次数が小さくなり, 更に除去できるノードが生じることがある. そのため幅優先探索を用い次数の小さなノードを逐次除去する.

(2) v-reduction: 定理 2 に基づいた除去アルゴリズムである, vertex-reduction を Algorithm 2 に示す. v-reduction では先に近傍の r_(G,S) (u)を求めておき(1 行目), vertex-reduction と同様の手順で除去を行う. ただし, 条件判定の度に c_(G,S) (u,v)計算する必要がある(4, 11 行目).

(3) k-reduction: 定理 3 に基づいた除去アルゴリズムである, k-reduction を Algorithm 3 に示す. k-reduction では定理 3 における条件 1 と 2 それぞれに除去を行う.

既存手法ではもう一つ subgraph-reduction と呼ばれる除去アルゴリズムを提案している. しかしながら, 提案手法では深く議論しないためこれについては付録にて述べる.

2.2.3. グラフ除去に基づく k-plex 探索

既存手法は, 2.2.2 節で述べた 4 つの除去アルゴリズムを適用しながら, ボトムアップに k-plex を探索する. Algorithm 4 に既存手法の概要を示す. 既存手法は, あるノードを一つ選択し, それを暫定 k-plex に追加する(6 行目), およびグラフから除去する(13 行目)の二通りの操作に基づき, 深さ優先探索を行う. 並行して, 枝刈りアルゴリズムをそれぞれ適用することでノードを除去し, グラフのサイズを動的に小さくする(8-10 行目). また 13 行目では, vを含む k-plex を全て探索し終えたため, vを除去し, それに伴い除去可能なノードを検出する(付録 B).

Algorithm 4: 既存手法

Input: $G=(V,E)$, 整数 k
Output: 最大 k-plex の大きさ
1. $S \leftarrow \emptyset$;
2. $(LB,G) \leftarrow$preprocessing(G,k);
3. **return** $BB(G,S,k,LB)$;

4. function $BB(G,S,k,LB)$
5. **if** $V \setminus S=\emptyset$ **then return** $|S|$;
6. $v \in V \setminus S$を一つ選択, $S \leftarrow S \cup \{v\}$;
7. $G_r \leftarrow G$;
8. $G \leftarrow$k-reduction(G,S,k);
9. $G \leftarrow$vertex-reduction(G,LB,k);
10. $G \leftarrow$v-reduction(G,S,LB,k,v);
11. **if**$\{u|u \in V \setminus S \wedge d_G(u)+k>LB\}|+|S|>LB$ **then**
12. $LB \leftarrow \max(LB, BB(G,S,k,LB))$;
13. $LB \leftarrow \max(LB,$remove-reduction$(G_r,S,LB,k,U))$;
14. **return** LB;

既存手法は, 探索を行う前に簡単な除去アルゴリズムを適用することにより, 静的なグラフサイズの削減とLBの初期化を試みる(2 行目). 既存手法の実験によると, 139 のデータのうち 69 のデータにおいてこの前処理により最大の解が見つかり, 動的な探索, 即ち function BBの実行が不要になったと示されている. しかしながら, 既存手法は依然として多くのデータにおいて動的な探索が必要であり, その一部は現実的な時間で応答できない. そのため本稿では動的な探索の高速化に絞って検討を行うものとし, 前処理の概要は付録 C にて述べる.

最後に, 既存手法の時間計算量について議論する. 既存手法の最悪時間計算量は, $O(2^{|V|}|V|^3)$である. これは非常に大きな計算量であるが, 枝刈りによる定数倍高速化により, $|V|=5\times10^5$程度の実データに対して, 現実的な実行時間に収まることが示されている.

3. 提案手法

本節では提案手法について説明する. 提案手法は最大 k-plex 抽出問題に対して既存手法 [8]より高速に計算することを目指す. 最初に 3.1 節にて提案手法を構成する基本的なアイデアについて触れ、その詳細を 3.2 節にて示す.

3.1. 基本アイデア

3.1 節で述べたように, 既存手法 [8]は枝刈り処理が効率的でなく, 多くの計算時間を要する. 提案手法はこの問題を解決するために, 次の 2 つのアイデアを導入する.

(1) 同型性を利用した枝刈りのメモ化
(2) 起点とするノードの順序の最適化

(1) について, 既存手法は除去したノードの情報を保持しない. しかしながら, 同型の部分グラフ構造に対して枝刈りの結果は極めて類似するため, 多くの枝刈り情報は異なる起点ノード間で共有可能である. この性質に着目して, 枝刈りアルゴリズムの枝刈りに要する時間を削減する.

(2) について, (1)に述べたアルゴリズムは起点ノードの選択順序に依存して性能が大きく変化する. そのため, いくつかのモデルを比較し最適な探索順序について検討する.

3.2. 同型性を利用した枝刈りのメモ化

既存手法が用いる除去アルゴリズムは，現在保持している k-plex Sに対するノードの追加可能性を判定する．我々は，「Sへの隣接関係が等しい，つまり，Sと隣接ノード集合との積集合が一致する二つのノードは，追加可能性の判定結果が一致しやすい」という性質を発見した．これを用いて，一方の判定結果を共有することで他方の判定処理を省略するアルゴリズムを提案する．本節ではこの性質を説明する 4 つの定理と, それに基づいたアルゴリズムを示す．まず簡単のために，次の記号を定義する．

定義 4 (グラフに対する除去演算子)
グラフ$G(V,E)$と除去アルゴリズム r(G)ついて，r により除去されるノード集合がR_Gであるとき，除去されたノード集合による誘導部分グラフ$G[V \setminus R_G]$を，単に$G - R_G$と表す．

定義 5 (k-reduction によって除去されるノード集合)
グラフ$G(V,E)$，k-plex Sおよびノード$v \in V \setminus S$について，k-reduction$(G, S \cup \{v\}, k)$(Algorithm 3)によって除去されるノード集合を$KR_{G,k}(S,v)$によって表す．このとき，

$$KR_{G,k}(S,v) = \{u \in V \setminus S_v \mid |S_v \setminus N_G(u)| > k-1\} \cup \{u \in N_G(x) ¥ S_v \mid \forall x \in S_v s.t. |S_v \setminus N_G(x)| = k-1\}$$

である．ただし，$S_v = S \cup \{v\}$である．

定義 4，5 より，k-reduction について次の定理 4 を導く．

定理 4
グラフ$G(V,E)$, および k-plex Sが与えられたとき, 2 つのノード$u, v \in V$について， k-reduction$(G, S \cup \{u\}, k)$を実行したグラフである$G' = G - KR_{G,k}(S,u)$を考える．このときuとvのSへの隣接関係が等しいならば，k-reduction$(G', S \cup \{v\}, k)$によって新たに除去されるノードは，必ずvに隣接する．つまり，$N_G(u) \cap S = N_G(v) \cap S$ ならば，$KR_{G',k}(S,v) \subseteq N_G(v)$である．

証明
$S_u = S \cup \{u\}$とし，$KR_{G,k}(S,u)$を$K_1(u) = \{w \in V \setminus S_u \mid |S_u \setminus N_G(w)| > k-1\}$ と $K_2(u) = \{w \in N_G(x) ¥ S_u \mid \forall x \in S_u \, s.t. |S_u \setminus N_G(x)| = k-1\}$に分けて考える．このとき$K_1(u)$について，仮定$N_G(u) \cap S = N_G(v) \cap S$より，$S \setminus N_G(v) = S \setminus N_G(u)$である．したがって，

$$K_1(v) \setminus K_1(u) = (V \setminus S_v) \setminus (V \setminus S_u) = \big((V \setminus S) \setminus \{v\}\big) \setminus \big((V \setminus S) \setminus \{u\}\big) = \emptyset. \quad (1)$$

となる．また$K_2(u)$について，明らかに$K_2(u) \subseteq N_G(S_v)$であり，また$N_G(S_v) \setminus N_G(S_u) \subseteq N_G(v)$である．ゆえに，

$$K_2(v) \setminus K_2(u) \subseteq N_G(v). \quad (2)$$

となる．したがって，(1),(2)より

$$KR_{G,k}(S,v) \setminus KR_{G,k}(S,u) \subseteq N_G(v). \quad (3)$$

となる．ここで，$G' \subseteq G$より，

$$KR_{G',k}(S,v) \setminus KR_{G',k}(S,u) \subseteq KR_{G,k}(S,v) \setminus KR_{G,k}(S,u). \quad (4)$$

また，$G' = G - KR_{G,k}(S,u)$より

$$KR_{G',k}(S,u) = \emptyset. \quad (5)$$

(3),(4),(5)より

$$KR_{G',k}(S,v) \subseteq KR_{G,k}(S,v) \setminus KR_{G,k}(S,u) \subseteq N_G(v)$$

となるため，定理 4 は成り立つ． □

定理 4 はSに対する 2 ノードの接続関係が等しいとき，2 回目の k-reduction の適用範囲が限定されることを示す．これは全ノード数に対して線形の時間計算量を要する k-reduction が，2 回目以降は平均次数に対して線形の時間計算量となり，飛躍的に小さくなることを示唆している．また実世界のグラフにはべき乗則が成り立つ [9]ことから，同型の部分グラフ構造を多量に含むことが知られており [1]，定理 4 の条件を満たす場面は非常に多い．

Algorithm 4 に示したように，既存手法 [8]は，k-reduction を実行した後に必ず vertex-reduction 及び v-reduction を実行する．そのため，k-reduction の後に実行される vertex-reduction および v-reduction についても実行される範囲を解析することで，更なる高速化を行う．以下ではまず，この高速化に必要な定義を導入する．

定義 6 (vertex-reduction によって除去されるノード集合)
グラフ$G(V,E)$，及び整数k, LBについて，vertex-reduction(G, LB, k)(Algorithm 1)によって除去されるノード集合を$XR_{G,k}(LB)$によって表す．このとき，

$$XR_{G,k}(LB) = \{u \in V \mid d_G(u) + k \leq LB\}$$

である．

定義 7 (v-reduction によって除去されるノード集合)
グラフ$G(V,E)$，整数k, LB，ノードvおよび k-plex Sについて，v-reduction(G, LB, k)(Algorithm 2)によって除去されるノード集合を$VR_{G,k}(G, S, LB, k, v)$によって表す．このとき，

$$VR_{G,k}(S, LB, v) = \{u \in N_G(v) ¥ S \mid c_{G,S}(u,v) + |S| - 1 \leq LB\}$$

である．

定義 6 および定義 7 より，以下の定理が導出できる．

定理 5
グラフ$G(V,E)$，整数k, LBおよび k-plex Sが与えられたとき，2 つのノード$u, v \in V$について， k-reduction$(G, S \cup \{u\}, k)$ と vertex-reduction$(G - KR_{G,k}(S,u), \mathrm{LB}, k)$を実行したグラフである$G' = G - KR_{G,k}(S,u) - XR_{G,k}(LB)$を考える．このとき$N_G(u) \cap S = N_G(v) \cap S$ならば，k-reduction$(G', S \cup \{v\}, k)$と vertex-reduction$(G' - KR_{G',k}(S,v),\ LB, k)$によって新たに除去されるノードは，必ず$N_G(v)$に隣接する．言い換えると，$v$からの距離が2以下である．

証明
定理 4 により，k-reduction$(G', S \cup \{v\}, k)$によって新たに有無が変化するノードの候補は$N_G(v)$に限られる．したがって，次数が変化するノードの候補は$N_G(v)$に隣接するノードのみであるため，定理 5 は成り立つ． □

定理 6
グラフ$G(V,E)$，整数k, LBおよび k-plex Sが与えられたとき, 2 つのノード$u, v \in V$について， v-reduction$(G, S, LB,\ k, v)$を実行したグラフである$G' = G - VR_{G,k}(S, LB, v)$を考える．このとき$u$と$v$の$S$への隣接関係が等しいならば，v-reduction(G', S, LB, k, v)によって新たに除去されるノードは，必ず$N_G(v)$に含まれる．

証明
$VR_{G,k}(S, LB, v)$の要素の候補は$N_G(v) ¥ S$のみであるため定理 6 は明らかに成り立つ． □

Algorithm 4 がノードを追加した際(6 行目)に k-reduction, vertex-reduction, v-reduction の順に実行することに着目する．本稿ではこれらを束ねて kxv-reduction と呼ぶことにする．定理 4, 定理 5, 定理 6 より，kxv-reduction に関して次に示す定理 7 が成り立つ．

定理 7
グラフ$G(V,E)$, および k-plex Sが与えられたとき, 2 つのノード$u, v \in V$について，uとvのSへの隣接関係が等しいならば，kxv-reduction(v)によって除去されるノードのうち，kxv-reduction(u)によって除去されないノードは, vからの距離が2以下である．

証明
定理 4, 5, 6 より明らかである． □

定理 7 より，k-plex Sへの隣接関係が等しいノード集合において，一度目の除去情報を記憶しておくことで，二度目以降の走査対象が著しく小さくなることがわかる. また，LBの異なる状況においては定理 7 を適用できないが，探索の中でLBが変化する回数は最大 k-plex の大きさを上限値とするため，定理 7 は多くの場面で適用可能である.

定理 7 に基づくアルゴリズムを Algorithm 5 に示す. Algorithm 5 は， Algorithm 4 の 8-10 行目にあたる 3 つの処理を束ねたアルゴリズムである. Algorithm 5 においては，二つのノードu, vがSへの隣接関係が等しいとき，uについての除去アルゴリズム(2-4 行目)を適用した後には，vについての除去アルゴリズムの適用範囲が小さくなる.

最後に Algorithm 5 の時間計算量を以下に示す.

定理 8

Algorithm 5 の時間計算量は，$O\left(d_{avg}^2(|V|+|U|)\right)$である. ただし，$U$は$V \setminus S$のうち$S$への隣接関係が等しいノードの集合である.

証明

Algorithm 5 は，一度のみ$O(|V|d_{avg}^2)$を要する計算をした後，$|U|-1$回$O(d_{avg}^2)$の計算をする. □

Algorithm 5 の計算量に対して，既存手法の該当部分の計算量は$O\left(d_{avg}^4|V||U|\right)$であるため，Algorithm 5 は提案手法の高速化に大きく寄与するといえる.

3.3. ノード選択順序の最適化

定理 7 に示した通り，Algorithm 5 は現在の k-plex Sへの隣接関係が等しいノードを順に探索することで効率的な処理を実現するため，最適なノードの選択順序について考える. 直感的には，提案手法は$G[S]$における接続関係を優先して順序を定める必要がある. その一方で，ベースとなる既存手法 [8]では$V \setminus S$についての次数降順，即ち$d_{G[V\setminus S]}(v)$降順が好ましいと報告されている. 本稿では次の特徴量を基準に，選択順序を決定する.

Feature 1: $d_{G[S\cup\{v\}]}(v)$ (降順)
Feature 2: Sへの隣接関係のハッシュ値
Feature 3: $d_{G[V\setminus S]}(v)$ (降順)
Feature 4: ノード id

またこれら 4 つの特徴量を用いて，ノード選択の優先順位のルールを次のように定める. 不等式は，左側の項が等しいとき一つ右の項により判定することを表す.

ルール A
Feature 1 > Feature 2 > Feature 3 > Feature 4
ルール B
Feature 3 > Feature 1 > Feature 2 > Feature 4
ルール C
Feature 1 > Feature 3 > Feature 2 > Feature 4
ルール D
Feature 3 > Feature 4

ルール A は，提案手法の特徴である隣接関係が等しいノードへの着目を最大限に重視する順序である. またルール B は, 既存手法の特徴であるSの外側における次数に着目する順序，ルール C はその中間といえる順序である. また，ルール D は既存手法と全く同じ順序であり，他の手法との比較のため導入する.

また，上記の順序を探索中に求めるアルゴリズムを Algorithm 6 に示す.

定理 9

Algorithm 6 の時間計算量は，$O\left(|S|d_{avg}\log(|S|d_{avg})\right)$である.

Algorithm 5: kxv-reduction

Input: $G=(V,E)$, 整数 k, LB, k-plex S
　　Sへの隣接関係が等しいノード集合$U \subseteq V \setminus S$
Output: 削減されたグラフG'
1. Uからノードuを一つ選択，$S \leftarrow S \cup \{u\}$;
2. $G \leftarrow$k-reduction(G,S,k);
3. $G \leftarrow$vertex-reduction(G,LB,k);
4. $G \leftarrow$v-reduction(G,S,LB,k,u);
5. $S \leftarrow S \setminus \{u\}$;
6. **foreach** $v \in U \setminus \{u\}$ **do**
7. 　　$S \leftarrow S \cup \{v\}$;
8. 　　$G \leftarrow$k-reduction$(G[N_G(v)],S,k)$;
9. 　　$G \leftarrow$vertex-reduction$(G[N_G(N_G(v))],S,k)$;
10. 　　$G \leftarrow$v-reduction$(G[N_G(v)],S,k)$;
11. 　　$S \leftarrow S \setminus \{v\}$;
12. **return** G;

Algorithm 6: staring-node-ordering(ルール A)

Input: $G=(V,E)$, 整数 k, k-plex S
Output: $V \setminus S$を並び替えた配列V'
1. $V_1 \leftarrow v \in V$ を $d_{G[S\cup\{v\}]}(v)$降順にソート
2. $V_2 \leftarrow V_1$が等しい組について
　　$N_{G[S]}(v)$をハッシュ化しソート
3. $V_3 \leftarrow V_2$が等しい組について
　　$v \in V$ を$d_{G[V\setminus S]}(v)$降順にソート
4. $V_4 \leftarrow V_3$が等しい組についてノード id でソート
5. **return** V_4;

証明

ソートの対象のなるノードは高々$|S| \times d_{avg}$個であり，それらのソートを行うため，定理 9 は明らかである. □

定理 9 より，$|S|, d_{ave} \ll |V|$であるため，Algorithm 6 の計算量は無視できるほど小さいことがわかる.

3.4. 提案手法のアルゴリズム

Algorithm 4, 5, 6 を統合し，提案手法の探索アルゴリズムを Algorithm 7 に示す. Algorithm 7 は，既存手法のボトルネックであるノードを追加する際の枝刈り処理時間を大幅に改善した最大 k-plex 抽出アルゴリズムである.

アルゴリズムの概要：Algorithm 7 について概説する. Algorithm 7 は, 既存手法である Algorithm 4 に Algorithm 5,6 を導入した高速な解法である. Algorithm 7 では，まず前処理の直後にモデルに従いノードをソートする(3 行目). またノードの選択は$V \setminus S$に残っているノードの中で最も先頭にあるものとし(7 行目)，ノードを追加する際の除去には kxv-reduction を用いる(9 行目).

Algorithm 7 は既存手法のボトルネックであるノード追加時の枝刈り処理の重複実行の改善に狙いを定めた高速な探索アルゴリズムであるといえる. Algorithm 7 の最悪時間計算量を定理 10 に示す.

定理 10

Algorithm 7 の最悪時間計算量は，$O(2^{|V|}|V|^2)$である.

証明

定理 8 より，提案手法は既存手法のボトルネックであった$O(|V|^2)$時間を要する処理を$O(|V|)$時間にて解決する. 既存手法の最悪時間計算量は$(2^{|V|}|V|^3)$であるため(2.2.3 節), 定理 10 は成り立つ. □

Algorithm 7: 提案手法

Input: $G=(V,E)$, 整数 k
Output: 最大 k-plex の大きさ

1. $S \leftarrow \emptyset$;
2. $(LB, G) \leftarrow$preprocessing(G,k);
3. $V \leftarrow$staring-node-ordering(G,S);
4. **return** BB-FastReduction(G,S,k,LB);

5. function BB-FastReduction(G,S,k,LB)
6. **if** $V \setminus S = \emptyset$ **then return** $|S|$;
7. $U \leftarrow (V \setminus S)$.same_topology();
8. $G_r \leftarrow G$;
9. $G \leftarrow$kxv-reduction(G,LB,S,k,U);
10. **if** $|\{u | u \in V \setminus S \wedge d_G(u) + k > LB\}| + |S| > LB$ **then**
11. $LB \leftarrow \max(LB, BB(G,S,k,LB))$;
12. $LB \leftarrow \max(LB,$ remove-reduction$(G_r,S,LB,k,U))$;
13. **return** LB;

表 2 データセットの統計情報

データセット	$\|V\|$	$\|E\|$	d_{avg}	κ
soc-brightkite	56.7K	213K	7	0.115
soc-delicious	536K	1.4M	5	0.010
soc-flixstar	2.5M	7.9M	6	0.014
soc-FourSquare	639K	3.2M	10	0.002
soc-gowalla	197K	950K	9	0.023
soc-lastfm	1.2M	4.5M	7	0.013
soc-LiveMocha	104K	2.2M	42	0.014
soc-pokec	1.6M	22.3M	27	0.047
soc-slashdot	70K	359K	10	0.026
soc-youtube	496K	1.9M	7	0.009

定理 10 より，提案手法は既存手法と比較して理論的に高速であることが示された．

4. 評価実験

提案手法の有効性について実データを用いて評価を行う．まず 4.1 節では実験方法や実験に使用したデータセットについて述べ，また 4.2 節にて提案手法の実行時間について評価する． 最後に，4.3 節にて各アルゴリズムの実行時間について細かく検証する．

4.1. 実験設定

表 2 に実験で用いるグラフデータの統計情報を示す．表のκは Fraction of closed triangles を表す．データセットは，Network Repository [10]により公開されている 10 個のソーシャルネットワークを用いて実験を行なった．また，これらのデータは比較可能性の観点から，既存手法 [8]が前処理によって解を出力できなかった，即ち本処理の実行が必要であったデータセットを選定した．既存手法の前処理については付録 C を参照されたい．

実装には C++言語(gcc 8.2.0) を用い，コンパイルオプションは g++ -g -std=c++11 -O3 とした．また，すべての実験は Intel(R) Xeon(R) E5-1620 v4(3.50GHz)，および，16GB のメインメモリを用いて行った．

本稿では最大 k-plex 抽出問題の最先端の手法である既存手法 [8]と，提案手法を比較する．また本実験では k-plex のパラメータ$k = 2,3,4$，および5の場合について評価を行ったが，いずれのパラメータも同様の傾向の結果を示したため，紙面の都合から k=2, 5 の場合についてのみ掲載する．

4.2. 実行時間

表 3 にノードの選択順序モデル毎の実行時間の比較を示す．パラメータ$k = 2$とし，空値は 1 時間以内に解を出力できなかったことを表す．表 3 からもわかるように，ルール A は 7 つのデータで他のルールより優れ，残る 3 つについても僅差である．したがって，これら 4 つのモデルの中ではルール A が最も効果的であることが示唆されている．今後の実験においては提案手法としてルール A を用いる．

表 3 ルール毎の実行時間についての比較

データセット	$k=2$			
	A	B	C	D
soc-brightkite	**0.04**	0.10	0.28	0.38
soc-delicious	**0.31**	0.61	1.4	1.8
soc-flixstar	**898**	1040	-	-
soc-FourSquare	**41**	598	2071	-
soc-gowalla	**21**	29	31	41
soc-lastfm	**33**	70	172	212
soc-LiveMocha	**270**	277	294	309
soc-pokec	24	**21**	59	121
soc-slashdot	261	274	**259**	329
soc-youtube	0.90	**0.68**	4.2	8.2

表 4 各手法の実行時間についての比較[s]

データセット	$k=2$		$k=5$	
	提案手法	既存手法	提案手法	既存手法
soc-brightkite	**0.04**	0.16	**0.09**	0.23
soc-delicious	**0.31**	1.4	**0.34**	1.8
soc-flixstar	**898**	-	**960**	-
soc-FourSquare	**41**	-	**18**	592
soc-gowalla	**21**	32	**25**	390
soc-lastfm	**33**	184	**620**	-
soc-LiveMocha	**270**	326	**37**	40
soc-pokec	**24**	98	**23**	88
soc-slashdot	**261**	299	**380**	470
soc-youtube	**0.90**	12	**1.01**	13

表 4 に実行時間についての比較を示す．表 4 からもわかるように，提案手法は明らかに既存手法より高速である．また，巨大なグラフである soc-FourSquare において最大となる 33 倍の高速化を実現している．これは，提案手法がノード数の多いデータにおいて非常に有効であることを示唆しており，さらに既存手法の実行できないグラフに対して解を出力していることからも，提案手法がより多くのグラフに適用可能であることを示している．また既存手法はルール D（表 3）と比較して僅かながら高速である．これはルール D が効率的な枝刈りを行えず，さらにメモ化処理のオーバーヘッドが上乗せされたためであると考える．

4.3. 枝刈り処理の実行回数

表 5 に提案手法と，既存手法の remove 処理の実行回数の比較を示す．表 5 からわかるように，提案手法は枝刈り処理による除去処理の実行回数が非常に少なくなっている．また提案手法は κ が小さい時ほど効果が大きく，最大で93%程度の削減を実現している．これは，グラフが疎である程ある k-plex に対して同様の構造を持つノードの数が相対的に多くなることに起因すると考える．

5. 関連研究

最大 k-plex 抽出問題に関するこれまでの研究を簡単に述べる．Balasundaram らは， [7]にて初めて最大 k-plex 抽出問題を定義し，それを最大 k-plex 問題の双対問題として貪欲法に基づく素朴な解法を示した．また Moser らは， [11]にて Balasundaram らのアルゴリズムを改良し，ノードの除去によるグラフサイズの削減アルゴリズムを提案した．しかしながら，Moser らの除去アルゴリズムはノードの次数に基づく単純なものでしかなく，実用的な実行速度は得ら

表 5　kxv-reduction における除去回数の比較

データセット	$k = 5$	
	提案手法	既存手法
soc-brightkite	**22K**	52K
soc-delicious	**34K**	109K
soc-FourSquare	**280K**	4.0M
soc-gowalla	**144K**	760K
soc-LiveMocha	**120K**	528K
soc-pokec	**611K**	1.3M
soc-slashdot	**220K**	1.8M
soc-youtube	**12K**	700K

れなかった．これに対して，Xiao らは [12]にてこれまでのアイデアに加えて，各ノードを含む k-plex の大きさの下限値推定を導入し，さらに高速なアルゴリズムを提案し，$|V| < 1000$程度まで有効であることを示した．これらの研究を踏まえ, Gao らは本稿の既存手法にあたる [8]にて 4 つの除去アルゴリズムに基づく高速な探索アルゴリズムを設計した．Gao らの提案した手法は$|V| < 2M$程度まで有効であるが, 疎なグラフに対して低速であるという性質があり，$|V| < 500K$程度のグラフに対しても 1000 秒以内に出力できないことが示されている．本研究は Gao らの示した除去アルゴリズムについて，除去情報をメモ化することにより，疎なグラフデータに対して特に大幅な高速化を実現した．

関連する問題に極大 k-plex 列挙問題 [13]や，最大クリーク問題がある．極大 k-plex 列挙問題は，グラフ中のそれ以上ノードを追加できないような k-plex を全て列挙する問題である．Wu らは [13]にて極大 k-plex 列挙問題を提案し，並列化によるアプローチにより密度が非常に小さなグラフに対してのみ実行可能であることを示した．また Wang らは， [14]にて MapReduce を用いた効率的な並列化を実現し，$|V| < 2M$程度の実世界のグラフに適用可能であることを示した．最大クリーク問題は，グラフ中の最も大きなクリーク，即ち本稿における 1-plex を求める問題であり，最大 k-plex 抽出問題は最大クリーク問題の一般化である．最大クリーク問題は NP 完全であることが知られており，Robson による時間計算量$O(1.1888^n)$の解法 [15]が有名である．

6. おわりに

本稿では，最大 k-plex 抽出問題における最先端の既存手法である, Gao らによる手法 [8]について高速化を試みた．提案手法は，既存手法における枝刈りアルゴリズムの実行に冗長性があることに着目し，枝刈りされるノードをメモ化することにより効率化を図り，我々の評価実験において提案手法は既存手法と比較して最大 33 倍高速であることが示された．今後の課題として，本稿では取り扱わなかった subgraph-reduction(付録 A)の高速化が挙げられる．

謝　辞

本研究の一部は JST ACT-I，JSPS 科研費 JP18K18057 ならびに JST さきがけ (JPMJPR2033) による支援を受けたものである．

参　考　文　献

[1] H. Shiokawa, T. Amagasa and H. Kitagawa, "Scaling Finegrained Modularity Clustering for Massive Graphs," in *IJCAI'19*, 2019.

[2] X. Huang and L. V. S. Lakshamanan, "Attribute-Driven Community Search," vol. 10, no. 9, pp. 949-960, 2017.

[3] M. Onizuka, T. Fujimori and H. Shiokawa, "Graph Partitioning for Distributed Graph Processing," vol. 2, pp. 94-105, 2017.

[4] C. Godsil , G. Royle, Algebraic Graph Theory, Springer, 1949.

[5] I. Bomze, M. Budinich, P. Pardalos and M. Pelillo, "The Maximum Clique Problem," *Handbook of Combinatorial Optimization,* vol. 4, 5 1999.

[6] J. Abello, M. Resende and S. Sudarsky, "Massive Quasi-Clique Detection," in *LATIN'02*, 2002.

[7] B. Balasundaram, S. Butenko and I. Hicks, "Clique Relaxations in Social Network Analysis: The Maximum k-Plex Problem," *Operations Research,* vol. 29, no. 1, pp. 133-142, 2011.

[8] J. Gao, J. Chen, M. Yin, R. Chen and Y. Wang, "An Exact Algorithm for Maximum k-Plexes in Massive Graphs," in *IJCAI'18*, 2018.

[9] S. B. Seidman, "Network Structure and Minimum Degree," *Social Networks,* vol. 5, no. 3, pp. 269-287, 1983.

[10] R. Rossi and N. Ahmed, "The Network Data Repository with Interactive Graph Analytics and Visualization," in *AAAI'15*, 2015.

[11] H. Moser, R. Niedermeier and M. Sorge, "Exact Combinatorial Algorithms and Experiments for Finding Maximum k-Plexes," *Journal of Combinatorial Optimization,* vol. 24, no. 3, pp. 1-27, 2012.

[12] M. Xiao, W. Lin, Y. Dai and Y. Zeng, "A Fast Algorithm to Compute Maximum k-Plexes in Social Network Analysis," in *AAAI'17*, 2017.

[13] B. X. Wu, "A Parallel Algorithm for Enumerating All the Maximal k -Plexes," in *PAKDD Workshops'07*, 2007.

[14] Z. Wang, Q. Chen, B. Hou, B. Sho, Z. Li, W. Pan and Z. Ives, "Parallelizing Maximal Clique and k-Plex Enumeration over Graph Data," *Journal of Parallel and Distributed Computing,* vol. 106, pp. 79-91, 8 2017.

[15] J. M. Robson, “ Finding a maximum independent set in time O(2n/4),” Technical Report, 2001.

付録 A(subgraph-reduction)

本節では，既存手法に用いられる subgraph-reduction, およびそれを用いた remove-reduction (Algorithm 4)について述べる．まず subgraph-reduction の元となる定理を示し，アルゴリズムについて解説する．

定義 2 を用いて，次の定理が成り立つ．

定理 11

グラフ$G(V,E)$，整数k, k-plex $S \subseteq V$，およびノード$v \in V \setminus S$が与えられたとき，$S \cup \{v\}$について以下が成り立つ．定理 2 により除去可能なノードを除去した残りのグラフV^*が$|S| + |V^*| + r_{G,S}(v) \leq LB$を満たすとき，$S \cup \{v\}$は$LB$より大きな k-plex に含まれない．

証明　本稿では省略する([8]を参照されたい．)

定理 11 は，あるノードvを k-plex Sに追加したと仮定して，それにより除去されたグラフからvを評価する．定理 11 を用いて設計されるアルゴリズムを， Algorithm 8 に示す．Algorithm 8 は，候補集合Cの各要素について(1 行目)，一時的にSに追加したとき，v-reduction によって除去された結果が定理 11 を満たすか否かを判定する(4-15 行目)．

付録 B(remove-reduction)

本節では，既存手法に用いられる remove-reduction について説明する．remove-reduction はあるノードvをSに追加しないことを決定した際に実行される．例えば，vが含まれる k-plex を全て探索し終えた際には，v そのものを除去してしまって構わない．このような場面で呼ばれる除去アルゴリズムが remove-reduction である．remove-reduction を Algorithm 9 に示す．remove-reduction は，vertex-reduction と subgraph-reduction を実行した後，親となる再帰関数を実行する．

付録 C(前処理)

本節では，既存手法に用いられる preprocessing について説明する．preprocessing の目的は次の二つである．

1. LBの初期値を与える
2. 探索を始める前に静的にグラフサイズを小さくする

1 について，既存手法はLBが大きいとき程多くの枝刈りを実行可能である．そのため，前処理として素朴なアルゴリズムにより最低限のLB，つまり小さな k-plex を見つけることが重要である．

2 について，既存手法にて提案された除去アルゴリズムのうち，vertex-reduction と subgraph-reduction は起点ノードを指定せずとも実行可能である．そのため，まずこれらを実行することにより，探索を行う前にグラフサイズの削減を図る．Algorithm 10 に preprocessing を示す．Algorithm 10 では，V中の次数が最も少ないノードを除去し，Vが k-plex となるまでこれを繰り返す(16-19 行目)．これにより下限値LBを導出し，それに基づいてグラフサイズを削減する．これらを相互に繰り返し，前述した二つの目的を達成する．

既存手法では，前処理によってグラフサイズが極限まで小さくなり，最大の k-plex を検出できるケースが存在することが示されていた．本稿では提案手法との比較のため，そのようなデータセットは実験対象から除外しており，全てのデータセットに対して動的な探索を行う．

Algorithm 8: subgraph-reduction(G, S, LB, k, C)

Input: $G = (V, E)$,整数 k, LB, k-plex $S \subseteq V$, ノード集合C(ただし$C \cap S = \emptyset$)
Output: 削減されたグラフG'

1. **foreach** $v \in C$ **do**
2. $r_{G,S}(u)$を$\forall u \in (N_G(v) \setminus S) \cup U$について求める;
3. $G''(V'', E'') \leftarrow G, Q \leftarrow \emptyset$;
4. **foreach** $u \in N_{G''}(v) \setminus S$ **do**
5. $c_{G'',S}(u, v)$を求める;
6. **if** $c_{G'',S}(u, v) \leq LB - |S|$ **then** Q.push(u);
7. **while** $Q \neq \emptyset$ **do**
8. $u \leftarrow Q.pop(v)$;
9. $H \leftarrow \left(N_{G''}(u) \cap N_{G''}(v)\right) \setminus S$;
10. **if** $u \in V''$ **then** G''.remove(u);
11. **foreach** w$\in H$ **do**
12. $c_{G'',S}(w, v)$を求める;
13. **if** $c_{G'',S}(w, v) \leq LB - |S|$ **then** Q.push(w);
14. **if**$|N_{G''}(v) \cup \{v\}) \setminus S| + r_{G,S}(v) \leq LB - |S|$ **then**
15. G.remove(v);
16. **break;**
17. **return** G**;**

Algorithm 9: remove-reduction(G, S, LB, k, v)

Input: $G = (V, E)$,整数 k, LB, k-plex $S \subseteq V$,ノード$v \in V$
Output: $G \setminus \{v\}$に対する最大 k-plex の大きさLB

1. $S \leftarrow S \setminus \{v\}, G.remove(v)$;
2. $G \leftarrow$ vertex-reduction(G, LB, k);
3. $G \leftarrow$ subgraph-reduction$(G, S, LB, k, V \setminus S)$;
4. **if** $|\{u | u \in V \setminus S \wedge d_G(u) + k > LB\}| + |S| > LB$ **then**
5. $LB \leftarrow \max(LB, BB(G, S, k, LB))$;
6. **return** LB**;**

Algorithm 10: preprocessing(G, k)

Input: $G = (V, E)$,整数 k,
Output:下限値LBと，削減されたグラフG'

1. $m \leftarrow |V| + 1, LB \leftarrow 0$;
2. **while** $|V| < m$ **do**
3. $m \leftarrow |V|$;
4. $LB \leftarrow \max(LB, LB\text{-Heuristic}(G, k))$;
5. $G \leftarrow$vertex-reduction(G, LB, k);
6. $G \leftarrow$subgraph-reduction$(G, \emptyset, LB, k, V)$;
7. **return** (LB, G);
8.
9. function LB-Heuristic(G, k)
10. **for** $i \leftarrow 1$ *to* $|V|$ **do**
11. $v \leftarrow \text{argmin}_{u \in V} d_G(u)$;
12. **if** $d_G(v) + k \geq |V|$ **then break**;
13. G.remove(v);
14. **return** $|V|$;