Vol.2021-AL-184 No.7
2021/8/26

真区間グラフの高速な列挙アルゴリズムとその応用

武田 浩和[1]　斎藤 寿樹[1]

概要：区間グラフは区間の集合を表現に持つグラフである．真区間グラフは区間グラフの部分クラスで，どの区間も他のどの区間とも包含関係にない区間の集合を表現に持つグラフである．グラフ同型性を考慮した真区間グラフの列挙アルゴリズムとして逆探索法を用いたものが知られている．このアルゴリズムはグラフ 1 つあたりの出力に $O(1)$ 時間で，理論的に効率的であるものの，グラフの頂点数が大きくなると真区間グラフの数は膨大となり，すべてを出力することはできない．一方で，ゼロサプレス型二分決定グラフ (ZDD) は組み合わせ集合をコンパクトに表現できるデータ構造で，最適化問題や列挙問題を効率的に処理できるとして近年，注目を集めている．本研究では ZDD を用いた真区間グラフを高速に列挙するアルゴリズムを提案する．提案アルゴリズムすべての真区間グラフを表現する ZDD を頂点数 n の多項式時間・領域で構築する．これまでの逆探索法では実験的に頂点数 $n = 18$ までしか列挙できなかったのに対し，提案アルゴリズムでは $n = 700$ の真区間グラフを容易に計算可能となる．また本アルゴリズムは，頂点数だけでなく，最大クリークサイズや辺の本数などの制約を加えた真区間グラフもわずかな拡張によって列挙することができる．本稿ではこれらのアルゴリズムを解析するとともに，計算機実験によりその効率性を示す．

1. 序論

区間グラフは区間表現と呼ばれる数直線上の区間の集合を表現に持つグラフクラスである．区間グラフの頂点は区間と対応し，2 つの頂点間に辺があるとき，またそのときに限り，対応する 2 つの区間は重なりを持つ．区間グラフにはスケジューリングやバイオインフォマティクスに応用があることが知られている [1]．真区間グラフは区間グラフの部分クラスで，各区間は他のどの区間も真に含まないような区間表現を持つグラフである．真区間グラフはグラフパラメータの一つであるグラフのバンド幅と密接に関係がある．具体的には，グラフ G のバンド幅は G に辺を追加して得られる真区間グラフ G' の中で最大クリークサイズが最小のものと等しい [2]．こうしたグラフパラメータと関わりのあるグラフについて，各グラフ上における様々な構造の分布や特徴を調べるため，いくつもの列挙アルゴリズムが提案されている．

グラフの列挙とはある特徴を持つすべてのグラフを漏れなく，重複なく生成することである．特に，区間グラフのような幾何的な構造を特徴に持ついくつかのグラフクラスに対して，同型性を考慮したグラフを列挙する問題が考えられている．つまり，出力される任意の異なる 2 つのグラフは同型でないようにグラフを列挙するのである．これまでに真区間グラフ [3]，二部置換グラフ [4]，区間グラフ [5], [6]，置換グラフ [5]，トレマイックグラフ，距離遺伝的グラフ [7] に対して，1 つあたり頂点数の多項式時間でグラフを生成する列挙アルゴリズムが提案されている．これらのアルゴリズムは，Avis と Fukuda による逆探索法と呼ばれる手法を用いており，特に真区間グラフおよび二部置換グラフに対しては一つあたり $O(1)$ 時間と理論的に高速なアルゴリズムが得られている．しかし，この手法はグラフを 1 つ 1 つ出力するため，出力対象となるグラフの数が膨大になると，計算時間も多くかかってしまう．そのため，これらの列挙アルゴリズムでは，10 から 20 頂点程度のグラフしか列挙することはできない [5], [6], [7], [8]．

一方で，列挙問題や最適化問題を解くアプローチとしてゼロサプレス型二分決定グラフ (ZDD) を用いた手法が盛んに研究されている [9]．ZDD は組合せ集合をコンパクトに表現するデータ構造で，膨大な情報も高圧縮で表現できるため，様々な列挙アルゴリズムで用いられている．フロンティア法は効率的にある特徴を持つすべての部分グラフを表す ZDD を構築する手法で，この手法を用いることで配電網の構成や種々のリンクパズルへの応用など，様々な問題へと応用できる [10], [11]．近年では，ZDD およびフ

[1] 九州工業大学
Kyushu Institute of Technology

ロンティア法を用いて，幾何的な特徴を持つ弦グラフや区間グラフを列挙するアルゴリズムが提案されている [12]．しかし，これらで提案されているアルゴリズムは同型性を考慮していないラベル付きのグラフを扱っている．つまり，ZDD によって表現されているグラフの中には同型なグラフが含まれているのである．それに対し，ZDD を用いてグラフ同型に関するグラフの標準形のみを列挙するアルゴリズムが提案されているが [13]，同型性はある種の対称的な構造を保持しなければならないため，ZDD では圧縮が起きにくく，頂点数 8 までしか列挙できていない．そのため，ZDD はグラフの同型性を考慮した列挙アルゴリズムを扱うことは困難であると考えられてきた．

そこで，本研究では，真区間グラフを対象に ZDD を用いた列挙アルゴリズムを提案する．n 頂点の真区間グラフは 2 種類 $\Sigma = \{L, R\}$ の $2n$ 個の文字からなる対応の取れた文字列で表現できる [3]．対応の取れた文字列は，Dyck パスと呼ばれる構造で表現でき，文字列の左から順に ZDD の構造に自然に対応させることができる．しかし，真区間グラフは文字列を反転させた構造も同じグラフを表現してしまうため，このような単純な表現では効率的に同型性を排除することはできない．そこで，提案アルゴリズムではその自然な順序を崩し，両端の文字を交互に表現させていくことで，標準形となる文字列のみを効率的に列挙する．これにより，提案アルゴリズムが構築する ZDD のサイズは $O(n^3)$ で，最悪計算量として $O(n^3 \log n)$ 時間で列挙することができる．逆探索法によるアルゴリズムでは頂点数 $n = 20$ 程度までしか列挙できなかったのに対し，提案アルゴリズムでは $n = 800$ 程度のすべての真区間グラフを 1 分以内に計算できることを計算機実験により示す．また提案アルゴリズムは拡張性が高く，わずかな変更により，真区間グラフの最大クリークサイズや辺の本数を制限することができる．特に，真区間グラフ G が最大クリークサイズ k を持つとき，G の任意の部分グラフのバンド幅は k 以下であることが言える．さらに ZDD の応用の 1 つにランダム生成があり，それを用いて列挙された最大クリークサイズ k の真区間グラフをランダムに生成することで，バンド幅問題に対するインスタンス生成として利用することができるようになる．また，最大クリークサイズにおける真区間グラフの分布がわかることで，最大クリークサイズに依存するアルゴリズムに対する平均時間解析など，様々な応用が期待できる．本稿では，これらの拡張されたアルゴリズムを実装し，最大クリークの大きさに関するグラフの数の分布や，辺の本数を制限した場合における計算時間について議論する．

本論文では，まず 2 節で真区間グラフの定義や特徴および ZDD の定義を行う．3 節でアルゴリズムを提案し，このとき各アルゴリズムにおける ZDD の節点数を解析する．4 節では各アルゴリズムを実装し，計算機実験によりアルゴリズムの効率性や出力から得られる真区間グラフの特徴について考察する．

2. 準備

無向グラフとは，頂点集合 V と辺集合 E からなる組 (V, E) である．ここで辺は V 上の非順序対である [14]．また，$n = |V|$，$m = |E|$ とする．頂点の列 $P = (v_1, v_2, \ldots, v_k)$ に対し，任意の $i \in \{1, \ldots, k-1\}$ において $\{v_i, v_{i+1}\} \in E$ であるとき，P を**道**という [15]．頂点 v_1 と v_k は P の**端点**という．グラフ G の任意の 2 頂点 v, w に対して，v と w を端点に持つ道が存在するとき，G は**連結**であるという [15]．2 頂点 $u, v \in V$ について，$\{u, v\} \in E$ であるとき，頂点 u, v は**隣接する**という．相違なる 2 頂点がすべて隣接しているグラフを**完全グラフ**といい，n 頂点からなる完全グラフを K^n で表す [16]．また，あるグラフ G と H について，$V(G)$ から $V(H)$ への 1 対 1 対応 f で

$$\{u, v\} \in E(G) \leftrightarrow \{f(u), f(v)\} \in E(H)$$

を満たすものが存在するとき，G と H は同型であるという [16]．グラフ $G = (V, E)$ に対し，$V' \subseteq V$，$E' \subseteq E$ であるとき，$G' = (V', E')$ を G の**部分グラフ**という．部分グラフ G' が完全グラフ K^k と同型であるような辺部分集合 E' が存在するとき，V' を G の大きさ k の**クリーク**という [15]．

2.1 真区間グラフ

グラフ $G = (V, E)$ に対し，$V = \{v_1, \ldots, v_n\}$ とする．また，数直線上のある n 個の区間の集合 $I = \{I_1, I_2, \ldots, I_n\}$ とする．ここで，区間 $I_i (i \in \{1, \ldots, n\})$ は半開区間 $[l_i, r_i)$ とする．このとき，任意の辺 $\{v_i, v_j\} \in E$ であることの必要十分条件が $I_i \cap I_j \neq \emptyset$ であるとき，G を**区間グラフ**といい，I を G の**区間表現**という．真区間グラフは，区間グラフの部分クラスで，任意の区間がほかのどの区間も含むことがない区間表現をもつ．このような区間表現を**真区間表現**と呼ぶ．真区間グラフの真区間表現を左から走査することで，区間の始点を表す文字 L と終点を表す文字 R の $\Sigma = \{L, R\}$ を用いた Σ^* 上の文字列として自然に表すことができる [3]．真区間グラフから得られる文字列を，真区間グラフの**文字列表現**と呼ぶ．**図 1** に真区間グラフとそれに対応する真区間表現及び文字列表現の例を示す．真区間表現は他の区間を含むような区間がないため，文字列表現における左から i 個目の L は必ず左から i 個目の R と対応し，真区間表現から文字列表現は一意に定まる．真区間グラフの文字列表現について以下の定理が成り立つ．

定理 1 ([3])．文字列 $x \in \Sigma^*$ が頂点数が n 個である真区間グラフの文字列表現であるとき，x は $2n$ 個の文字を持ち，L と R の数はそれぞれ n 個ずつ存在する．

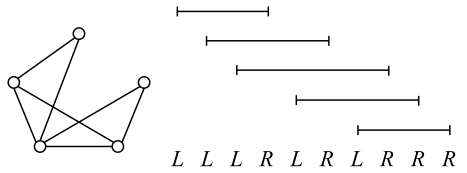

図 1 真区間グラフと対応する真区間表現及び文字列表現の例

真区間グラフを表す文字列表現に対して，**高さ** $h_x(i)$ を定義する．真区間グラフの文字列を $x = x_1x_2 \dots x_{2n}$ とし，$1 \leq i \leq 2n$ とするとき，文字列中の各文字に対して以下の漸化式で定義する．

$$h_x(i) = \begin{cases} 0 & \text{if } i = 0 \\ h_x(i-1) + 1 & \text{if } x_i = L \\ h_x(i-1) - 1 & \text{if } x_i = R \end{cases}$$

直感的に i 文字目の高さは部分文字列 $x_1 \dots x_i$ に含まれる L の数から R の数を引いたものである．つまり区間表現を左から走査したときの左端点の数から右端点の数を引いたものであるため，必ず 0 以上となる．ここで，連結な真区間グラフを表す文字列表現について，以下の定理が成り立つ [3].

定理 2. 長さ $2n$ の文字列 $x = x_1x_2 \dots x_{2n}$ が以下を満たすとき，x は連結な真区間グラフの文字列表現である．

(1) $x_1 = L$ かつ $x_{2n} = R$

(2) $h_x(0) = 0$ かつ $h_x(2n) = 0$

(3) $i = 1, 2, 3, \dots, 2n-1$ において $h_x(i) > 0$ である

文字列 x を反転させた文字列を考える．まず $\overline{L} = R$, $\overline{R} = L$ とする．文字列 x を $x = x_1x_2 \dots x_n$ とする．このとき，x の**反転** $\overline{x}$ を $\overline{x} = \overline{x_n}\ \overline{x_{n-1}} \dots \overline{x_1}$ と定義する．ここで，$x = \overline{x}$ であるとき，x を**対称文字列**，$x \neq \overline{x}$ であるとき，x を**非対称文字列**と呼ぶ．

定理 3 ([3]). x をある連結な真区間グラフ G を表す文字列表現とする．このとき，x と $\overline{x}$ が表す真区間グラフは同型である．また，G は x と $\overline{x}$ 以外の文字列表現を持たない．

ここで，$L \prec R$ と定義する．長さ n の 2 つの文字列 $x = x_1, \dots x_n$ と $y = y_1 \dots y_n$ が異なる文字列 $x \neq y$ であるとする．このとき，$i \in \{1, \dots, n\}$ について，$\forall j \in \{1, \dots, i\}, x_j = y_j$ でかつ $x_i \prec y_i$ であるとき，$x \prec y$ と書く．また，$x \prec y$ もしくは $x = y$ であるとき，$x \preceq y$ と記述する．ある真区間グラフ G の 2 つの文字列表現 x と $\overline{x}$ に対して，$x \preceq \overline{x}$ であるとき，x を G の**標準形**とする．

2.2 ゼロサプレス型 2 分決定グラフ

ZDD は非巡回有向グラフの一種であり，二分決定木を圧縮することにより得られる [17]. **図 2** に二分決定木の例

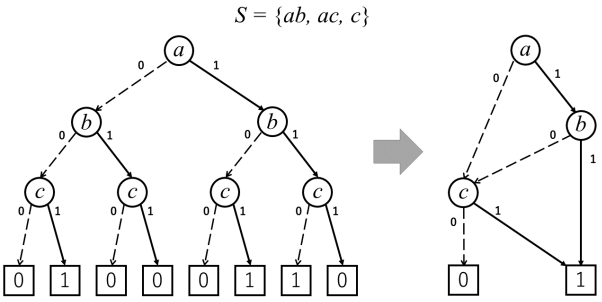

図 2 二分決定木と ZDD

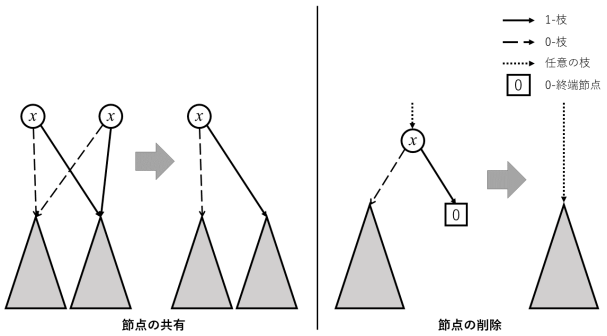

図 3 ZDD の簡約化規則

とそれを圧縮した ZDD を示す．

ZDD は入次数が 0 の根節点と，出次数が 0 の 2 つの終端節点を持つ．各節点には集合の要素ラベルを持ち，2 つの終端節点には 0 と 1 のラベルを持つ 0-終端節点と 1-終端節点がある．要素ラベルは根から順に付けられているものとする．各内部節点は 2 つの枝と接続していて，接続する 2 本の枝には 0 と 1 のラベルがそれぞれ付けられており，0-枝と 1-枝と呼ぶ．ZDD には 2 つの簡約化規則が存在し，この簡約化規則に基づき二分決定木の圧縮を行う．簡約化規則には等価節点の共有と冗長節点の削除がある．同じ変数を持つ 2 つの節点の 0-枝と 1-枝がそれぞれ同じ節点を指す場合，等価な節点としてそれを共有する．また，ある節点の 1-枝が 0-終端節点を指す場合，その節点を冗長な節点として削除する．2 つの規則を図示したものを**図 3** に示す．任意の ZDD に対して，可能な限り簡約化規則を適用すると ZDD は一意に定まる [17]. このような ZDD を**既約な** ZDD と呼ぶ．図 2 右側に示される ZDD は，図左側の二分決定木に簡約化規則を適用した既約な ZDD である．

3. アルゴリズム

本節では，真区間グラフを列挙するアルゴリズムについて示す．3.1 節では，入力を自然数 n とし，頂点数が n である非同型な連結真区間グラフ全てを表す ZDD を生成するアルゴリズムについて述べる．3.2 節では自然数 n と k を入力し，頂点数が n でありグラフの最大クリークサイズが k であるすべての真区間グラフを，3.3 節では自然数

n と m を与え，頂点数が n でありグラフの辺の数が m であるすべての真区間グラフを表す ZDD を出力する問題を扱う．

3.1 非同型な連結真区間グラフ列挙アルゴリズム

本アルゴリズムは真区間グラフを表す標準形の文字列をすべて表す ZDD を構築する．**定理 1** より，頂点が n 個である真区間グラフの文字列表現は $2n$ 個の文字を持つ．その文字列表現を $x = x_1x_2 \dots x_{2n}$ とする．構築する ZDD の各節点のラベルは文字と対応し，各文字は 1-枝で L，0-枝で R を表すとする．ただし，ZDD における文字ラベルの順序は $x_1, x_{2n}, x_2, x_{2n-1}, \dots, x_n, x_{2n-(n-1)}$ と文字列の左側と右側を中央に向けて交互に表すものとする．ZDD の根節点からラベル $j \in \{1, \dots, 2n\}$ を持つ任意の節点 v への道は部分的な文字列 $x' = x'_1 \dots x'_{\lceil j/2 \rceil} x'_{n-\lfloor j/2 \rfloor} \dots x'_{2n}$ を表す．このとき，$x'_l = x'_1 \dots x'_{\lceil j/2 \rceil}$ および $x'_r = x'_{n-\lfloor j/2 \rfloor} \dots x'_{2n}$ とする．

ZDD はトップダウン的に根節点から順に構築する．ZDD の各節点に**計算状態**と呼ばれる情報を持たせながら，**枝刈り**や**節点の共有**を行っていく．ある節点から 0-枝または 1-枝の先の節点を作成するとき，標準形の真区間グラフを表す文字列表現が作成できない場合はその枝を 0-終端節点に接続する．この処理を枝刈りと呼ぶ．また，ある 2 つの節点が同じ計算状態であるとき，その 2 つの節点以降での標準形の文字列となる残りの文字列の組合せがすべて等しくなり，それら 2 つの節点を共有する．これらの処理を効率的に行うために，提案アルゴリズムでは各節点に次の 4 つの計算状態をもたせる．

(1) 左側の部分文字列 x'_l の高さ h_l
(2) 右側の文字列の反転 $\overline{x'_r}$ の高さ h_r
(3) 1 つ前で選択した文字 c
(4) 文字列の任意の拡張が標準形かを表すフラグ f

まずは左側の文字列の高さおよび右側の文字列の高さを表す計算状態 h_l と h_r について説明するため，次の定理を示す．

定理 4. 文字列 $x = x_1 \dots x_{2n}$ を真区間グラフを表す文字列表現とし，$x_l = x_1 \dots x_n, x_r = \overline{x_{2n}x_{2n-1}} \dots \overline{x_{n+1}}$ とする．このとき，x_l および x_r は次の 3 つを満たす．

(1) $h_{x_l}(0) = 0$ かつ $h_{x_r}(0) = 0$
(2) $j = 1, 2, 3, \dots, n$ において $h_{x_l}(j) > 0$ かつ $h_{x_r}(j) > 0$
(3) $h_{x_l}(n) = h_{x_r}(n)$

証明. (1) および (2) は**定理 2** の (2) と (3) から自明に導かれる．(3) は背理法で示す．つまり，$h_{x_l}(n) \neq h_{x_r}(n)$ とする．定義から，$h_{x_l}(n)$ は n 文字目までに現れる L の数から R の数を引いたもので， $h_{x_r}(n)$ は n 文字目までに現れる R の数から L の数を引いたものである．そのため，$h_{x_l}(n) \neq h_{x_r}(n)$ であるとき，x に含まれる L の数と R の数は異なる．一方で，**定理 1** から x に含まれる L と R の数が等しいため，x が真区間グラフの文字列表現であることに矛盾する． □

この定理から左から $k \in \{1, \dots, n\}$ 番目までの文字列 $x_1 \dots x_k$ の高さは 0 よりも大きく，右から $k \in \{1, \dots, n\}$ 番目までの文字列 $\overline{x_{2n}} \dots \overline{x_{k+1}}$ の高さは 0 よりも大きくなければならない．そのため，これまでに構築してきた部分的な文字列 x' の左側 x'_l と右側 x'_r のそれぞれの高さ h_l および h_r で管理する．

枝刈り：2 つの高さ h_l と h_r および**定理 4** から次の 2 つの枝刈りが導ける．1 つ目は **定理 4** (1)，(2) より，$j = 1, 2, \dots, n$ で $h_l \leq 0, h_r \leq 0$ のいずれかを満たすときである．2 つ目は**定理 4** (3) を満たさない場合である．これは，文字列の左右それぞれにおいて構築終了時に到達できる高さの最大値と最小値を求め，左右の高さの到達可能範囲に重なりがあるかどうかを確認することで判断できる．左側の部分文字列 $x'_l = x'_1 \dots x'_j$ において，x'_l に $n-j$ 個の文字を連結して得られる文字列 y'_l を考える．$h_{y'_l}(n)$ が最大となるのは拡張部分の文字がすべて L のときであり，最小となるのは拡張部分の文字がすべて R のときである．したがって，x'_l を拡張したときに現れる n 番目の文字での高さの範囲は $(h^{\min}_{x'_l}, h^{\max}_{x'_l})$ はで表すことができる．ここで，

$$h^{\min}_{x'_l} = h_{x'_l}(j) - (n-j)$$
$$h^{\max}_{x'_l} = h_{x'_l}(j) + (n-j)$$

とする．文字列の右側に関しても同様に高さの範囲, $h^{\max}_{x'_r}$ 及び $h^{\min}_{x'_r}$ を定義することができる．ある部分文字列 $x' = x'_l x'_r$ において，$h^{\max}_{x'_l} < h^{\min}_{x'_r}$ もしくは $h^{\max}_{x'_r} < h^{\min}_{x'_l}$ であるとき，それは**定理 4** (3) を満たさない文字列である．よって，そのような文字列に対応する節点においては，左と右の 2 つの高さを用いて枝刈りを行う．

次に標準形の文字列のみを抽出するための仕組みについて説明する．ここでは，1 つ前で選択した文字 c と標準形かを表すフラグ f の 2 つの計算状態を用いて，構築中の部分文字列が標準形かどうかを判定する．真区間表現を表す文字列表現 x に対して，$x \preceq \overline{x}$ であるとき，x は標準形である．ここで，$j \in \{1, \dots n\}$ に対して，$2j$ 文字の部分文字列 $x = x_1 \dots x_j x_{2n-j+1} \dots x_{2n}$ を構築中の文字列とし，$x_l = x_1 \dots x_j, x_r = \overline{x_{2n}} \dots \overline{x_{2n-j+1}}$ とする．計算状態のフラグ f は初期状態を **False** とし，x の任意の拡張が標準形であるとき f は **True** とする．つまり，f が **True** であれば，部分文字列は標準形であることを表す．ここで 1 つの観察として，$x_l \prec x_r$ であるとき，x の任意の拡張した文字列も標準形であることが言える．逆に，$x_r \prec x_l$ であるとき，x の任意の拡張は標準形ではない．そのため部分文字列 x が $x_r \prec x_l$ となった時点で枝刈りが行う．そ

のため，文字列を拡張しながら，$x_l \prec x_r$ もしくは $x_r \prec x_l$ を判別することで，標準形のみを列挙することができる．

文字列 x に対して $x_l = x_r$ とする．つまり，この部分文字列の任意の拡張が標準形かどうかを判別することはできない．ここで，x に 2 文字を拡張して得られる部分文字列 $x' = x_1 \dots x_j x_{j+1} x_{2n-j} x_{2n-j+1} \dots x_{2n}$ とし，$x'_l = x_l x_{j+1}, x'_r = x_r \overline{x_{2n-j}}$ とする．このとき次のことが言える．

- $x_{j+1} \prec \overline{x_{2n-j}}$ であるとき，$x'_l \prec x'_r$
- $x_{j+1} \prec \overline{x_{2n-j}}$ であるとき，$x'_l = x'_r$
- $\overline{x_{2n-j}} \prec x_{j+1}$ であるとき，$x'_r \prec x'_l$

ここで ZDD における変数に対応する文字列の順序から部分文字列 x' で最後に選ばれた文字は x_{2n-j} であり，計算状態 c はその 1 つ前に選択した文字 $c = x_{j+1}$ を表す．この計算状態 c を用いることで，標準形の判別が行える．

節点の共有：ZDD の各節点にある 4 つの計算状態は節点の共有条件となる．今，ある 2 つの節点が持つ計算状態をそれぞれ (h_l, h_r, c, f) と (h'_l, h'_r, c', f') とし，$h_l = h'_l, h_r = h'_r, c = c', f = f'$ とする．また，根から 2 つの節点への道が表す部分文字列をそれぞれ $x = x_1 \dots x_j x_{2n-j+1} \dots x_{2n}$ と $x' = x'_1 \dots x'_j x'_{2n-j+1} x'_{2n}$ とする．このとき，$h_l = h'_l$ かつ $h_r = h'_r$ であることから，x を拡張して得られる任意の真区間グラフの文字列表現を $y = x_1 \dots x_j y_{j+1} \dots y_{2n-j} x_{2n-j+1} \dots x_{2n}$ とすると，x' に $y_{j+1} \dots y_{2n-j}$ を拡張して得られる文字列 $y' = x'_1 \dots x'_j y_{j+1} \dots y_{2n-j} x'_{2n-j+1} x'_{2n}$ も **定理 4** を満たすため，真区間グラフの文字列表現となる．また，$f = f'$ であることから，すでに標準形 $x_l \prec x_r$ かつ $x'_l \prec x'_r$ か，$x_l = x_r$ かつ $x'_l = x'_r$ かがわかる．そのため，x および x' に対して拡張可能である部分文字列 $y_{j+1} \dots y_{2n-j}$ は一致するため，この 2 つの節点は共有できる．

ZDD の節点数の解析：ZDD の各節点には，4 つの状態が保存されている．h_r と h_l はそれぞれ 0 から n の高々 n 通りの値をとる．最後に構築した左側の文字 c には文字の種類の数，すなわち L と R の 2 種類が存在する．すでに部分文字列が標準形であるかどうかを判別するフラグ f は **True** か **False** の二値である．レベルが同じ節点において，この 4 つの状態が等しいときかつそのときに限り節点の共有が行われる．したがって同じレベルに存在する節点の数は高々節点の状態の数，つまり $n \cdot n \cdot 2 \cdot 2 = O(n^2)$ となる．ZDD のレベルは文字列の文字の数，つまり $2n$ である．したがって ZDD の節点数は $4n^2 \cdot 2n = O(n^3)$ となる．

3.2 最大クリークサイズを指定した列挙アルゴリズム

本節では，最大クリークサイズに制限を持つ真区間グラフを列挙するアルゴリズムを述べる．真区間グラフにおける最大クリークサイズを考える上で，重要な性質について述べる．区間表現において重なりを持つ 2 つの区間に対応する頂点間には辺が存在する．そのため，数直線上のある点を l 個の区間が含んでいる場合，区間グラフにおいて，この l 個の区間に対応する頂点はクリークを形成する．また，文字列表現の各文字における高さは，その文字に対応した端点の座標を含む区間の数を表す．ここで，以下の定理が成り立つ．

定理 5. 真区間グラフ G の最大クリークサイズが k 以下である必要十分条件は，G の文字列表現での各文字における高さが k 以下である．

入力する値にクリークサイズ k を追加し，頂点数が n，グラフの最大クリークサイズが k である非同型な連結真区間グラフの列挙を行う．これは，3.1 節の手法を用いて連結真区間グラフを表す ZDD を構築する中で，各節点において最大クリークサイズが k でないものを枝刈りすることで実現する．**定理 5** より，$1 \leq j \leq n$ において $k < h_{x_l}(j)$ あるいは $k < h_{x_r}(j)$ となる場合と $h_l(j)$ と $h_r(j)$ の最大値がいずれも k 未満の場合を 0-終端節点とすることで求めるグラフを表す ZDD を構築できる．

また，クリークサイズに関しても**定理 4**（3）から行える枝刈りと同様の枝刈りを考える．文字列構築の過程において，その時点までの高さの最大値が k 未満であり文字列の中央で到達できる高さの最大値が k 未満であるとき，その文字列は明らかに最大クリークサイズが k 未満である真区間グラフを表す．したがって ZDD の各節点において以下の条件が成り立つときも枝刈りが可能である．

- $h^{\max}_{x'_r} < k$ または $h^{\max}_{x'_l} < k$

最大クリークサイズを指定したアルゴリズムにおいては，ZDD の節点は 3.1 節で述べた 4 つの計算状態に加え新たな計算状態を 1 つ持っている．この計算状態は最大クリークサイズが k に到達しているか否か，という情報を保存する．また，最大クリークサイズが k より大きくなる場合は枝刈り処理が行われるため，h_r と h_l の最大値は k となる．したがって ZDD の節点数は $4k^2 \cdot 2 \cdot 2n = O(k^2 \cdot n)$ である．

3.3 辺の数を指定した列挙アルゴリズム

本節では，入力の自然数 m に対して，m 本の辺を持つ真区間グラフを列挙するアルゴリズムを提案する．このアルゴリズムは 3.1 節のアルゴリズムに辺の本数を表す計算状態を追加することで実現する．辺の数の計算では，区間表現における始点に注目する．文字列の構築において区間の始点が現れたとき，その区間と重複する区間の数だけグラフにおいて辺が追加される．文字列の高さは重複する区間の数と対応している．したがって文字列 x の辺の数 e は x_i が L のとき $h_x(i) - 1$ だけ e を増加させる操作を $i = 1, 2, \dots 2n$ で行うことで得られる．ただし，ZDD の構築においては文字列を左右から構築している．したがって今回は文字列の左側と右側それぞれで辺の数を数える．こ

Algorithm 1 Calculate number of edges e

```
1: e = 0
2: for i = 1...2n do
3:     if i is left side of the string then
4:         if x_i =' L' then
5:             e = e + h_l(i) − 1
6:         end if
7:     else
8:         if x_i =' R' then
9:             e = e + h_r(i) − 1
10:        end if
11:    end if
12: end for
```

のとき，文字列の右側からの走査では R が現れたとき新しい区間が追加されたとみなせる．以上より全体の辺の数 e を計算する **Algorithm 1** を示す．

このアルゴリズムの実行終了時，計算状態 e はグラフ全体の辺の数を正しく取得できていない．文字列構築の順序に基づき辺の数も文字列の両側から計算しているため，文字列の中央付近では重複して数えている辺が存在するためである．したがって，上記アルゴリズムの実行後重複して数えている辺の数を減算することで正しい辺の数を得られる．

減算する値は重複して数えている部分だが，$h_x(n) = 1$ のとき，区間表現で中央に存在する区間が 1 つのみであるため，重複して数えている辺はない．$h_x(n) \geq 2$ のとき，重複して数えている辺の数は $h_x(n) - 1$ 頂点の完全グラフが持つ辺の数に相当する．したがって文字列の構築終了ときに e から減算する値 p は以下のように定義できる．

$$p = \sum_{k=1}^{h_r(n)-1} k = \frac{h_r(n)\{h_r(n)-1\}}{2} \qquad (1)$$

これを **Algorithm 1** の実行後の e から減算することで，その真区間グラフの辺の数を求めることができる．

文字列構築の過程において，p の最大値は $\min(h^{\max}_{x'_l}, h^{\max}_{x'_r})$ に依存し，その時点での全体の辺の数から p の最大値を減算しても m を超える場合，その文字列は以降どのように文字列を構築しても必ず辺の本数が m を超えると判断できる．したがって，構築中の文字列 x' に対して，計算状態を $e_{x'}$，$\min(h^{\max}_{x'_l}, h^{\max}_{x'_r}) = a$ とすると ZDD の各節点において以下が成り立つとき枝刈りを行うことができる．

$$e_{x'} - \frac{a(a-1)}{2} > m \qquad (2)$$

以上のことから，アルゴリズムでは，真区間グラフ全てを列挙する ZDD と比べ辺の数を保存する計算状態 e を 1 つ追加する．辺の数が最大となるのは全ての区間が重複するとき，つまり区間グラフが n 頂点の完全グラフとなるときである．n 頂点の完全グラフの辺の本数は $\frac{n(n-1)}{2}$ である．ある．このとき，減算する p の値は $n-1$ 頂点の完全グラフが持つ辺の数に相当する．したがって辺の数の状態がとりうる値の数は高々 $m + \frac{(n-1)(n-2)}{2}$ である．したがって ZDD の節点数は $4n^2 \cdot 2n \cdot \left(m + \frac{(n-1)(n-2)}{2}\right) = O(n^5)$ となる．

表 1　実行環境 1

OS	macOS Catalina バージョン 10.15.7
プロセッサ	3.3 GHz デュアルコア Intel Core i7
メモリ	16 GB 2133 MHz LPDDR3
コンパイラ	g++ 11.0.3

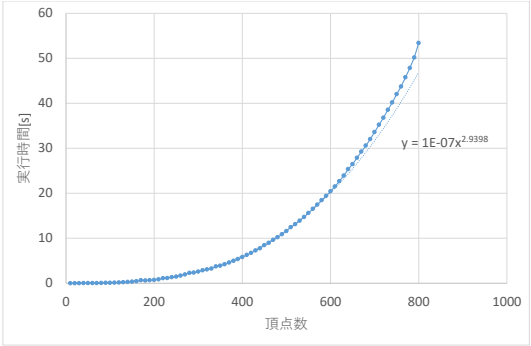

図 4　**Alg** の実行結果

4. 計算機実験

本節では 3 節にて提案した 3 つのアルゴリズムを実装し，その計算結果を示す．非同型な連結真区間グラフを列挙するプログラムを **Alg**，クリークサイズを指定したグラフを列挙するプログラムを **Clique**，辺の数を指定したグラフを列挙するプログラムを **Edge** とする．実験に用いた 3 つのプログラムでは，ZDD を効率的に構築する岩下洋哲氏開発の TdZdd ライブラリ [18] を用いるため，C++言語にて記述している．各プログラムにおいて目標となる ZDD を構築し TdZdd ライブラリの関数を用いて解の数を出力した．**Alg**，**Clique**，**Edge** を実行した環境を**表 1** に示す．

4.1 計算結果

各プログラムの実行時間について示す．自然数 n について，$n+1$ 頂点の連結真区間グラフの数は $\frac{1}{2}\left(C(n) + \binom{n}{n/2}\right)$ である [3]．**Alg** の実行時は実行時に入力した全ての n において，出力される解の数が連結真区間グラフの数と一致することを確認した．なお，全ての実験について実行時間は浮動小数点で有効数字を 6 桁に丸め，近似曲線の導出には Microsoft Office Excel 2016 の近似曲線ツールを用いた．**図 4**，**図 5**，**図 6** にそれぞれ **Alg**，**Clique**，**Edge** を実行した際の実行時間とその近似

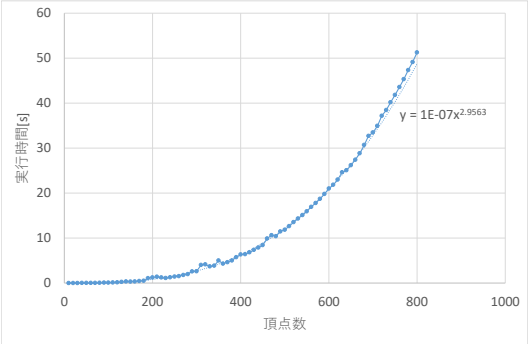

図 5　Clique の実行結果 1

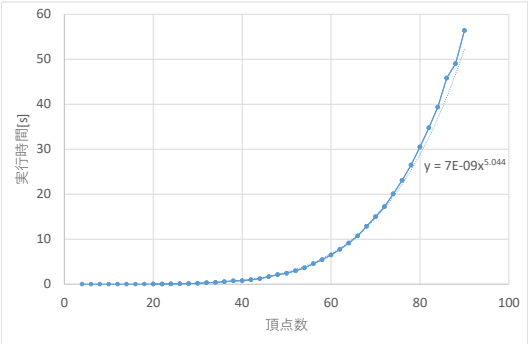

図 6　Edge の実行結果 1

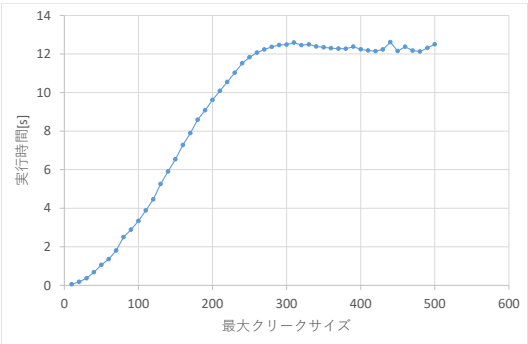

図 7　Clique の実行結果 2

曲線を示す．各図において，x 軸を入力した頂点数 n，y 軸を実行時間としている．波線が近似曲線を表し，図中の数式は Excel で得られた近似曲線の式である．また，図 5 では各頂点数において入力する k の値は $\frac{n}{2}$ とし，図 6 では各頂点数において入力する m の値は $\lfloor \frac{n(n-1)}{4} \rfloor$ とした．

次に，頂点数を固定し最大クリークサイズを変化させた際の実行時間の推移を調査する．**図 7** に入力する頂点数 n を $n = 500$ とし，**Clique** を実行した際の実行時間を示す．

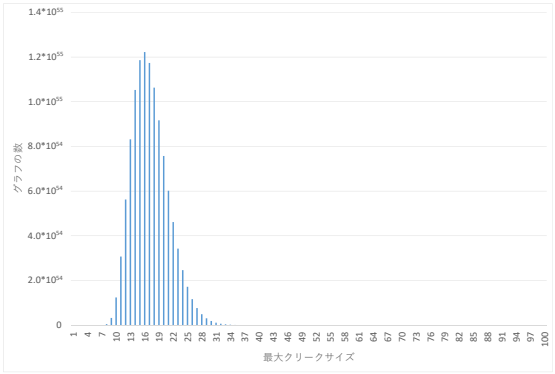

図 8　Clique の実行結果 3

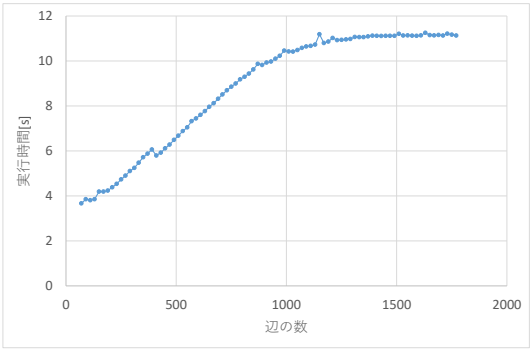

図 9　Edge の実行結果 2

x 軸を入力した最大クリークサイズ k，y 軸を実行時間としている．

次に，ある頂点数において最大クリークサイズを指定したグラフの数はどのような分布となるのかを調査する．**図 8** では，**Clique** 実行時に得られた真区間グラフの数を示す．入力する頂点数は $n = 100$ とする．x 軸を入力した最大クリークサイズ k，y 軸を真区間グラフの数としている．グラフの数は $n = 100$，$k = 10$ の時点で 55 桁に上る膨大な数となる．図 8 ではグラフの数を浮動小数点で有効数字を 15 桁に丸め，描画している．

図 9 に入力する頂点数を $n = 60$ とし，**Edge** を実行した際の実行時間を示す．x 軸を入力した辺の数 m，y 軸を実行時間としている．

4.2　考察

連結真区間グラフの列挙は，先行研究によるアルゴリズムでは $n = 18$ までの列挙しか実行できていなかった．しかし，今回作成した **Alg** では $n = 800$ の連結真区間グラフを列挙する ZDD を約 53 秒で構築することができた．

クリークサイズを指定した **Clique** では，同じ頂点数において指定するクリークサイズが小さいほど実行時間が小さく，クリークサイズが大きいほどその実行時間は **Alg**

における同じ頂点数の実行時間に近いものとなった．これは，クリークサイズが小さいほど枝刈りが起こる回数が多いため，概ね予測通りの結果であると言える．

辺の数を指定した **Edge** では，$n = 90$，$m = 2002$ としたときに実行に約 56 秒かかり，**Alg** に比べ非常に小さな頂点数でしか列挙できなかった．この実行時間の大きさは，文字列の構築順序に起因する．今回のアルゴリズムはグラフ同型の判定のため文字列をその両側から構築しているが，そのために文字列の構築途中において辺の数が指定された値を正確に超えているかどうかを判断することが難しくなっているため，枝刈りが起こりづらい．また，同じ頂点数において指定する辺の数が小さいほど実行時間が小さくなっているのは **Clique** と同じく枝刈りの頻度が高いためであると言える．

図 4，図 5 より **Alg** 及び **Clique** の計算時間はおよそ $O(n^3)$ となっている．また図 6 より，**Edge** の計算時間はおよそ $O(n^5)$ となっている．これらのプログラムの計算時間が 3 章にて解析した各 ZDD の節点数と概ね一致していることが確認できた．

5. 結論

本研究では ZDD を用い非同型な連結真区間グラフを高速に列挙するアルゴリズムを提案した．このアルゴリズムは頂点数 n に対して大きさが $O(n^3)$ である ZDD を構築する．また，計算機実験においてアルゴリズムの実行時間が $O(n^3)$ に近いことを確認した．加えて，応用として入力に頂点数だけでなく最大クリークサイズや辺数を与えて真区間グラフを列挙するアルゴリズムも提案した．

今後の課題として，本研究のアルゴリズムのアイデアを他のグラフクラスに応用することが考えられる．例えば，真区間グラフと同様に文字列で表現することのできる 2 部置換グラフ等が挙げられる．こうしたグラフクラスに対して，ZDD を用いたアルゴリズムが設計できないかを検討する．

参考文献

[1] Golumbic, M. C.: *Algorithmic graph theory and perfect graphs*, Elsevier (2004).
[2] Kaplan, H. and Shamir, R.: Pathwidth, bandwidth, and completion problems to proper interval graphs with small cliques, *SIAM Journal on Computing*, Vol. 25, No. 3, pp. 540–561 (1996).
[3] Saitoh, T., Yamanaka, K., Kiyomi, M. and Uehara, R.: Random Generation and Enumeration of Proper Interval Graphs, *IEICE Transactions on Information and Systems*, Vol. E93.D, No. 7, pp. 1816–1823 (2010).
[4] Saitoh, T., Otachi, Y., Yamanaka, K. and Uehara, R.: Random generation and enumeration of bipartite permutation graphs, *Journal of Discrete Algorithms*, Vol. 10, pp. 84–97 (2012).
[5] Yamazaki, K., Saitoh, T., Kiyomi, M. and Uehara, R.: Enumeration of nonisomorphic interval graphs and non-isomorphic permutation graphs, *Theoretical Computer Science*, Vol. 806, pp. 310–322 (2020).
[6] Mikos, P.: Efficient enumeration of non-isomorphic interval graphs, *Discrete Mathematics & Theoretical Computer Science*, Vol. vol. 23 no. 1 (online), available from ⟨https://dmtcs.episciences.org/7246⟩ (2021).
[7] Yamazaki, K., Qian, M. and Uehara, R.: Efficient Enumeration of Non-isomorphic Distance-Hereditary Graphs and Ptolemaic Graphs, *WALCOM: Algorithms and Computation: 15th International Conference and Workshops, WALCOM 2021, Yangon, Myanmar, February 28–March 2, 2021, Proceedings 15*, Springer International Publishing, pp. 284–295 (2021).
[8] 原沢寿美子，上原隆平：CONNECTED PROPER INTERVAL GRAPH の効率の良い列挙アルゴリズムに関する研究，信学技報，Vol. 118, No. 517, pp. pp.9–16 (2019).
[9] ERATO 湊離散構造処理系プロジェクト：超高速グラフ列挙アルゴリズム：“フカシギの数え方”が拓く、組合せ問題への新アプローチ，森北出版 (2015).
[10] 井上　武，高野圭司，渡辺喬之：フロンティア法による電力網構成制御，オペレーションズ・リサーチ/ 日本オペレーションズ・リサーチ学会，Vol. 57, No. 11, pp. pp.610–615 (2012).
[11] 吉仲　亮，岩下洋哲，川原　純，斎藤寿樹，鶴間浩二，湊　真一：種々のリンクパズルへの応用，オペレーションズ・リサーチ/ 日本オペレーションズ・リサーチ学会，Vol. 57, No. 11, pp. pp.616–622 (2012).
[12] Kawahara, J., Saitoh, T., Suzuki, H. and Yoshinaka, R.: Colorful frontier-based search: implicit enumeration of chordal and interval subgraphs, *International Symposium on Experimental Algorithms*, Springer, pp. 125–141 (2019).
[13] 大澤賢悟，中畑　裕，湊　真一：グラフ同型に関する代表元のグラフを列挙する ZDD の構築について (特集「知識に関する処理の最新動向」および一般)，人工知能基本問題研究会，Vol. 110, pp. 25–30（オンライン），入手先 ⟨https://ci.nii.ac.jp/naid/40022030225/⟩ (2019).
[14] Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C.: *Introduction to Algorithms, third edition*, The MIT Press (2009).
[15] Diestel, R.: *Graph Theory, Second Edition*, Springer-Verlag New York, Inc. (2000).
[16] 恵羅　博，土屋守正：シリーズ/情報科学の数学 グラフ理論，産業図書 (1996).
[17] ERATO 湊離散構造処理系プロジェクト：超高速グラフ列挙アルゴリズム，森北出版株式会社 (2015).
[18] Hiroaki, I., Jun, K. and Kohei, S.: kunisura/TdZdd, Kyoto University (online), available from ⟨https://github.com/kunisura/TdZdd⟩ (accessed 2021-07-06).