Vol.2020-AL-180 No.23
2020/11/26

グラフスペクトルによる多項式時間最大クリーク抽出

大戸 康紀

概要：本論文では、NP 完全問題の一つである k-Clique 問題の最適化バージョンである最大クリーク問題が多項式時間で解けることを証明します。入力グラフ G を辺に有理数の重みを持つ Undirected graph へ拡張します。すべての辺が重み 1 であり、すべての頂点が他の頂点との間に辺を持つ Subgraph をクリークとします。最大クリークの一つを多項式時間で抽出するアルゴリズムを構成する上で必要な証明を行うとともに、そのアルゴリズムを擬似コードの形で与えます。Adjacency matrix の -1 以下の固有値の個数を $k_m(G)$ とすると、Cauchy Interlacing Theorem により最大クリークの頂点数 k は $k_m(G)+1$ 以下となります。アルゴリズムでは、$k_m(G)$ の変化を用いて、頂点を削除したときに最大クリークサイズを変化させない頂点を削除していきます。このアルゴリズムの計算量は $O(n^8)$ です。この結果は complexity class NP と P が同じであることを示しています。

Polynomial-time Extraction of the Maximum Clique Using Eigenvalue Relation

1. Introduction

本論文では、NP 完全問題の一つである k-クリーク問題の最適化バージョンである最大クリーク問題が多項式時間で解けることを証明します。

最大クリーク問題における計算量の下界を求めるアプローチでは、計算量を $O(2^{f(n)})$ と仮定した上で、木探索時のバックトラッキングのショートカットを効率的に行うことで、$f(n)$ の下限を求める研究が行われてきました [1]。しかし、これらは離散数学の範囲での試みであり、他の数学領域の手法を用いることはありませんでした。スペクトルグラフ理論では、グラフの特性とその特性多項式、固有値、固有ベクトル、隣接行列、およびラプラシアン行列との関係を研究しています [2]。Cauchy Interlacing Theorem では対称行列とその主部分行列の間の固有値の関係を述べており、この関係を用いたグラフとその固有値間の関係についての研究があります [3]。しかし、グラフスペクトルを使用して NP 問題を多項式時間で解こうとする研究はありませんでした。この理由は、一般的にはスペクトルは実数であり、コンピュータで正確に扱うことができないためです。

本論文では、入力グラフ G を辺に有理数の重みがついたつ Undirected graph へ拡張します。すべての辺が重み 1 であり、すべての頂点が他の頂点との間に辺を持つ Subgraph をクリークとします。最大クリークの一つを多項式時間で抽出するアルゴリズムを構成する上で必要な証明を行うとともに、そのアルゴリズムを擬似コードの形で与えます。Adjacency matrix の -1 以下の固有値の個数を $k_m(G)$ とすると、Cauchy Interlacing Theorem により最大クリークの頂点数 k は $k_m(G)+1$ 以下となります。アルゴリズムでは、$k_m(G)$ の変化を用いて、頂点を削除したときに最大クリークサイズを変化させない頂点を削除していきます。$k_m(G)$ を求めるために、行列 A を Frobenius normal form [4], [5], [6] に変形し、有理数計算の範囲内で固有方程式の係数を求めたのち、Sturm's theorem [7] を用いて A の固有値の個数を求めます。本紙のアルゴリズムは多項式時間で最大クリークを一つ出力します。このアルゴリズムの計算量は $O(n^8)$ です。この結果は complexity class NP と P が同じであることを示しています。

本論文の残りの部分では、Section 2 において、本論文で使用する定義を示します。Section 3 において、最大クリークを抽出するアルゴリズムとその計算量を示します。最後に、Section 4 本論文のまとめを行います。

[1] 情報処理学会
IPSJ, Chiyoda, Tokyo 101–0062, Japan

2. Definition

ここでは、本論文において用いる定義を示します。本紙では、入力するグラフは多重辺とループを含まない、辺の重みが 1 であり、独立頂点を持たないものとします。また、最大クリークの抽出過程において扱う Weighted graph は、辺の重み w を有理数とし、$0 < w \leq 1$ の範囲に限定します。

Definition 1. *Graph* G を集合 V と V^2 の部分集合 E の組 (V, E) とします。V の元を頂点、E の元 (v_a, v_b), $v_a, v_b \in V$ を辺と呼びます。V は有限であるとします。n は *Graph* G の頂点の個数とします。$V = \{v_1, \cdots, v_n\}$ と整列しておきます。

Definition 2. 関数 $w : e \to \mathbb{R}_+$ を G 上の重みとします。*Graph* $G = (V, E, w)$ を *Weighted graph* とします。

Definition 3. *Weighted graph* G の *Adjacency matrix* A は $n \times n$ 行列であり、次を満たすものとします。

(1) $\{v_i, v_j \in E\}$ ならば $A_{i,j} = w(\{v_i, v_j\})$,

(2) $\{v_i, v_j \notin E\}$ ならば $A_{i,j} = 0$.

Graph G はループを含まない *Undirected graph* とします。よって、*Adjacency matrix* A は非負値対称行列で、その対角成分は 0 となります。このことから、対角成分が 0 となる非負値対称行列と、多重辺とループを含まない *Undirected graph* は同型を除いて一対一関係となります。

Definition 4. $m \times m$ の正方行列 *Principal submatrix* P は、$n \times n$ の正方行列 M における任意の $n - m$ 個の $i, \cdots, j$ 番目の行と列を除去した *Submatrix* です。

Definition 5. *Graph* $G = (V, E)$ における *Induced subgraph* $G_i = (V_i, E_i)$ は、頂点 $v_a, v_b \in V_i \in V$ において、$(v_a, v_b) \in E$ のとき、かつそのときに限り、$(v_a, v_b) \in E_i$ とします。

よって、*Induced subgraph* G_i の *Adjacency matrix* A_i は、*Graph* G の *Adjacency matrix* A の *principal submatrix* となります。

Definition 6. *Complete graph* K を、*Weighted graph* $G = (V, E, w)$ において、$E = V^2$ かつ $w : e \to 1$ となるグラフとします。

Definition 7. *Weighted graph* $G = (V, E, w)$ において、V の部分集合 C がクリークを形成するとは、頂点集合 C が誘導する G の部分グラフが *Complete graph* となることです。複数存在するクリークにおいて、クリークを構成する頂点数が最も大きいものを最大クリークと呼びます。

3. Proof

この Section では、最大クリークの一つを多項式時間で抽出するアルゴリズムを構成する上で必要な証明を行うとともに、そのアルゴリズムを擬似コードの形で与えます。

Input graph は多重辺とループを持たず、辺の重みは 1 です。入力データは、頂点番号とその組である辺の配列で構成されます。入力データから独立した頂点を削除します。独立した頂点を削除したのち、頂点が無くなる場合、本論文のアルゴリズムは任意の頂点を出力して終了します。そして、独立した頂点を削除した入力データから、辺の重みが $w \in \mathbb{Q}_+$、$0 < w \leq 1$ の Weighted graph を表す Adjacency matrix を生成します。

Proposition 8. サイズ n の *Complete graph* K_n の *Adjacency matrix* A における固有値は、重複を含めると、固有値 -1 が $n-1$ 個、固有値 $n-1$ が 1 個となります。

Proof. 正方行列 A における固有値 λ は、行列式を $|A|$、対角成分が 1 で他の成分が 0 となる単位行列 I を用いて、$|A - \lambda I| = 0$ を満たす実数となります。A は Definition 6 により、対角成分が 0、その他の成分が 1 となる対称行列です。サイズ n の対称行列は重複を含めて n 個の実数からなる固有値を持ちます。$rank(A)$ を A の一次独立な行ベクトルの最大本数とします。このとき、固有値 λ の重複度は $n - rank(A - \lambda I)$ となります。$-1, n-1$ を $|A - \lambda I|$ に代入すると、いずれも 0 となります。また、$n - rank(A - \lambda I)$ は $n-1, 1$ となることから、A は 固有値 -1 を $n-1$ 個、固有値 $n-1$ を 1 個持ちます。よって、Proposition 8 が成り立ちます。 □

Matrix A を Graph G の Adjacency matrix とします。Graph G の最大クリークサイズを k とします。また、A の固有値のうち、-1 以下であるものの重複を含めた個数を $k_m(G)$ とします。このとき、k と $k_m(G)$ の関係を示すために次の Theorem を用います。

Theorem 9. *(Cauchy Interlacing Theorem,[3]):* A をサイズ n の対称行列, B をサイズ $m < n$ の対称行列とします。A の固有値を重複を含めて $\lambda_1 \geq \cdots \geq \lambda_n$ とします。同様に B の固有値を $\mu_1 \geq \cdots \geq \mu_m$ とします。もし、B が A の *Principal submatrix* であれば、任意の $0 < i \leq m$ において以下の *Inequation* が成り立ちます。

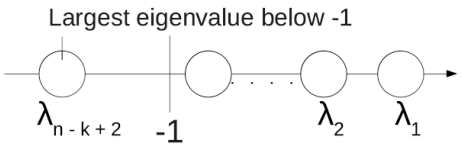
$$\lambda_i \geq \mu_i \geq \lambda_{n-m+i}. \tag{1}$$

この Theorem から以下が成り立ちます。

Corollary 10. 辺を持つ *Graph* G の *Adjacency matrix* A は、重複を含めると、-1 以下に一つ以上の固有値が存在します。また、1 以上にも一つ以上の固有値が存在します。

Proof. 前提より、Graph G は辺を持つため、最小頂点からなる Graph はサイズ 2 の Complete graph K_2 となります。Proposition 8 より、K_2 の固有値は ± 1 となります。Inequation 1 より、A の最小となる固有値 λ_{min} は -1 以下となります。また、最大となる固有値 λ_{max} は 1 以上と

図 1 $n-k+2$ 番目に大きい固有値以降が -1 以下となるとき、Inequation 1 を満たします。



なります。よって、Corollary 10 が成り立ちます。 □

これらから次の不等式が成り立ちます。

Lemma 11. *Graph G の最大クリークサイズ k は $k \leq k_m(G)+1$ を満たします。*

Proof. Graph G の頂点数を n とします。Graph G の Adjacency matrix A は対称行列となります。Graph G の最大クリーク G_c は Complete graph であり、その Adjacency matrix A_c は対称行列となります。ここで、クリーク G_c は G の Induced subgraph になるため、A_c は A の Principal submatrix となります。

$k=n$ の場合: Proposition 8 より、サイズ n の Complete graph K_n の Adjacency matrix における固有値は、重複を含めると、固有値 -1 を $n-1$ 個、固有値 $n-1$ を 1 個となることから、$k_m(G)=n-1$ となります。よって、$k \leq (n-1)+1$ となり、Lemma11 が成り立ちます。

$k=1$ の場合: Graph G が独立頂点のみで構成されているとき、A の固有値は n 個の固有値 0 となるため、$k_m(G)=0$ となります。よって、$1 \leq 0+1$ となり、Lemma11 が成り立ちます。Graph G が辺を持つとき、Corollary 10 より、$k_m(G) \geq 1$ となります。よって、$1 \leq 1+1$ となり、Lemma11 が成り立ちます。

$1<k<n$ の場合: Corollary 10 より $\lambda_n \geq -1$ となります。Proposition 8 より、行列 A_c の固有値を重複を含めて μ_j, $0<j\leq k$ とすると、$\mu_1=k-1$, $\mu_i=-1$, $1<i\leq k$ となります。固有値 -1 について Inequation 1 を適用すると、$1<i\leq n$ において、$\lambda_i \geq -1 \geq \lambda_{n-k+i}$ となります。$i=2$ に着目すると、Inequation は $\lambda_2 \geq -1 \geq \lambda_{n-k+2}$ となります。よって、$n-k+2$ 番目に大きい固有値以降が -1 以下となるとき、Inequation 1 を満たします (Figure 1)。よって、-1 を超える固有値の個数 $n-k_m(G)$ において、以下が成り立ちます。

$$\begin{aligned} n-k_m(G) &\leq n-k+2-1, \\ -k_m(G) &\leq -k+1, \\ k &\leq k_m(G)+1. \end{aligned} \tag{2}$$

よって、Lemma 11 が成立します。 □

Graph G の Adjacency matrix A における -1 以上の固有値の重複を含めた個数を $k_p(G)$ とします。

Lemma 12. *Graph G において、G から頂点を一つ削除した後の Graph を G' とします。頂点削除後の Graph G' の Adjacency matrix A' における固有値の個数の組 $(k_m(G'),k_p(G'))$ は、$(k_m(G)-1,k_p(G))$ もしくは $(k_m(G),k_p(G)-1)$ となります。*

Proof. Graph G の頂点数を n、Adjacency matrix を A、頂点削除後の Graph を G'、Adjacency matrix を A' とします。A は $n\times n$、A' は $n-1\times n-1$ の正方行列となります。Graph G' は G の Induced subgraph となります。このことから、A' は A の principal submatrix となります。

Graph G が独立頂点のみのとき、A の固有値は 0 のみとなります。このとき、$k_m(G)=0$ であり、Graph G' についても $k_m(G')=0$ となります。

Graph G が辺を持つとき、Corollary10 から、A は -1 以下の固有値、1 以上の固有値を持ちます。-1 以下の固有値の個数を m とします。1 以上の固有値が存在することから、$m<n$ となります。A の -1 以下の固有値を $\lambda_1 \leq \cdots \leq \lambda_m$ とします。A' の固有値において、小さいものから m 個を取り出して、$\mu_1 \leq \cdots \leq \mu_m$ とします。Inequation 1 より、$\lambda_i \geq \mu_i \geq \lambda_{i+1}$ となります。よって、$\lambda_m \leq \mu_m$ となり、$\mu_m \leq -1$ もしくは $-1<\mu_m$ となります。$m=1$ のとき、$k_m(G')=k_m(G)$ もしくは $k_m(G')=k_m(G)-1$ となります。$1<m$ のとき、μ_{m-1} は存在しますが、Inequation 1 より、$\mu_{m-1} \leq \lambda_m \leq -1$ となります。よって、頂点削除により $k_m(G)$ は変化しないか、1 減少します。

頂点削除に伴って Graph G の頂点数が 1 減少するため、$k_m(G'))=k_m(G)$ のとき $k_p(G'))=k_p(G)-1$、$k_m(G'))=k_m(G)-1$ のとき $k_p(G'))=k_p(G)$ となります。よって、Lemma 12 が成立します。 □

Lemma 11 より、Graph G の最大クリークサイズ k は $k \leq k_m(G)+1$ となります。また、Lemma 12 より、Graph G から頂点を削除したとき、$k_m(G)$ は変化しないか、1 減少します。本論文のアルゴリズムは、頂点削除における $k_m(G)$ の変化を観察することで、Graph G の最大クリークサイズを変更しないような頂点集合を同定して、この頂点集合に属する頂点を削除することで最大クリークを抽出します。

固有値は一般的には実数であり、コンピュータで正確に扱うことができません。しかし、-1 以下の固有値の個数を求めるには有理数の範囲の計算で可能です。そして、有理数については、二つの整数を用いて表現することができて、その計算量は多項式時間となります。

Proposition 13. 有理数の四則演算は多項式時間で計算できます。

Proof. 有理数 $r \in \mathbb{Q}$ を a/b として、二つの整数の組 $(a,b) \in \mathbb{N}^2$, $b \neq 0$ で表すことができます。二つの有理数 $r_1=(a_1,b_1)$, $r_2=(a_2,b_2)$ の四則演算を次に示します。

$$(a_1, b_1) \pm (a_2, b_2) = (a_1 b_2 \pm b_1 a_2,\ d_1 d_2), \tag{3}$$
$$(a_1, b_1) \times (a_2, b_2) = (a_1 a_2,\ b_1 b_2), \tag{4}$$
$$(a_1, b_1) \div (a_2, b_2) = (a_1 b_2,\ b_1 a_2). \tag{5}$$

計算結果において、a, b の最大公約数 m が 1 を超えるとき、$(a, b) \leftarrow (a/m, b/m)$ とします。最大公約数を求めるユークリッドの互除法の計算量が $O(log_2 n)$ となることから、有理数の四則演算の計算量は多項式時間となります。よって、Proposition 13 が成り立ちます。 □

多項式時間アルゴリズム内で有理数の四則演算を用いても、当該アルゴリズムが多項式時間で解けることから、本論文では、有理数の四則演算に伴う計算量を $O(1)$ として扱います。

辺の重み w が有理数の範囲にあるとき、Graph G における $k_m(G)$ をアルゴリズムで構成できます。Graph G の辺の重み w が有理数の範囲にあることから、Adjacency matrix A は有理数からなる正方行列となります。そして、A の特性方程式の係数は有理数となります。特性方程式の係数を求めるために、有理数の四則演算の範疇で、正方行列 A を Frobenius normal form [4], [5] に変換します。Frobenius normal form は一つないしは複数の対角線ブロック行列からなります。よって、A の固有方程式は対角線ブロック行列の固有方程式の積になります。個々の対角線ブロック行列における特性方程式は重根を持ちません。-1 以下の固有値の個数を求めるために、各々の対角線ブロック行列において Strum Theorem [7] を使用します。各対角線ブロック行列における -1 以下の固有値の個数の和が A の -1 以下の固有値の個数となります。

以下に、この計算が多項式時間で行うことを示します。

Lemma 14. *サイズ n の Graph G の Adjacency matrix A の -1 以下の固有値の個数を求める関数 $k_m(G)$ の計算量は $O(n^3)$ となります。*

Proof. Graph G の Adjacency matrix A を有理数の範囲内で $A \leftarrow S^t A S$ として表される相同変換を用いて、Frobenius normal form に変換します。対角線ブロック行列の個数を m、各々のブロック行列を F_i、ブロック行列のサイズを n_i とします。変換によって得られる Frobenius normal form は以下になります。

$$S^t A S \rightarrow \begin{pmatrix} F_1 & & & & \\ & F_2 & & * & \\ & & \ddots & & \\ 0 & & & & \\ & & & & F_m \end{pmatrix}. \tag{6}$$

各対角ブロック F_i は次のようになります。

$$F_i = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & -a_0 \\ 1 & 0 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & 0 & \cdots & 0 & -a_2 \\ . & . & . & \cdots & . & . \\ 0 & 0 & 0 & \cdots & 1 & -a_{n_i-1} \end{pmatrix}. \tag{7}$$

A の特性多項式はブロック F_i の特性多項式 C_i の積 $\prod_{i=1}^{m} C_i$ となります。ブロック F_i の特性多項式は、$C_i = a_0 + \sum_{k=1}^{n_i-1} a_k \lambda^k + \lambda^{n_i} = 0$ となります。これらの方程式は n_i 個の根を持ちますが、重根はありません。Frobenius normal form への変換により、各々の対角ブロック F_i における特性方程式の係数を取得するまでにかかる計算量は $O(n^3)$ となります [6]。

次に、各 F_i の特性多項式が重根を持たないことから、Strum Theorem [7] を使用します。サイズが n_i の特性多項式で始まる多項式の有限列を以下のように算出します。

$$\begin{aligned} p_0(x) &= p(x), \\ p_1(x) &= p'(x), \\ p_j(x) &= -rem(p_{j-2}(x), p_{j-1}(x)). \end{aligned} \tag{8}$$

ここで、$rem(a, b)$ は a, b を多項式として、a を b で割ったときの剰余とします。多項式列の最後の方程式 $p_{m_j}(x)$, $m_j \leq n_i$ は非ゼロの定数です。ユークリッドの互除法を使用して多項式の剰余を取得します。$p_0(x), \cdots, p_{m_j}(x)$ を Strum 列の任意の多項式列とします。数式に対するユークリッドの互除法の計算量は $O(n_i \, log \, n_i)$ となります。多項式の数 m_j は n_i 以下となるため、多項式列を求めるための計算量は $O(n_i^2 \, log \, n_i)$ となります。$\sigma_i(\alpha)$ を、ゼロを無視した、この多項式列の符号変化の数として定義します。$\sigma_i(\infty)$ は各多項式における一番高い次数を持つ項の係数の符号変化の数の数となります。a を超える値となる根の個数は、$\sigma_i(\infty) - \sigma_i(a)$ です。n_i 次多項式を高次から順に計算すると計算量は $O(n_i)$ となります。多項式の数 m_j は n_i 以下となるため、$\sigma_i(\alpha)$ の計算量は $O(n_i^2)$、$\sigma_i(\infty)$ の計算量は $O(n_i)$ となります。各対角ブロック F_i の -1 以下の固有値の個数は、$n - (\sigma_i(-1) - \sigma_i(\infty))$ となります。Graph G の Adjacency matrix A の -1 以下の固有値の個数は、各対角ブロック F_i における -1 以下の固有値の個数を足し合わせたものとなります。よって、$k_m(G)$ は $n - \sum_{i=1}^{m} (\sigma_i(-1) - \sigma_i(\infty))$ となります。

以上より、Graph G の Adjacency matrix A の -1 以下の固有値の個数 $k_m(G)$ を求める関数の計算量は $O(n^3)$ となります。よって、Lemma 14 が成り立ちます。 □

Lemma 11 より、Graph G の最大クリークサイズ k は $k \leq k_m(G) + 1$ となります。ゴールは頂点削除によって最大クリークを一つ抽出することです。よって、最大クリークに属さない頂点を削除することで、-1 未満の固有値を

除去していき、$k = k_m(G) + 1$ を目指すことになります。

Graph が一つ以上の Complete graph からなるとき、-1 未満の固有値の個数が 0 になります。よって、最大クリークに影響しないような辺を加えることで、-1 未満の固有値の個数が 0 になるときに Complete graph が一つとなるように、固有値の個数をコントロールします。

Lemma 15. *Graph G の辺が無い頂点間に $0 < w < 1$ となる重み w の辺 e を加えた Graph を G' とします。このとき、以下の関係が成り立ちます。*

$$\begin{aligned}(k_m(G'), k_p(G')) \quad &\text{is} \quad (k_m(G) - 1, k_p(G) + 1),\\ &\text{or} \quad (k_m(G), k_p(G)),\\ &\text{or} \quad (k_m(G) + 1, k_p(G) - 1).\end{aligned} \tag{9}$$

Proof. Graph G のサイズを n とします。頂点 $v_i, v_j \in V$ for $0 < i, j \leq n$, $i \neq j$ 間に辺が存在しないとします。Graph G から頂点 v_i を削除した Graph を G'' とします。Lemma 12 より、以下の関係が成り立ちます。

$$\begin{aligned}(k_m(G''),\, k_p(G'')) \quad &\text{is} \quad (k_m(G) - 1,\, k_p(G)),\\ &\text{or} \quad (k_m(G),\, k_p(G) - 1).\end{aligned} \tag{10}$$

Graph G'' に頂点 v_i と、頂点 v_i, v_j 間に辺 e を加えた Graph を G' とします。よって、Graph G' は G に辺 e を加えた Graph となります。Lemma 12 より以下が成り立ちます。

$$\begin{aligned}(k_m(G''),\, k_p(G'')) \quad &\text{is} \quad (k_m(G') - 1,\, k_p(G')),\\ &\text{or} \quad (k_m(G'),\, k_p(G') - 1).\end{aligned} \tag{11}$$

よって、以下が成り立ちます。

$$\begin{aligned}(k_m(G'), k_p(G')) \quad &\text{is} \quad (k_m(G'') + 1,\, k_p(G'')),\\ &\text{or} \quad (k_m(G''),\, k_p(G'') + 1).\end{aligned} \tag{12}$$

固有値の個数の関係 10, 12 より、以下が成り立ちます。

$$\begin{aligned}(k_m(G'), k_p(G')) \quad &\text{is} \quad (k_m(G'') + 1,\, k_p(G'')) \quad = \quad (k_m(G) + 1, k_p(G) - 1),\\ &\text{or} \quad (k_m(G'') + 1,\, k_p(G'')) \quad = \quad (k_m(G), k_p(G)),\\ &\text{or} \quad (k_m(G''),\, k_p(G'') + 1) \quad = \quad (k_m(G), k_p(G)),\\ &\text{or} \quad (k_m(G''),\, k_p(G'') + 1) \quad = \quad (k_m(G) - 1, k_p(G) + 1).\end{aligned} \tag{13}$$

よって、Lemma 15 が成立します。 □

Corollary 16. *Graph G の辺が無い頂点間に $0 < w < 1$ となる重み w の辺 e を加えても、最大クリークサイズは変化しません。*

Proof. Definition 7 より、クリークは重み 1 の辺からなっています。追加する辺 e の重み w は 1 ではないため、最大クリークは変化しません。よって、Corollary 16 が成立します。 □

Algorithm 1: 最大クリークを抽出する前の事前準備方法。Graph G において、辺が無いすべての頂点間において、$k_m(G)$ の変化を順次確認していきます。そして、$k_m(G)$ が小さくなる場合に、対象となる頂点間に重み 1/2 の辺を順次挿入していきます。

```
Function prepare(G):
    A ← Adjacency matrix of G
    k_m ← k_m(G)
    foreach v_i, v_j ∈ V such that (v_i, v_j) ∉ E do
        G' ← graph which is added an edge with weight
          1/2 to G
        k'_m ← k_m(G')
        if k'_m < k_m then
            G ← G'
            k_m ← k'_m
        end
    end
    return G
```

Lemma 15 により、Graph G の辺が無い頂点間に $0 < w < 1$ となる重み w の辺 e を加えたとき、$k_m(G)$ は変化する可能性があります。しかしながら、Corollary 16 により、辺 e を加えても最大クリークサイズは変化しません。

Algorithm 1 において、Graph G から最大クリークを抽出する前の事前準備方法を示します。Lemma 15 により、Input graph に対して、重み 1/2 の辺 e を辺が無い頂点間に挿入したとき、$k_m(G)$ は変化しないか、1 増加するか、1 少なくなります。そこで、辺が無いすべての頂点間において、$k_m(G)$ の変化を順次確認していきます。そして、$k_m(G)$ が小さくなるたびに、対象となる頂点間に重み 1/2 の辺を挿入していきます。これにより、辺の挿入は複数本になる場合があります。なお、辺を挿入することで k_m(G) が変化したとしても、Corollary 16 より、最大クリークへの影響はありません。Algorithm 1 の計算量は $O(n^5)$ となります。

事前準備を行った Graph を G_s とします。Graph G_s に重み 1/2 の辺を追加しても、$k_m(G_s)$ は減少しません。

Complete graph $K_{n_i} = (V_i, E_i)$ for $0 < i \leq m$, $1 < V_i$ とします。$G_u = (\bigcup_{i=1}^{m} V_i, \bigcup_{i=1}^{m} E_i)$ とします。

Corollary 17. *Graph G_u の Adjacency matrix A_u における -1 未満の固有値は存在しません。*

Proof. 前提より、Graph G_u は独立頂点を持ちません。Proposition 8 より、サイズ n_i の Complete graph K_{n_i} の Adjacency matrix A_i における固有値は、重複を含めると、固有値 -1 を $n_i - 1$ 個、固有値 $n_i - 1$ を 1 個持ちます。よって、Graph G_u における Adjacency matrix A_u における -1 以下の固有値の数は、重複を含めると、$\sum_{i=1}^{m} n_i - 1$ 個となります。よって、-1 未満の固有値は存在しません。このことから、Corollary 17 が成立します。 □

Corollary 18. *Graph G_u から任意の頂点を削除すると、*

$k_m(G_u)$ が減少します。

Proof. Graph G_u のいずれかの頂点を削除すると、G_u を構成する K_{n_i} のいずれかの頂点を削除することになります。この結果、A_u の -1 となる固有値の一つが減少します。このことから、Corollary 18 が成立します。 □

Lemma 19. *Graph* G_s は G_u を含みません。

Proof. Graph G_u の頂点数を n とします。Graph K_i の Adjacency matrix を A_i、サイズを n_i とします。Graph G_u の Adjacency matrix A は A_i を対角ブロックとして持つ行列となります。Proposition 8 より、固有値 -1 の個数は $\sum_{i=1}^{m}(n_i-1)$ 個、-1 を超える固有値の個数は m 個となります。

$m=1$ とのき G_u は Complete graph となるため、任意の二つの頂点 $v_i, v_j \in V, i \neq j$ 間に辺が存在します。このため、Algorithm 1 によって辺を挿入されないことから、G_s と G_u は同じ Graph となります。

次に、$m \neq 1$ のとき、G_s が Complete graph となると仮定します。Graph G_u の Subgraph である任意の二つの Complete graph G_i, G_j, $i \neq j$, $0 < i, j \leq m$ における頂点集合 V_i, V_j において、任意の頂点 $v_i \in V_i$ と $v_j \in V_j$ の間に辺はありません。Algorithm 1 によって頂点 v_i, v_j 間に重み 1/2 の辺を挿入することにより、Graph G_u の Adjacency matrix A_u において、$A_u + I$ の一次独立な行ベクトルが 2 本減少します。よって、固有値 -1 の個数が 2 つ減少して $\sum_{i=1}^{m}(n_i-1)-2$ 個となります。そして、Lemma 15 より、-1 未満の固有値の個数が 2 つ増加する、もしくは -1 を超える固有値の個数が 2 つ増加することはありません。よって、-1 未満の固有値の個数が 1 つ増加し、-1 を超える固有値の個数が 1 つ増加します。以上より、$m \neq 1$ のとき、G_s が Complete graph となる仮定は矛盾します。

以上より、Lemma 19 が成り立ちます。 □

Graph G_s において頂点を削除した Graph を G_t とします。$k_m(G_t) = k_m(G_s)$ のとき、Graph G_t に重み 1/2 の辺を挿入することで、$k_m(G_t)$ が減少する場合があります。$k_m(G_t) = k_m(G_s)+1$ のとき、Graph G_t に重み 1/2 の辺を挿入しても、$k_m(G_t)$ は減少しません。

Lemma 20. *Graph* G_t *を、Graph* G_s *に対して頂点の削除を行った Graph とします。*$k_m(G_t) = k_m(G_s)+1$ *のとき、*G_t *に重み* w *for* $0 < w < 1$ *の辺を挿入しても、*$k_m(G_t)$ *は減少しません。*

Proof. Graph G_s から頂点 $v \in V_s$ を削除して、頂点削除した Graph G_t において $k_m(G_t) = k_m(G_s)-1$ となるとします。Graph G_u を G_t に重み w for $0 < w < 1$ の辺 e を挿入した Graph とします。Graph G_u に頂点 v を追加した Graph を G_v とします。Graph G_u は G_s に辺 e を加えた Graph となります。

Algorithm 2: Graph G_s において、頂点 $v_i \in V$ を削除する方法。頂点を削除したさいに $k_m(G_s)$ が変化しないとき、辺が存在しない頂点ペア間に重み 1/2 の辺 e を挿入することで、$k_m(G_s)$ が減少する場合に辺 e を挿入します。

```
Function delete_vertex(G_s, v):
    k_{m_s} ← k_m(G_s)
    G_s ← graph which is deleted v of G_s
    if k_m(G_s) = k_{m_s} then
        foreach v_i, v_j ∈ V such that (v_i, v_j) ∉ E do
            G'_s ← graph which is added an edge with
              weight 1/2 to G_s
            k'_{m_s} ← k_m(G'_s)
            if k'_{m_s} < k_{m_s} then
                return G_s
            end
        end
    end
    return G_s
```

$k_m(G_u) = k_m(G_t)-1$ となると仮定します。Lemma 15 より、Graph G_s に辺 e を加えたときの $k_m(G_s)$ の変化幅は ± 1 となります。

$k_m(G_v) = k_m(G_u)-1$ となると仮定すると、$k_m(G_v) = k_m(G_t)-2 = k_m(G_s)-3$ となります。よって、$k_m(G_v) = k_m(G_u)$ の仮定は成立しません。

$k_m(G_v) = k_m(G_u)$ となると仮定すると、$k_m(G_v) = k_m(G_t)-1 = k_m(G_s)-2$ となります。よって、$k_m(G_v) = k_m(G_u)$ の仮定は成立しません。

$k_m(G_v) = k_m(G_u)+1$ となると仮定すると、$k_m(G_v) = k_m(G_t) = k_m(G_s)-1$ となります。Graph G_s は重み w for $0 < w < 1$ の辺を加えても、$k_m(G_s)$ が減少しないことから、よって、$k_m(G_v) = k_m(G_u)+1$ の仮定は成立しません。

以上より、$k_m(G_u) = k_m(G_t)-1$ の仮定は成立しません。よって、Lemma 20 が成立します。 □

Algorithm 2 において、Graph G_s において、頂点 $v_i \in V$ を削除する方法を示します。Lemma 12 より、頂点を削除することで、$k_m(G_s)$ は変化しないか 1 つ減少します。$k_m(G_s)$ が 1 つ減少する場合、Lemma 20 より、辺の挿入により $k_m(G_s)$ が更に減少することはありません。頂点を削除したさいに $k_m(G_s)$ が変化せず、辺が存在しない頂点間に重み 1/2 の辺 e を挿入することで $k_m(G_s)$ が減少するとき、辺 e を挿入します。Lemma 12 より、辺の挿入による $k_m(G_s)$ の減少幅は最大 1 であるため、挿入する辺はたかだか一つとなります。Algorithm 2 の計算量は $O(n^5)$ となります。

Algorithm 2 によって頂点削除した Graph に重み 1/2 の辺を追加しても、$k_m(G_t)$ は減少しません。よって、頂点削除後の Graph は G_s となります。

Graph G_s において、頂点削除によって最大クリークサ

Algorithm 3: Graph G_s に対する頂点削除において、$k_m(G_s)$ を変化させない頂点集合 V_a を取得する方法。Graph G_s から頂点削除において、$k_m(G_s)$ が変化しない頂点を順番に削除していきます。削除した頂点集合を V_a とします。頂点削除後の Graph を G_r としたとき、$k_m(G_r) = k_m(G_s)$ となります。

```
Function remove_peripheral_vertex_set(G_s):
    V_t ← V_s
    V_a ← ∅
    k_{m_s} ← k_m(G_s)
    foreach v ∈ V_t do
        G_r ← delete_vertex(G_s, v)
        if k_m(G_r) = k_{m_s} then
            G_s ← G_r
        end
    end
    return (G_s, V_a)
```

イズを変更しないような頂点集合を得るのに必要な証明を行います。

Algorithm 3 において、Graph G_s に対する頂点削除に対して、$k_m(G_s)$ を変化させない頂点集合 V_a を取得する方法を示します。Graph G_s から頂点削除において、$k_m(G_s)$ が変化しない頂点を順番に削除していきます。削除した頂点集合を V_a とします。頂点削除後の Graph を G_r としたとき、$k_m(G_r) = k_m(G_s)$ となります。さらに G_r のいずれの頂点を削除しても $k_m(G_r) = k_m(G_s) - 1$ となります。Algorithm 3 の計算量は $O(n^6)$ となります。

Graph G_s に対する頂点削除において、Algorithm 3 による、$k_m(G_s)$ を変化させない頂点集合 V_a を取得します。G_r, V_a において、以下の 3 つの場合があります。

(1) G_r が Complete graph のとき,

(2) G_r が Complete graph ではなく、かつ $V_a = \emptyset$ のとき,

(3) G_r が Complete graph ではなく、かつ $V_a \neq \emptyset$ のとき.

頂点削除によって独立頂点が生じた場合、独立頂点は頂点集合 V_a、V_b のいずれか、もしくは両方に所属することになります。よって、G_r には独立頂点は含まれません。

Case 1: のとき、以下を満たします。

Lemma 21. *Graph* G_r が *Complete graph* のとき、G_r は *Graph* G_s の最大クリークとなります。

Proof. Lemma 11 より、Graph G_s の最大クリークサイズ k は $k \leq k_m(G_s) + 1$ となります。$k_m(G_r) = k_m(G_s)$ かつ、G_r が Complete graph であることから、Lemma 21 が成り立ちます。 □

よって、Graph G_r が Complete graph のとき、Graph G_r を最大クリークとして出力してプログラムを終了します。

Lemma 22. *Graph* G_r が *Complete graph* ではないとき、*Graph* G_r において、任意の頂点削除によって $k_m(G_r)$ が一つ減少したあと、それ以上頂点を削除すると $k_m(G_r)$ が更に減少するような、削除頂点の集合 V_b を取得します。このとき、サイズが 2 以上となる頂点集合 V_b が存在します。

Proof. Graph G_r サイズが 2 以下のとき、G_r は Complete graph となります。前提より G_r は Complete graph ではないため、G_r のサイズは 3 以上になります。

Graph $G_r = (V_r, E_r)$ の最大クリークを $G_c = (V_c, E_c)$ とします。任意の頂点 $v \in V_r, v \notin V_c$ を G_c に加えた Graph を G_d とします。G_d は Complete graph とはならないため、$k_m(G_d) = k_m(G_c)$ となります。

$k_m(G_r) = k_m(G_c)$ と仮定すると、G_r から v を削除したさいに $k_m(G_r)$ が一つ減少するという前提に反します。

a: $k_m(G_r) = k_m(G_c)+1$ のとき、頂点 v を加えた Graph G_d において、$k_m(G_d) = k_m(G_c)$ であったことから、頂点集合 V_b は頂点 v 以外の頂点を含むことになります。よって、V_b のサイズは 2 以上となります。

b: $k_m(G_r) = k_m(G_c) + i$ において、G_r の任意の 2 つの頂点 v_1, v_2 を削除したとき、$k_m(G_r)$ が 2 減少すると仮定します。G_r から頂点 v_1 を削除した Graph を G_{r1} とします。このとき、$k_m(G_{r1}) = k_m(G_c) + i - 1$ となります。Graph G_{r1} において、あるサイズ 2 以上の頂点集合 V_{b1} があるとします。頂点集合 V_{b1} に属する頂点を削除した Graph を G_{r2} として、$k_m(G_{r2}) = k_m(G_{r1}) - 1$ となります。ここで、Graph G_{r2} に頂点 v_a を追加した Graph を G_{r3} とすると、$k_m(G_{r3}) = k_m(G_r) - 1$ となります。これは、サイズ 2 以上の頂点集合 V_{b1} がある仮定と矛盾します。

a, b より、任意の i について、サイズ 2 以上の V_b が存在することになります。よって、Lemma 22 が成立します。 □

Case 2: Graph G_r が Complete graph ではなく、かつ $V_a = \emptyset$ のとき、以下を満たします。なお、$V_a = \emptyset$ のため、$G_r = G_s$ となります。

Lemma 23. *Graph* G_r からサイズが 2 以上となる頂点集合 V_b に属する頂点を削除した *Graph* を G_t とします。*Graph* G_t の最大クリークサイズは G_s と同じです。

Proof. 頂点集合 V_b に属する頂点削除の順番に依らず、$k_m(G_t) = k_m(G_r) - 1$ となります。よって、V_b に属する頂点を削除しても、最大クリークサイズに変化はありません。よって、Lemma 23 が成り立ちます。 □

Algorithm 4 において、Graph G_r が Complete graph ではなく、かつ $V_a = \emptyset$ のとき、頂点削除によって最大クリークサイズが変化しない頂点を削除する方法を示しま

Algorithm 4: Graph G_r が Complete graph ではなく、かつ $V_a = \emptyset$ のとき、最大クリークサイズを変更しない頂点集合を削除する方法。Lemma 22 より、サイズが 2 以上となる頂点集合 V_b が存在します。Lemma 23 より、G_r と G_r から頂点集合 V_b に属する頂点を削除した Graph G_t の最大クリークサイズが同じとなります。よって、G_t を返します。$2 \leq |V_b|$ となるため、このアルゴリズムによって、Graph の頂点数は必ず小さくなります。

```
Function remove_vertex_set_without_remain_vertex_set(G_r):
    V_d ← ∅
    k_{m_r} ← k_m(G_r)
    while true do
        get v_i ∈ V_r − V_d
        V_d ← V_d ∪ {v_i}
        G_t ← delete_vertex(G_r, v_i)
        V_b ← {v_i}
        foreach v_j ∈ V_r − V_d do
            G_p ← delete_vertex(G_t, v_j)
            if k_m(G_p) = k_{m_r} − 1 then
                G_t ← G_p
                V_b ← V_b ∪ {v_j}
            end
        end
        /If1 < |V_b| return G_t
    end no reach
```

す。頂点削除によって k_{m_r} が一つ減少したあと、それ以上頂点を削除すると $k_m(G_r)$ が更に減少するような、削除頂点の集合で、サイズが 2 以上となる頂点集合 V_b を取得します。Lemma 23 より、サイズが 2 以上となる任意の頂点集合 V_b において、Graph G_s と G_t の最大クリークサイズは同じになるため、G_r から頂点集合 V_b に属する頂点を削除します。Algorithm 4 の計算量は $O(n^6)$ となります。

Graph G_r から頂点集合 V_b に属する頂点を削除した Graph を G_t とします。Graph G_t のいずれの頂点を削除しても、$k_m(G_t)$ が減少します。よって、$G_t \to G_r$ としたのち、G_r が Complete graph となるまで頂点削除を続けます。

Algorithm 5 において、Graph G_r が Complete graph ではなく、かつ $V_a = \emptyset$ のとき、最大クリークを一つ抽出する方法を示します。Graph G_r, G_t の頂点集合 V_r, V_t は、$|V_r| > |V_t|$ となります。よって、ループ毎に頂点が削除され、最終的に G_t が Complete graph となります。Lemma 21 より、G_t が Complete graph のとき、G_t が最大クリークとなることから、G_t を出力してプログラムを終了します。Algorithm 5 の計算量は $O(n^7)$ となります。

Case 3: のとき、頂点削除によって最大クリークサイズを変更しない、頂点集合を求めるために必要な証明を行います。

Algorithm 2 によって、重み 1/2 の辺 e が頂点削除中の Graph G_s に挿入される場合があります。しかし、辺 e の挿入による頂点集合 V_a への影響はありません。

Algorithm 5: Graph G_r が Complete graph ではなく、かつ $V_a = \emptyset$ のとき、最大クリークを一つ抽出する方法。

```
Function ExtractMaximumClique2G_r:
    repeat
        G_r ← remove_vertex_set_without_remain_vertex_set(G_r)
    until G_r = Complete graph
    return G_r
```

Corollary 24. *Graph* G_r に重み $0 < w < 1$ の辺を一つ追加した *Graph* を G'_r とします。このとき、$k_m(G'_r) = k_m(G_r)$ となります。

Proof. Algorithm 2 による頂点集合 V_a に属する頂点を削除するときに、辺 e を挿入したさいに $k_m(G_s)$ が変化しないことを保証しているため、$k_m(G'_r) = k_m(G_r)$ となります。Corollary 24 が成立します。 □

Graph G_r に頂点 $v \in V_a$ を加えた Graph を G_d とします。このとき、$k_m(G_d) = k_m(G_r)$ となります。$k_m(G_d)$ が減少しないように頂点を順次削除していきます。G_d から削除した頂点集合を V_d とします。

Lemma 25. $V_d = \emptyset$ のとき、頂点 v の削除による最大クリークサイズの減少はありません。

Proof. Graph G_d から頂点 v を削除したグラフは G_r となります。$k_m(G_r) = k_m(G_s)$ から、Lemma 25 が成立します。 □

$V_d \neq \emptyset$ のとき、Graph G_d から頂点集合 V_d に属する頂点を削除した Graph を G'_t とします。$k_m(G'_t) = k_m(G_r)$ となります。また、Graph G_r から V_d に属する頂点を削除した Graph を G_t とします。$k_m(G_t) = k_m(G_r) - 1$ となります。Graph G_t から $k_m(G_t)$ が減少しないように順次頂点を削除します。削除した頂点集合を V_u とします。

Lemma 26. $V_u \neq \emptyset$ のとき、頂点 v の削除による最大クリークサイズの減少はありません。

Proof. Graph G'_t から頂点集合 $V_u \cup \{v\}$ に属する頂点を削除するさい、頂点削除順番によらず、$k_m(G'_t)$ が減少します。よって、Lemma 26 が成立します。 □

Lemma 27. $V_u = \emptyset$ のとき、頂点集合 $V_a \cup V_d - \{v\}$ に属する頂点の削除による最大クリークサイズの減少はありません。

Proof. Graph G_t について、$k_m(G_t) = k_m(G_r) - 1$ となります。G_t に頂点 v を加えた Graph G'_t について、$k_m(G'_t) = k_m(G_r)$ となります。よって、頂点 v は最大クリークに属します。

頂点集合 V_a 内に $V_u = \emptyset$ となるような頂点が複数存在するとします。$k_m(G_t) = k_m(G_r) - 1$ であることから、この条件を満たす任意の頂点 $v_a, v_b \in V_a$ を取ったとき、v_a, v_b

Algorithm 6: Graph G_r が Complete graph ではなく、かつ $V_a \neq \emptyset$ のとき、Graph G_r から最大クリークサイズを変更しない頂点を削除します。

```
1  Function remove_vertex_set_with_remain_vertex_set(G_r, V_a):
2      k_{m_r} ← k_m(G_r)
3      foreach v ∈ V_a do
4          G_d ← create graph G_r with v
5          V_d ← ∅
6          foreach v_r ∈ V_r do
7              G'_d ← delete_vertex(G_d, v_r)
8              if k_m(G'_d) = k_{m_r} then
9                  G_d ← G'_d
10                 V_d ← V_d ∪ {v_r}
11             end
12         end
13         if V_d ≠ ∅ then
14             G_t ← delete_vertex(G_d, v)
15             k_{m_t} ← k_{m_r} − 1
16             V_u ← ∅
17             foreach v_t ∈ V_t do
18                 G'_t ← delete_vertex(G_t, v_t)
19                 if k_m(G'_t) = k_{m_t} then
20                     G_t ← G'_t
21                     V_u ← V_u ∪ {v_t}
22                 end
23             end
24             if V_u = ∅ then
25                 return create graph G_t with v
26             end
27         end
28     end
29     return G_r
```

の両方が同時に最大クリークに属することはありません。よって、Lemma 27 が成立します。 □

Algorithm 6 において、Graph G_r が Complete graph ではなく、$V_a \neq \emptyset$ のとき、最大クリークサイズを変化させない頂点集合を削除する方法を示します。Algorithm 6 の計算量は $O(n^5)$ となります。

Theorem 28. *Input graph G から最大クリークを多項式時間で一つ抽出できます。*

Proof. Algorithm 7 において、最大クリークを一つ抽出する方法を示します。Algorithm 7 の計算量は $O(n^8)$ となります。

まずはじめに、Line 2–9 において、最大クリークを抽出する準備を行います。Graph G に頂点が無い場合、'error' を返します。Graph G に辺が無い場合、頂点集合 V 内の頂点を一つ返します。Input graph G から独立頂点を省く処理を行います。そして、Algorithm 1 を用いて k_m を削減した Graph G_s を取得します。この部分の計算量は $O(n^5)$ となります。

Line 11–14 において、Graph G_s から、頂点削除によって k_m を変化させない頂点集合 V_a を抽出します。Graph G_s から頂点集合 V_a に属する頂点を削除した Graph を G_r とします。この部分の計算量は $O(n^6)$ となります。

Graph G_r と 頂点集合 V_a において、次に示す場合が存在します。

Algorithm 7: Input graph G から最大クリークサイズを変化させない頂点を削除することで、最大クリークを一つ抽出します。

```
1  Function ExtractMaximumCliqueG:
2      if V = ∅ then
3          return error
4      end
5      if E = ∅ then
6          return any vertex ∈ V
7      end
8      G ← remove_independent_vertex(G)
9      G_s ← prepare(G)
10     while G_s ≠ Complete graph do
11         (G_r, V_a) ← remove_peripheral_vertex_set(G_s)
12         if G_r = Complete graph then
13             return G_r
14         end
15         if V_a = ∅ then
16             return extract_maximum_clique_when_va_is_empty(G_r)
17         else
18             G_s ← remove_vertex_set_with_remain_vertex_set(G_r, V_a)
19         end
20     end
21     return G_s
```

(1) Line 12–14: G_r が Complete graph のとき,

(2) Line 15–16: G_r が Complete graph ではなく、かつ $V_a = \emptyset$ のとき,

(3) Line 17–18: G_r が Complete graph ではなく、かつ $V_a \neq \emptyset$ のとき.

Line 12–14: Graph G_r が Complete graph のとき、Lemma 21 より、Graph G_r を最大クリークとして出力してプログラムを終了します。

Line 15–16: Graph G_r が Complete graph ではなく、かつ $V_a = \emptyset$ のとき、Lemma 22 より、サイズが 2 以上となる頂点集合 V_b が存在します。また、Lemma 23 より、サイズが 2 以上となる頂点集合 V_b に属する頂点を削除した Graph G_t の最大クリークサイズは G_s と同じです。Graph G_r から頂点集合 V_b に属する頂点を削除した Graph を G_t とすると、Graph G_t のいずれの頂点を削除しても、$k_m(G_t)$ が減少します。よって、G_t を G_r としたのち、G_r が Complete graph となるまで頂点削除を続けます。Graph G_r が Complete graph となった時点で、Graph G_r を最大クリークとして出力してプログラムを終了します。この部分の計算量は $O(n^7)$ となります。

Line 17–18: Graph G_r が Complete graph ではなく、かつ $V_a \neq \emptyset$ のとき、Lemma 25, Lemma 26, Lemma 27 を用いて、頂点削除によって最大クリークサイズが同じとなる頂点集合を取得します。Graph G_s からここで取得した頂点集合に属する頂点を削除します。この部分の計算量は $O(n^5)$ となります

Line 10–20 のループは、頂点が少なくとも 1 つ削除されるため、必ず終了します。 □

4. Conclusion

本論文では、最大クリークの一つを多項式時間で抽出するアルゴリズムを構成する上で必要な証明を行うとともに、そのアルゴリズムを擬似コードの形で与えました。アルゴリズムの計算量は、有理数の演算を $O(1)$ としたとき $O(n^8)$ となります。NP 完全問題の一つが多項式時間で解けることから、Complexity class NP が P と同じことを示しています。

本紙の結果は、任意の NP 問題を多項式還元によって最大クリーク問題へ還元したとしても、その計算量は $O(n^8)$ 以上となります。よって、NP 問題を現実的な時間で解けることを示してはいません。比較的小さなサイズの問題については、指数的計算量 $O(2^{f(n)})$ を持つアルゴリズムを利用するほうが現実的です。

参考文献

[1] Singh, K. K. and Pandey, A. K.: Survey of Algorithms on Maximum Clique Problem, *International Advanced Research Journal in Science, Engineering and Technology*, Vol. 2, No. 2, pp. 18–20 (2015).
[2] Zhang, X.-D.: The Laplacian eigenvalues of graphs: a survey, *arXiv preprint arXiv:1111.2897* (2011).
[3] Haemers, W. H.: Interlacing eigenvalues and graphs, *Linear Algebra and its applications*, Vol. 226, pp. 593–616 (1995).
[4] Howell, J. A.: An algorithm for the exact reduction of a matrix to Frobenius form using modular arithmetic. I, *mathematics of computation*, Vol. 27, No. 124, pp. 887–904 (1973).
[5] Howell, J. A.: An algorithm for the exact reduction of a matrix to Frobenius form using modular arithmetic. II, *Mathematics of Computation*, Vol. 27, No. 124, pp. 905–920 (1973).
[6] Storjohann, A.: An O (n 3) algorithm for the Frobenius normal form, *ISSAC*, Vol. 98, Citeseer, pp. 101–104 (1998).
[7] : STRUM'S METHOD FOR THE NUMBER OF REAL ROOTS OF A POLYNOMIAL, `https://www.imsc.res.in/~knr/past/sturm/formal_notes.pdf` (2016). [Online; accessed 06-May-2019].