ZDDを用いた三角形分割パターンの列挙とその応用に向けて

Enumeration of triangulations by ZDD

熊澤輝顕 *　鈴木浩史　石畠正和
浅井哲也　池辺将之　本村真人　高前田伸也
Teruaki Kumazawa　Hiroshi Suzuki　Masakazu Ishihata
Tetsuya Asai　Masayuki Ikebe　Masato Motomura　Shinya Takamaeda

北海道大学　大学院情報科学研究科
Graduate School of Information Science and Technology, Hokkaido University

Abstract: In this paper, we propose a new ZDD-based algorithm for enumerating all triangulations. A triangulation of a given vertex set V is an edge set $T \subseteq E = V \times V$ such that a graph (V, T) is a maximal planer graph. Generally, the number of all possible triangulations is exponential in $|V|$. To cope with such large triangulations, we use ZDDs that is a compact DAG representation of a set family. We propose a top-down algorithm for constructing a ZDD representing all triangulation. We also empirically show that our algorithm is drastically faster than an existing enumeration algorithm for triangulations.

1　はじめに

1.1　背景

三角形分割とは頂点が入力された時に、連結である平面グラフで表現するものである。頂点集合に対して三角形分割の方法は複数存在する。三角形分割問題の応用先としては、連続体をより単純なモデルで表現する有限要素法や、CG などに応用されている。ある評価基準を満たす三角形分割を得る方法は様々なものが知られている。例えば最小角最大の三角形分割を得るアルゴリズムとしてはドロネー三角形分割法が知られている。ドロネー三角形分割の計算量は入力頂点が n である時、$O(n \log n)$ である。最適解を一つ求めれば良い問題や二次元では絶大な効力を発揮する。ドロネー三角形分割を三次元で用いる場合、局所解に陥ることがある。また、何らかの評価値が一定の閾値を超える複数の解が必要な場合、解を一つ求める手法では対応できない。これらの問題は、複数存在する三角形分割（三角形分割パターン）を列挙することで解決できる。しかし、三角形分割パターンを全て列挙する場合、その解の数は入力頂点数の増加によって指数的に増える。そのため効率良く三角形分割パターンの列挙を行うアルゴリズムが求められている。

*連絡先：北海道大学　大学院情報科学研究科
〒 060-0814 札幌市北区北 14 条西 9 丁目 北海道大学 大学院情報科学研究科 情報エレクトロニクス専攻 集積システム講座情報棟（M 棟） 2F M252 室
E-mail:kumazawa.teruaki.b5 at ist.hokudai.ac.jp

1.2　関連研究

三角形分割パターンの列挙アルゴリズムは Avis と Fukuda によって提案された [5]。さらに Katoh と Tanigawa によって、より効率的な手法が提案された [6]。これらの手法は与えられた頂点が同一直線上にある場合を考慮していない。一方で、同一直線上にある点を考慮した上で同程度の計算量で列挙するアルゴリズムも提案されている [7]。一般的に三角形分割パターンの列挙は入力頂点数が増えるにつれ計算量は指数的に多くなる。[7] では三角形分割パターンの列挙を行なった際、入力頂点数 16 に対して計算時間が約 100 秒程度である。

1.3　本研究の貢献

本論文では三角形分割パターンの列挙をより高速に行うアルゴリズムを提案する。三角形分割とは本質的には頂点が入力された時にどの頂点同士を繋ぐか (どの線分を利用するか) を決める問題である。すなわち、三角形分割パターン列挙は利用する線分の組み合わせ問題として解くことができる。

組み合わせ問題の解の列挙に対して、ZDD というデータ構造を利用した研究が盛んに行われている [1, 8, 4]。ZDD (Zero-suppressed binary Decision Diagram) とは場合分け二分決定木をある規則に従って圧縮したデータ構造である [2]。本論文では三角形分割パターン列挙問題を、ZDD を用いることで効率的に解く手法を提案する。

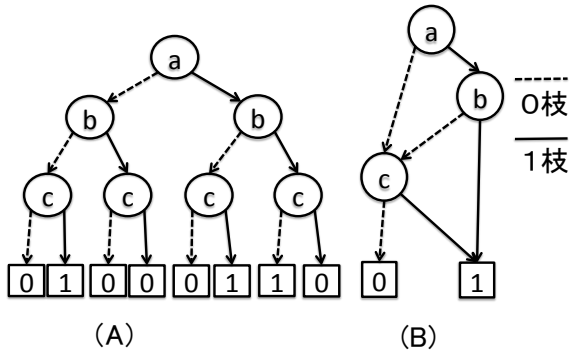

Figure 1: 場合分け二分木 (A) と ZDD(B)

2 問題定義

この章では、三角形分割の定義、並びに本論文で扱う三角形分割パターン列挙問題の定義を述べる。

2.1 三角形分割問題

$V := \{v_i \in \mathrm{R} \times \mathrm{R} \mid i = 1, \ldots, n\}$ を 2 次元空間の頂点集合とし、$E := \{[v, v'] \mid v, v' \in V\}$ を V の要素を端点とする線分（区間）の集合とする。E の要素を $e_i \in E$ $(i = 1, \ldots, m)$ と表す。ここで、m は全ての頂点の組み合わせの総数に一致するため $m = n(n+1)/2$ である。$T \subseteq E$ に対して、グラフ (V, T) が極大平面グラフであるとき、T を V の **三角形分割** という。T が V の三角形分割であることを $Tri(V, T)$ と表す。三角形分割 T は以下を満たす。

$$\forall e, e' \in T, e \sqcap e' = \emptyset \tag{1}$$

$$|T| = 3n - 3 - k \tag{2}$$

ここで $e \sqcap e'$ は線分 e と e' の交差区間を意味し、k は V の凸包の上の頂点数である。

2.2 三角形分割パターン列挙問題

三角形分割パターン列挙問題とは、頂点集合 $V = \{v_1, ..., v_n\}$ が与えられたとき、全三角形分割パターンの集合 $\Delta(V) = \{T \subseteq 2^E \mid Tri(V, T)\}$ を返す問題である。一般的に三角形分割パターンの総数 $|\Delta(V)|$ は n に対して指数的なサイズとなるが、その具体的な数は入力頂点 V のサイズとその座標に依存する。

3 提案手法

三角形分割パターンの総数 $|\Delta(V)|$ は $|V|$ に対して指数的に増えるため、愚直に解を出力すると指数的な時間を要する。本論文では $\Delta(V)$ を ZDD と呼ばれるデータ構造によって圧縮することによりこの問題を回避する。3.1 節では ZDD とその性質について述べ、3.2 節では $\Delta(V)$ を表現する ZDD をトップダウンに構築する方法を提案する。

3.1 ZDD

ZDD は、集合族の有向非巡回グラフによる表現方法である [2]。集合族 $S = \{ab, c\}$ を表す場合分け二分木を Figure 1(A) に示す。この場合分け二分木を圧縮することにより得られる ZDD のグラフが Figure 1(B) である。ZDD の内部ノードは、集合の要素をラベルとして持ち、2 種類の出力エッジ (1 枝と 0 枝) を持つ。ZDD では 1 つの集合を根から端点へのパスで表現する。あるパスがラベル a を持つノードの 1 枝 (0 枝) を含むとき、そのパスに対応する集合は a を要素として持つ (持たない) ことを意味する。あるノード α の 1 枝側の子を 1 子、0 枝側の子を 0 子と呼び、それぞれ α_1, α_0 と表す。その集合が集合族に含まれる場合、その端点は 1 終端に、含まれない場合には 0 終端にそれぞれ達する。つまり ZDD の根から 1 終端に至るパスの数は、集合族のサイズに一致する。ZDD は Figure 2 に示すように (A) 冗長点の削除と (B) 等価なノードの共有の二つの圧縮規則により集合族をコンパクトに表現する。ZDD に対してこれ以上圧縮規則を適用できないとき、その ZDD を既約であるという。

ZDD の構築方法は apply 演算によるボトムアップ方式 [2] と、フロンティア法によるトップダウン方式 [1] の二種類がある。同じ ZDD をボトムアップ方式とトップダウン方式で構築した場合、トップダウン方式の方が経験的に高速であることが知られている [1]。

トップダウン方式は、ZDD をレイヤー毎に上から構成する。具体的には、深さ i までのノード集合 N_i が構成されているとき、その情報を利用して深さ $i+1$ のノード集合 N_{i+1} を構成する。ノード $\alpha \in N_i$ が与えられたとき、その子 α_1 と α_0 を構成し、N_{i+1} に追加する。しかし場合によっては、α_x $(x \in N_{i+1})$ を構成したとき、既に等価なノードが N_{i+1} に存在する場合がある。また、α_x が端点になる場合もある。ZDD のトップダウン構築では、この**等価なノードの検出と端点の検出**をどのように行うかが問題となる。この検出を効率的に行うため、各ノードは *configure* と呼ばれる情報を保持する。つまり、ZDD のトップダウン構築とは、configure の定義とそれを用いた等価なノードの検出および端点の検出をどのように行うかを定義したものである。トップダウン構築では、構築された ZDD が必ずしも既約とは限らない。そのため、必要に応じて Figure 2 に示す圧縮規則を適用して ZDD を既約にする。

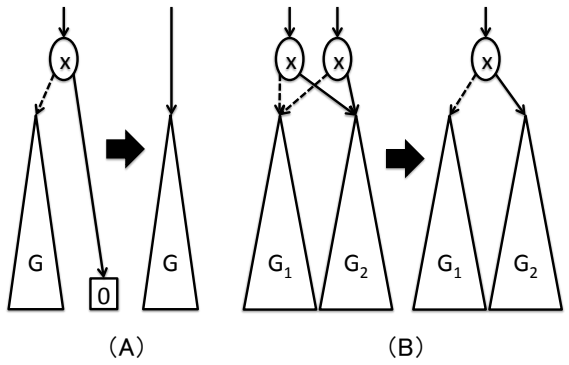

Figure 2: 不要ノードの削除 (A) とノードの共有 (B)

3.2 三角形分割パターン列挙における ZDD のトップダウン構築法

ここでは三角形分割パターン $\Delta(V)$ を表現する ZDD をトップダウン構築する手法を提案する。まず各ノードが持つ情報 configure を定義し、configure を利用した等価なノードの検出と端点の検出を定義する。

3.2.1 configure

ZDD のトップダウン構築では、等価なノードの検出と端点の検出を効率的に行うために、ZDD の各ノードに configure と呼ばれる情報を保持する。提案法では、ノード α の configure として、は線分のタブーリスト $L_\alpha \subseteq E$ とこれまでに利用した線分の数 s_α を利用する。$\mathcal{T}_\alpha \subseteq 2^E$ を根から α に至る全てのパスに対応する線分集合の族とする。このとき L_α と s_α は以下を満たす。

$$e \in L_\alpha \Longleftrightarrow \forall T \in \mathcal{T}_\alpha, \exists e' \in T, e \sqcap e' \neq \emptyset \quad (3)$$

$$\forall T \in \mathcal{T}_\alpha, |T| = s_\alpha \quad (4)$$

三角形分割の定義によりすでに使用した線分と交差している線分は利用できない。タブーリスト L_α はこの利用できない線分を保持していることに対応する。

ラベル $e \in E$ を持つノード α の子 α_1, α_0 を生成したとき、それらの configure はそれぞれ以下を満たす。

$$L_{\alpha_0} = L_\alpha, \qquad s_{\alpha_0} = s_\alpha \quad (5)$$

$$L_{\alpha_1} = L_\alpha \cup C_e, \qquad s_{\alpha_0} = s_\alpha + 1 \quad (6)$$

ここで $C_e := \{e' \in V \times V \mid e \sqcap e'\}$ は e と交差している線分集合と定義する。

3.2.2 端点の検出

端点の検出とは、ラベル e を持つノード α の子ノード α_1, α_0 を生成するとき、それらが端点となるかを判定することである。α が以下を満たすとき、α_1 は 0 終端となる。

$$e \in L_\alpha \quad (7)$$

また生成された α が以下を満たすとき、α_0 は 0 終端となる。

$$\forall e' \in C_e, e' \in L_\alpha \quad (8)$$

上式の枝刈り（終了判定）は以下の命題より導かれる。

$$Tri(V,T) \Rightarrow \forall e \in E, \exists e' \in C_e, e' \in T \quad (9)$$

また、最終レイヤーにおいては、式 (2) により、$|T| \neq s_{\alpha_x}$ $(x = 0, 1)$ の時、α_x は 0 終端となる。更に $|T| = s_{\alpha_x}$ のときは 1 終端となる。端点の検出の疑似コードを Algorithm 1 に示す。入力は configure、出力は端点検出の結果となっている。

Algorithm 1 End point detection

```
getRoot() //Create root node. m = 1
for i = 1, ..., m do
  while 0-edge and 1-edge do
    if 0-edge then
      if C_ei ⊂ taboolist then
        return 0 //not Triangulation
      else
        if e_i ∈ taboolist then
          return 0 //not Triangulation
        else
          taboolist ← C_ei
        end if
      end if
    end if
  end while
  if The last layer (i = m) then
    if s = |T| then
      return 1 // Triangulation
    else
      return 0 //not Triangulation
    end if
  end if
end for
```

3.2.3 等価なノードの検出

ZDD を圧縮する際に重要になるのが等価なノードの共有である。ノードの共有も端点の検出と同様に configure を用いて行う。$\alpha, \alpha' \in N_i$ を満たす二つのノードについて、L_α と $L_{\alpha'}$ が一致する場合 (以降 $L_\alpha = L_{\alpha'}$

と表現する)、α と α' は等価であるとみなされる。以下にその正当性を示す。

$E_{1:i} := \{e_1, e_2, ..., e_i\}$、$E_{i+1:m} := \{e_{i+1}, e_{i+2}, ..., e_m\}$ とし、深さ (線分番号)i までの処理は終わっているものとする。その状態を $S, S' \in E_{1:i}$ と表す。ここで $S \equiv S'$ とは、$S \cup X \in \Delta(V)$ かつ $S' \cup X \in \Delta(V)$ $(X \in E_{i+1:m})$ を意味する。二つの状態においてその後のノードの作られ方が同じという意味で、等価であることを $E(S) = \{e \in T_S \mid e \in E_{i+1:m}\}$,$E(S') = \{e \in T_{S'} \mid e \in E_{i+1:m}\}$ と、ある関数を用いて $P(E(S)) = P(E(S'))$ と表現する。$E(S), E(S')$ は三角形分割する上で必要な辺であるから、使用すべき辺であると言える。$P(E(S)) = P(E(S'))$ であるならば、$E(S) = E(S')$ であり、これが一致する場合 $\overline{E}(S) = \overline{E}(S')$ が成立する。$\overline{E}(S)$ はここでは L と同義なので、L の情報が一致する二つのノードは共有可能である。

以上により、$L_\alpha = L_{\alpha'}$ の時、α と α' は等価である。$L_\alpha = L_{\alpha'}$ となる α, α' が存在している場合、後で作成されたノード α' は削除され、先に作成されたノード α に枝をつなぎ直す。

4 実験と考察

この章では、提案手法をランダムに生成された頂点集合に適用し、その実行時間を測定する。

4.1 実験設定

本実験では、入力頂点集合 V を 100×100 の格子グラフから頂点を n 個ランダムに選ぶことで生成した。これは従来手法 [7] と同じ実験設定である。

実験環境は以下の通りである。使用 OS は macOS(Intel Core i7 3.3 GHz),LPDDR3 2133 MHz 16 GB, コンパイラは Apple LLVM version 8.1.0 (clang-802.0.42) の環境のもと行い、オプションとして-O2 を使用した。C++で提供されているライブラリ「TdZdd」を用いて実装した [3]。

実験は各頂点数 n ごとに 10 回ずつ行い、実行時間と解の総数はその平均を示す。実行時間と解の総数を Table 1 に示す。また構築された ZDD のノード数を Table 2 に示す。ここで、提案手法は規約な ZDD を出力するとは限らないが、実験では既約化は行っていない。

4.2 実験結果

従来手法 [7] と比較して、提案手法は約 1/1000 の時間で ZDD を構築できた。ZDD のノード数が増えるにつれ実行時間も増えていることが Table 1 と Table 2 を見るとわかる。今回の計測では ZDD ノード数の変化がノード数が 1 増えると約 2 倍になっているため、頂点数が多くなればなるほど実行時間が指数的に増えていくことが予測される。

n	time(sec)	solutions
16	0.102	8,717,336.3
17	0.219	28,529,622
18	0.457	138,780,797.8
19	0.975	725,263,808.8
20	2.510	6,606,992,577

Table 1: ZDD による三角形分割計測時間

n	nodes
16	252,259.3
17	568,930.4
18	1,193,304
19	2,598,218
20	5,288,618.1

Table 2: ZDD ノード数の増え方

5 おわりに

本論文では三角形分割パターン列挙問題を ZDD を用いて高速に解く手法を提案した。また、ランダムグラフにおいて提案手法が高速に実行できることを実証した。

ZDD は解集合をコンパクトに表現可能であるだけでなく、何らかの条件を指定し、その条件満たす解を集合演算により効率的に取り出すことが可能である。CG などに代表されるように「見て美しいかどうか」等を厳密な関数により唯一解を定めることは難しい。提案手法ではあらかじめ解を全列挙・保存しておき、利用者の要請により、特定の条件を満たす解を取り出し、選択することが可能である。

本論文ではタブーリストを用いた、共有判定を提案したが、多くの共有・削除可能なノードが検出できていない。今後は更に効率的に ZDD を構築するために、それらのノードを可能な限り検出できるような新たなけ n

References

[1] Kawahara, Jun and Inoue, Takeru and Iwashita, Hiroaki and Minato, Shin-ichi, "Frontier-based search for enumerating all constrained subgraphs

with compressed representation" *TCS-TR-A-14-76. Hokkaido University*, Citeseer, 2014.

[2] Minato, Shin-ichi "Zero-suppressed BDDs for set manipulation in combinatorial problems" *Proceedings of the 30th international Design Automation Conference 272–277*, ACM, 1993.

[3] 戸田貴久, 斎藤寿樹, 岩下洋哲, 川原純, 湊真一"ZDDと列挙問題ー最新の技法とプログラミングツール" コンピュータソフトウェア *Vol.34 3_97-3_120*, 2017.

[4] Donald, E Knuth and others "The art of computer programming" *Sorting and searching Vol.3 426–458*, 1999.

[5] Bespamyatnikh, Sergei "An efficient algorithm for enumeration of triangulations" *Computational Geometry Vol.23 p271-279*, 2002.

[6] Katoh, Naoki, and Shin-ichi Tanigawa "Enumerating edge-constrained triangulations and edge-constrained non-crossing geometric spanning trees" *Discrete Applied Mathematics Vol.157 p3569–3585*, 2009.

[7] Katsuhisa Yamanaka, Masatoshi Murakami, Takashi Hirayama, and Nishitani yasuaki "On the number of edge-constrained triangulations without the general position assumption" *The 30th Workshop on Circuits and Systems (KWS30)*, 2017.

[8] Takanori Maehara, Hirofumi Suzuki, Masakazu Ishihata "Exact Computation of Influence Spread by Binary Decision Diagrams" *The 2017 World Wide Web conference (WWW-2017)*, pp. 947-956, April, 2017.