Vol.2021-AL-182 No.5
2021/3/17

動的計画法に基づく Simple Polygonization 列挙アルゴリズムの実験的評価

中畑 裕[1,a)]　堀山 貴史[2]　湊 真一[1]　山中 克久[3]

概要：平面上の n 点に対し，それら n 点を頂点とする単純多角形を simple polygonization という．平面上の n 点が与えられたとき，simple polygonization すべてを多項式遅延（解 1 つあたり多項式時間）で列挙できるかは計算幾何学分野における重要な未解決問題である．山中らは，simple polygonization を拡張した surrounding polygon に対し，逆探索法に基づく多項式遅延の手法を与えた．山中らはこの手法を実験的に評価し，凸包内部の点が少ない点集合に対して高速に動作することを報告したが，理論的裏付けは示されていない．一方，中畑らは，動的計画法に基づく指数時間の前処理ののち，simple polygonization を多項式遅延で列挙する手法を与えたが，実験的評価は行われていない．本稿では，山中らの手法が凸包内部の点が少ない点集合に対して高速に動作することを裏付ける理論的解析を与える．また，中畑らの手法を効率化するための枝刈りを提案し，その枝刈りを用いることで，中畑らの手法を同様の入力に適用した際の計算量の上界を指数的に改善できることを理論的に示す．中畑らの手法を実験的に評価し，提案した枝刈りの効果を調べるとともに，山中らの手法や全探索法との比較を行う．

キーワード：simple polygonization，計算幾何学，列挙，多項式遅延，動的計画法

Experimental Evaluation of a Dynamic-Programming-Based Algorithm for Enumerating Simple Polygonizations

Yu Nakahata[1,a)]　Takashi Horiyama[2]　Shin-ichi Minato[1]　Katsuhisa Yamanaka[3]

1. はじめに

P を平面上の n 点の集合とする．ただし P のどの異なる 3 点も一直線上にないとする．P の simple polygonization とは，頂点集合がちょうど P であるような単純多角形である．図 1 の P に対し，図の最下段にすべての simple polygonization を示す．simple polygonization は計算幾何学においてよく研究されている．たとえば，n 点の集合に対する simple polygonization の個数の最大値に対する上界，下界が研究されており [1]，現在知られている最良の上界は $\mathrm{O}(54.55^n)$ [2]，下界は $\Omega(4.64^n)$ [3] である．また，点集合 P が与えられたとき，P の simple polygonization の個数を具体的に求める問題も研究されている．現在最速のアルゴリズムは Marx and Miltzow [4] による $n^{\mathrm{O}(\sqrt{n})}$ 時間のアルゴリズムである．いくつかの特殊ケースについては多項式時間アルゴリズムが知られている [5], [6] が，一般にこの問題が多項式時間で解けるかは未解決である [7].

本稿では，平面上の点集合 P が与えられたとき，P の simple polygonization すべてをもれなく重複なく列挙する問題を考える．この問題に対し，多項式遅延（解 1 つあたり多項式時間）のアルゴリズムが存在するかは未解決である [8]．山中ら [9] は，simple polygonization を一般化した surrounding polygon に対して，多項式遅延のアルゴリズムを与えた．単純多角形 S が P の surrounding polygon であ

[1] 京都大学大学院情報学研究科
Graduate School of Informatics, Kyoto University
[2] 北海道大学大学院情報科学研究院
Graduate School of Information Science and Technology, Hokkaido University
[3] 岩手大学理工学部
Faculty of Science and Engineering, Iwate University
[a)] nakahata.yu.27e@st.kyoto-u.ac.jp

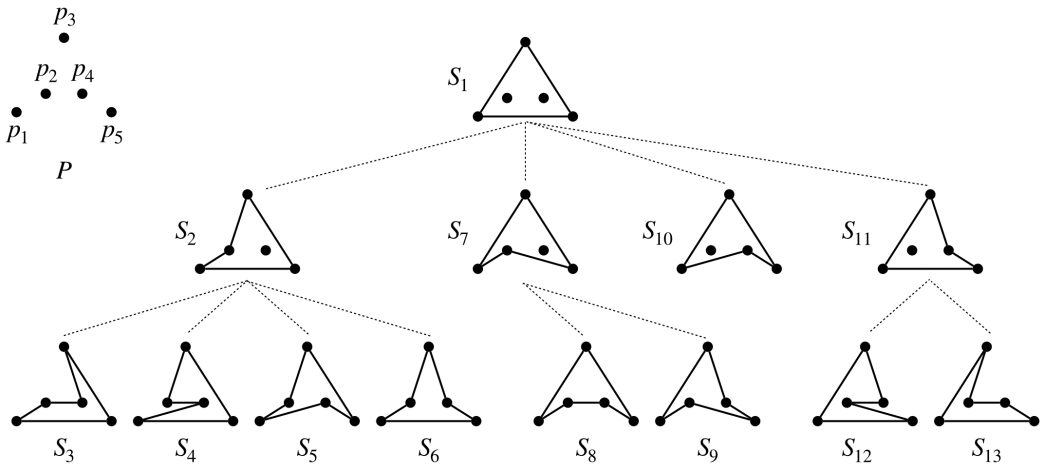


図 1: surrounding polygon に対する家系木.

るとは，S の各頂点が P に属し，かつ P のうち S の頂点でない点はすべて S の内側にあることをいう．図 1 の P に対するすべての surrounding polygon は図中の $S_1, \ldots, S_{13}$ である．surrounding polygon は simple polygonization の一般化なので，前者をすべて列挙すれば後者もすべて列挙できる．ただしこのアルゴリズムは simple polygonization に対して多項式遅延とは限らない．山中らはこの手法を実験的に評価し，凸包内部の点が少ない点集合に対しては surrounding polygon や simple polygonization の数が少なく，結果としてアルゴリズムが高速に動作することを報告したが，理論的裏付けは示されていない．

一方，中畑ら [10] は，動的計画法に基づく異なるアプローチを提案した．この手法は，指数時間の前処理ののちに simple polygonization を多項式遅延で列挙できる．より正確には，指数時間・領域で simple polygonization の集合を表す有向非巡回グラフ (DAG) を構築する．この DAG を用いることで，simple polygonization を線形遅延で列挙できる．また，列挙だけでなく，DAG の大きさに対する多項式時間で，数え上げ，ランダムサンプリング，最適化も可能である [10]．この手法はまだ実装評価されていない．また，どのような入力に対して効率よく動作するかの理論的解析もなされていない．

本稿では，山中らの手法が凸包内部の点が少ない点集合に対して高速に動作することを裏付ける理論的解析を与える．また，中畑らの手法を効率化するための枝刈りを提案し，その枝刈りを用いることで，中畑らの手法を同様の入力に適用した際の計算量の上界を指数的に改善できることを理論的に示す．中畑らの手法を実験的に評価し，提案した枝刈りの効果を調べるとともに，山中らの手法や全探索法との比較を行う．

本稿の構成は次の通りである．2 節で記法を導入し，3 節で山中らの手法を，4 節で中畑らの手法を説明する．5 節で中畑らの手法を効率化するための枝刈りを提案する．6 節で山中らの手法と中畑らの手法について，凸包内部の点が少ない点集合に対する理論的解析を与える．7 節で実験結果を示し，8 節で本稿をまとめる．

2. 準備

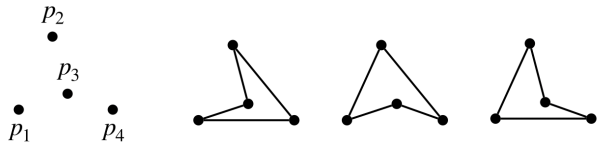
平面上の n 点の集合を P とする．本稿を通し，P のどの異なる 3 点も一直線上にないとする．また，P のどの異なる 2 点も x 座標が等しくないとする．これらの仮定のもとで，P の点は x 座標の昇順に一意に $p_1, \ldots, p_n$ と順序づけられる．たとえば図 1 の P はこのように順序づけられている．P の異なる 2 点 p_i, p_j を端点とする線分を e_{ij} とし，E_P を P 上の異なる 2 点を端点とする線分すべてからなる集合とする．線分の集合 $E' \subseteq E_P$ における点 $p \in P$ の**次数**とは，p を端点とする E' 中の線分の個数である．

3. 山中らの手法

本節では山中らの手法 [9] を説明する．山中らの手法は，simple polygonization を一般化した surrounding polygon を多項式遅延で列挙する手法である．1 節で述べたように，単純多角形 S が P の surrounding polygon であるとは，S の各頂点が P に属し，かつ P のうち S の頂点でない点はすべて S の内側にあることをいう．図 1 が示すように，surrounding polygon は simple polygonization や凸包の一般化である．

山中らの手法は列挙アルゴリズムの一般的な枠組みである逆探索法 [11] に基づく．この手法では surrounding polygon 間に親子関係を定め，その親子関係が定める木（**家系木**）を深さ優先探索することですべての surrounding polygon を列挙する．

surrounding polygon 間の親子関係を定義する．P の surrounding polygon S を，その頂点を反時計周りに並べた（巡回する）列 $\left\langle p'_1, \ldots, p'_{|S|} \right\rangle$ で表す．S の頂点 p が**埋め込み可能**であるとは，p とその前後の頂点 $\mathsf{pred}(p), \mathsf{succ}(p)$ が囲む三角形が S の内部と交わりをもたないことをいう．S が凸包でない限り埋め込み可能な頂点は存在する [9].



(a) 点集合 P. (b) P のすべての simple polygonization.

図 2: 点集合 P とすべての simple polygonization.

p が埋め込み可能なとき，p に対する**埋め込み操作**とは，S から線分 $(\mathsf{pred}(p), p), (p, \mathsf{succ}(p))$ を削除し新たな線分 $(\mathsf{pred}(p), \mathsf{succ}(p))$ を追加する操作である．たとえば図 1 の S_2 において，p_2 に対し埋め込み操作を行うと S_1 が得られる．このように，埋め込み操作によって新たな surrounding polygon が得られる．S の親を，S の埋め込み可能な添字最大の頂点に対して埋め込み操作を行って得られる surrounding polygon と定める．このように定めた親子関係において，どんな surrounding polygon も親への変形を繰り返すことで凸包に変形できることが示せる [9] ので，親子関係は凸包を根として surrounding polygon 全体を張るような木をなす．この根付き木を surrounding polygon に対する家系木という．図 1 に入力の点集合 P と家系木の例を示す．家系木を根から順に深さ優先探索することですべての surrouding polygon を列挙でき，ゆえにすべての simple polygonization も列挙できる．

家系木を根から深さ優先探索するために，埋め込み操作の逆操作である**掘削操作**を定義する．surrounding polygon S の辺 $(p', \mathsf{succ}(p'))$ と S 内部の点 p に対し，S から線分 $(p', \mathsf{succ}(p'))$ を削除し新たな線分 $(p', p), (p, \mathsf{succ}(p'))$ を追加して得られる多角形を $S(p', p)$ とおく．S から $S(p', p)$ を得る操作を掘削操作という．直感的には，掘削操作は p に対する埋め込み操作の逆操作である．ただし $S(p', p)$ は surrounding polygon とは限らない．そこでアルゴリズムでは S の可能な掘削操作を全通り試し，surrounding polygon であって親が S であるものについてのみ探索を続行する．この処理を家系木の根である凸包から順に深さ優先に行うことで，すべての surrounding polygon が列挙できる．

この手法の理論計算量は（surrounding polygon 1 つあたり）$\mathrm{O}(n^2 \log n)$ 遅延 $\mathrm{O}(n^2)$ 領域である．しかし定数倍のオーバーヘッドを避けるため本稿では $\mathrm{O}(n^3)$ 遅延の実装を行った．

4. 中畑らの手法

本節では中畑らの手法 [10] を説明する．中畑らの手法は，動的計画法に基づき指数時間かけて simple polygonization の集合を表す DAG を構築する．その DAG を用いることで simple polygonization を多項式遅延で列挙できる．以下ではまず構築する DAG がみたすべき条件を 4.1 節で説明し，次にそのような DAG を構築するためのアルゴリズムを 4.2 節で説明する．

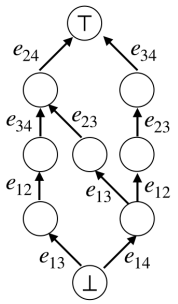


図 3: simple polygonization の集合を表す DAG.

4.1 構築する DAG がみたすべき条件

本小節では，中畑らの手法が構築する DAG がみたすべき条件を説明する．DAG は，入次数 0 の頂点 $\perp$ と出次数 0 の頂点 $\top$ をちょうど 1 つずつ持つ．直感的には，$\perp$ は初期状態，すなわち空集合に対応し，$\top$ は simple polygonization が完成した状態に対応する．辺には E_P の元がラベルとして対応づけられており，次のすべての条件をみたす: (1) ある頂点から出る 2 つの辺は同じラベルをもたず，(2) 任意の $\perp$-$\top$ パスにおいて，同じラベルが 2 度あらわれることはなく，かつ (3) $\perp$-$\top$ パスと P の simple polygonization とが 1 対 1 に対応している．図 3 に，図 2(a) の点集合に対するすべての simple polygonization の集合（図 2(b)）を表す DAG を示す．この DAG は (1) から (3) の条件をすべてみたす．たとえば図 3 における最左のパス $\perp \to e_{13} \to e_{12} \to e_{34} \to e_{24} \to \top$ は，図 2(b) 最左の simple polygonization と対応している．このような DAG を構築できると，simple polygonization の列挙はこの DAG 上の $\perp$-$\top$ パスの列挙と等価であり，深さ優先探索によって $\mathrm{O}(n)$ 遅延で列挙できる．同様に，数え上げ，ランダムサンプリング，最適化も DAG のサイズに対して多項式時間で行える [10].

4.2 DAG の構築アルゴリズム

本小節では，4.1 節の条件をみたす DAG を構築するアルゴリズムを説明する．アルゴリズムは，空集合から初めて，線分を 1 本ずつ追加していくことで simple polygonization を構成する．その過程で，動的計画法に基づき等価な状態を共有する．しかし，この方法では同じ解を重複して何度も調べてしまう可能性がある．そこで一意化のために，線分は下から上，左から右の順で追加するという規則を設ける．この規則を設けることで，線分の非交差性も同時に扱える [12]．図 3 の任意の $\perp$-$\top$ パスはこの規則をみたしている．$E' \subseteq E_P$ に対して，ある simple polygonization $C \subseteq E_P$ が存在して E' を規則に従った線分の追加によって C に変形できるとき，E' は**拡張可能**であるという．ア

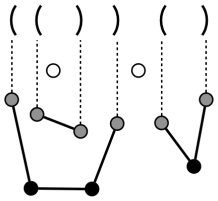


図 4: 線分の集合 E' とその状態．実線は E' 中の線分を表す．白，灰，黒の点はそれぞれ次数 0, 1, 2 の点を表す．点線は次数 1 の点（パスの端点）と括弧の対応を表す．

ルゴリズムの過程で，拡張可能でないとわかった状態については枝刈りを行うことで計算量を削減する．

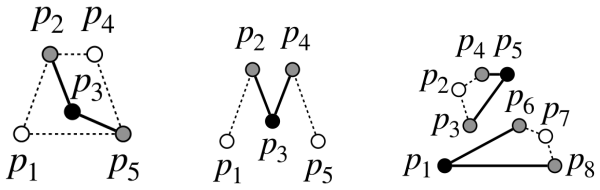
中畑らの手法は動的計画法に基づくため，状態の定義が重要である．$E' \subseteq E_P$ に対し，E' の**状態** $s(E')$ を対 (m, v) で定義する．ここで $m \in \{0, \ldots, n\}$ は，$E' = \emptyset$ のときは $m = 0$, $E' \neq \emptyset$ のときは E' において最後に追加された線分の左側の端点の添字である．また，v は長さ n の列である．各 $i \in \{1, \ldots, n\}$ に対し，$v[i] \in \{\circ, (,), \bullet\}$ は点 $p_i \in P$ の状態を表し，それぞれ以下の意味である:

- $v[i] =$ ‘$\circ$’ のとき p_i は次数 0 である．
- $v[i] =$ ‘(’ のとき p_i は次数 1，すなわちパスの端点であり，同じパスのもう一方の端点より左側にある．
- $v[i] =$ ‘)’ のとき p_i は次数 1，すなわちパスの端点であり，同じパスのもう一方の端点より右側にある．
- $v[i] =$ ‘$\bullet$’ のとき p_i は次数 2 である．

2 節で P のどの異なる 2 点も x 座標が等しくないと仮定したので，各 $E' \subseteq E_P$ に対し，p_i の次数が 1 のとき $v[i]$ が ‘(’ であるか ‘)’ であるかは一意に定まる．さらに，v の ‘(’, ‘)’ のみに注目すると，バランスの取れた括弧列をなす．バランスの取れた括弧列はパスの端点の対応関係を表している．図 4 にその例を示す．任意の $E'_1, E'_2 \subseteq E_P$ に対し，$s(E'_1) = s(E'_2)$ であれば，今後 E'_1, E'_2 にどのように線分を追加したら simple polygonization となるかが等価であることがいえる [10]．よって定義した状態を用いて動的計画法が実行できる．状態の数は m は $\mathrm{O}(n)$ 通り，v は各 $p \in P$ について 4 通りより全体で $\mathrm{O}(4^n n)$ 通りである．**状態 s が拡張可能**であるとは，状態が s となるような任意の $E' \subseteq E_P$ が拡張可能であることをいう．

アルゴリズムは初期状態（空集合）に対応する頂点 $\perp$ から順に幅優先的に DAG を構築する．その過程で，拡張可能でないとわかった状態については枝刈りを行う．しかしこの枝刈りは完全ではなく，DAG 構築後に $\top$ に到達可能でないとわかる頂点も一般には存在する．そのような不要な頂点の作成を減らすための枝刈りを 5 節で提案する．

DAG 構築アルゴリズムの理論計算量は $\mathrm{O}\left(4^n n^4\right)$ 時間・領域である．4.1 節で述べたように，構築された DAG を用いて $\mathrm{O}(n)$ 遅延で simple polygonization を列挙できる．



(a) 凸包による枝刈りが可能な例．　(b) 次数による枝刈りが可能な例．　(c) 連結性による枝刈りが可能な例．

図 5: 枝刈りが可能な例．実線は追加済みの線分を，点線は今後追加できる線分を表す．白，灰，黒の点はそれぞれ次数 0,1,2 の点を表す．

5. 中畑らの手法に対する枝刈りの提案

本節では，中畑らの手法を効率化するための枝刈りを提案する．

5.1 凸包による枝刈り

本小節では凸包による枝刈りを提案する．図 5(a) の状態を考える．図中の実線は追加済みの線分を，点線は凸包の辺を表す．図では点 p_1, p_4 が p_2, p_5 を端点とするパスによって「分断」されている．そのため，p_1 と p_4 を通る 1 つの単純多角形をつくることができず，拡張可能でないため枝刈りできる．この状態を凸包を用いて判定する．

P の凸包上の点を反時計回りに $p'_1, \ldots, p'_k$ とおく．凸包上の隣り合わない 2 点 p'_i, p'_j が同じパスの端点であるとする．凸包から p'_i, p'_j を除くと，凸包は 2 つの部分にわかれる．それぞれに属する点の集合を $H^1_{i,j} = \left\{p'_{(i+1) \bmod k}, \ldots, p'_{(j+k-1) \bmod k}\right\}$, $H^2_{i,j} = \left\{p'_{(j+1) \bmod k}, \ldots, p'_{(i+k-1) \bmod k}\right\}$ とおく．ここで $a \bmod b$ は a を b で割った余りを表す．図 5(a) の状態は，次の補題の条件をみたさないので枝刈り可能であると言える．

補題 1. 拡張可能な状態 s において，P の凸包上の隣り合わない *2* 点 p'_i, p'_j が同じパスの端点であるとき，$H^1_{i,j}, H^2_{i,j}$ の両方に次数 *2* 未満の点が存在することはない．

この判定は $\mathrm{O}\left(n^3\right)$ 時間で可能であり，アルゴリズムの計算時間をオーダーの意味で増加させない．なぜなら中畑らの手法の計算量 $\mathrm{O}\left(4^n n^4\right)$ は，$\mathrm{O}(4^n n)$ 個の状態ごとに $\mathrm{O}\left(n^3\right)$ 時間の計算を行っているとみなせるからである．この枝刈りは凸包上の点が多い場合に特に効果が大きいと考えられる．6.2 節ではそのような入力に対し，この枝刈りを加えることで計算量の上界を指数的に改善できることを理論的に示す．

なお，図 5(a) の例はこれから示す補題 1–3 のうち補題 1 の条件でのみ枝刈り可能である．同様に，後に示す図 5(b),

5(c) はそれぞれ補題 2, 3 の条件でのみ枝刈り可能である．

5.2 次数による枝刈り

本小節では次数による枝刈りを提案する．状態 s において，線分 e_{ij} が以下の条件のいずれかをみたすとき，e_{ij} は今後ずっと追加できない: (1) e_{ij} の上側に次数 1 以上の点が存在する，(2) p_i または p_j が次数 2 である，または (3) p_i, p_j が同じパスの端点であり，かつ e_{ij} を追加しても simple polygonization が完成しない（孤立したサイクルが発生する）．これらの条件を少なくとも 1 つみたす線分の集合を $E_{\mathrm{dead}}(s)$ とおくと，次の補題が成り立つ．ただし以下で，$p_i \in P$ の状態 s における次数とは，状態が s となるような $E' \subseteq E_P$ における p_i の次数である．

補題 2. 状態 s が拡張可能ならば，すべての点 $p \in P$ に対し，p の s における次数と $E \setminus E_{\mathrm{dead}}(s)$ における次数の和が *2* 以上である．

補題 2 の条件をみたさないとき状態 s は拡張可能ではないため枝刈りできる．たとえば図 5(b) の状態を考える．今 p_1 は次数 0 であるが，p_1 に接続する線分のうち，今後採用できるのは e_{12} のみである．なぜならば，e_{13} は端点 p_3 が次数 2 のため $E_{\mathrm{dead}}(s)$ に属し，e_{14}, e_{15} は次数 1 の点 p_2 の下を通るため $E_{\mathrm{dead}}(s)$ に属するからである．よって p_1 が補題 2 の条件に反するため枝刈りできる．なお，p_5 についても同様である．

補題 2 の条件を判定するには $\mathrm{O}(n^2)$ 個の各線分について $E_{\mathrm{dead}}(s)$ に属するかの判定が必要であり，これは線分ごとに $\mathrm{O}(n)$ 時間で行える．よって全体の判定は状態ごとに $\mathrm{O}(n^3)$ 時間で可能であり，5.1 節の議論と同様にアルゴリズムの計算量のオーダーを増加させないことが言える．

5.3 連結性による枝刈り

本小節では連結性による枝刈りを提案する．この枝刈りは 5.2 節と同様の $E_{\mathrm{dead}}(s)$ を用いる．

補題 3. 状態 s が拡張可能ならば，グラフ $G(s) = (P_{\leq 1}(s), (E_P \setminus E_{\mathrm{dead}}(s)) \cup E_{\mathrm{pair}}(s))$ は連結である．ただし $P_{\leq 1}(s)$ は s における次数が *1* 以下の点の集合であり，$E_{\mathrm{pair}}(v)$ は s において同じパスの端点である点どうしを結ぶ辺の集合である．

図 5(c) に枝刈り可能な例を示す．この例では $G(s) = (\{p_2, p_3, p_4, p_6, p_7, p_8\}, \{e_{23}, e_{24}, e_{34}, e_{67}, e_{68}, e_{78}\})$ となるが，p_2, p_3, p_4 と p_6, p_7, p_8 とが非連結であり，今後 1 つの単純多角形にできないため枝刈りできる．この枝刈りは 5.2 節と同様の議論から状態ごとに $\mathrm{O}(n^3)$ 時間で可能であり，アルゴリズムの計算量のオーダーを増加させない．

6. 理論的解析

本節では，凸包内部の点が少ない場合に山中らの手法と枝刈りを加えた中畑らの手法が効率よく動作することを裏付ける理論的解析を与える．以下では入力の点集合における凸包上の点の個数を k, 凸包内部の点の個数を r とおく．

6.1 山中らの手法

山中ら [9] は入力の点の個数を固定したとき，凸包内部の点が少ないほど simple polygonization や surrounding polygon の数が少ないという実験結果を報告した．次の定理はこの実験結果に対する理論的説明を与えるものであり，凸包内部の点の個数 r が定数のとき, simple polygonization や surrounding polygon が多項式個しかないことを述べている．以下で $\#\mathsf{simp}(P), \#\mathsf{surp}(P)$ はそれぞれ P の simple polygonization, surrounding polygon の個数を表す．

定理 1. $\#\mathsf{simp}(P) = \mathrm{O}(r!n^r)$ かつ $\#\mathsf{surp}(P) = \mathrm{O}(r \cdot r!n^r)$ が成り立つ．

Proof. P の任意の surrounding polygon は P の凸包上の点をすべて頂点としてもつ．$i = 0, \ldots, r$ に対し, $\#\mathsf{surp}_i(P)$ を頂点が $k+i$ 個である（凸包内部の点を i 個頂点としてもつ）P の surrounding polygon の個数とする．このとき

$$\#\mathsf{surp}_i(P) \leq \prod_{j=0}^{i-1}(k+j)(r-j) \qquad (1)$$

が成り立つことを帰納法により示す．$i = 0$ のとき $\#\mathsf{surp}_0(P) = 1$（凸包のみ）より成り立つ．ある $i < r$ について式 (1) が成り立つと仮定する．家系木において頂点が $k + (i+1)$ 個である任意の surrounding polygon は，頂点が $k+i$ 個である surrounding polygon の子である．頂点が $k+i$ 個である任意の surrounding polygon S について，S の子の数は，S に掘削操作を行って得られる $S(p', p)$ の数でおさえられる．これは S の頂点 p' が $k+i$ 通り，S 内部の点 p が $r-i$ 通りより高々 $(k+i)(r-i)$ 通りである．よって $\#\mathsf{surp}_{i+1}(P) \leq \#\mathsf{surp}_i(P) \cdot (k+i)(r-i) = \prod_{j=0}^{i}(k+j)(r-j)$ が成り立つ．ゆえに $i+1$ についても式 (1) が成立し，帰納法よりすべての $i = 0, \ldots, r$ について式 (1) が成り立つ．$\#\mathsf{simp}(P) = \#\mathsf{surp}_r(P)$ より $\#\mathsf{simp}(P) \leq \prod_{j=0}^{r-1}(k+j)(r-j) = \mathrm{O}(r!n^r)$ が従う．また，$\#\mathsf{surp}(P) = \sum_{i=0}^{r} \#\mathsf{surp}_i(P) \leq 1 + \sum_{i=1}^{r} \#\mathsf{surp}_i(P) \leq 1 + r\prod_{j=0}^{r-1}(k+j)(r-j) = \mathrm{O}(r \cdot r!n^r)$ が成り立つ． □

6.2 中畑らの手法

中畑らの手法における状態数は $\mathrm{O}(4^n n)$ である．次の定理は，中畑らの手法に補題 1 の凸包による枝刈りを加えることで，凸包内部の点の個数 r が定数の場合には状態数を $\mathrm{O}(3^n n)$ へと指数的に改善できることを示している．

定理 2. 中畑らの手法に補題 *1* の枝刈りを加えたとき，状態数は $\mathrm{O}(4^r \cdot 3^n n)$ となる．

Proof. 状態 (m, v) の種類数を考える．m は $\mathrm{O}(n)$ 通りである．v において，各点の次数は $0, 1, 2$ の 3 通りなので，

次数だけを考えると v の種類数は $O(3^n)$ 通りである．そのそれぞれについて，次数 1 の点がどのようにペアになっているか（どの点とどの点が同じパスの端点となっているか）を考える．次数 1 の点は偶数個とする．パスのうち，凸包内部の点を少なくとも一方の端点とするものは高々 r 個である．それらのパスの端点（高々 $2r$ 個）のペアへの分け方は，パスどうしが非交差であることから長さが高々 $2r$ の括弧列で表現できるので $O(2^{2r}) = O(4^r)$ 通りである．残る次数 1 の点は偶数個で，すべて凸包上にあり，凸包上の点どうしでペアとなる．これらの点のペアへの分け方が高々 2 通りであることを示す．これらの点を反時計回りの巡回する列 $H' = \langle p'_1, \ldots, p'_i \rangle$ で表す．i は偶数である．補題 1 の枝刈りより，H' において隣り合わない点どうしがペアになることはない．よって H' のペアへの分け方は $(p'_1, p'_2), (p'_3, p'_4), \ldots, (p'_{i-1}, p'_i)$ または $(p'_2, p'_3), (p'_4, p'_5), \ldots, (p'_i, p'_1)$ の高々 2 通りである．以上より状態数は $O(2 \cdot 4^r \cdot 3^n \cdot n) = O(4^r \cdot 3^n n)$ である． □

7. 実験

本節で用いるプログラムはすべて C++で実装し，g++ 5.4.0 で -O3 オプション付きでコンパイルした．実行環境は Intel Xeon(R) W-2133 CPU 3.60GHz x 6 processor, 256 GB, CentOS Linux release 7.6.1810 (Redhat) である．

7.1 中畑らの手法と枝刈りの効果の分析

中畑らの手法の性能と枝刈りの効果を分析するため，山中ら [9] と同様の入力で実験を行った．入力は単位正方形内のランダムな点集合であり，$k = 3, 4, \ldots, 12$, $r = 0, 1, \ldots, 9$, $k + r \leq 12$ をみたす各 (k, r) について，凸包上の点が k 個，凸包内部の点が r 個であるような点集合を 100 個ずつ生成した．枝刈りの効果を調べるため，動的計画法の状態数について次の 3 つの値を調べた: (1) 5 節の枝刈りをどれも加えなかったときの状態数，(2) 5 節の枝刈りをすべて加えたときの状態数，(3) 完全に枝刈りができたと仮定したときの状態数．DAG 構築後の simple polygonization の列挙は枝刈りの有無によらず同じ手続きなので，この実験ではアルゴリズムは DAG の構築までのみ実行した．

表 1 に実験結果を示す．表の各セルは 3 つの値を含み，上から順に (1) 枝刈りなしのときの状態数 $\#\mathsf{node}(k, r)$, (2) 枝刈りありのときの状態数 $\#\mathsf{prune}(k, r)$, (3) 完全に枝刈りができたと仮定したときの状態数 $\#\mathsf{esse}(k, r)$ である．値はどれも各 (k, r) に対する 100 ケースの平均値であり，小数部分は切り捨てている．

表から次の基本的な傾向がわかる．$\#\mathsf{node}(k, r)$, $\#\mathsf{prune}(k, r)$, $\#\mathsf{esse}(k, r)$ のどの値も，k を固定すると r の増加につれて増加し，r を固定すると k の増加につれて増加する．また，$k + r$ を固定すると，k の減少（r の増加）につれて，3 つの値は増加する傾向がある．これらの傾向は，山中らが報告した simple polygonization や surrounding polygon の数の傾向 [9] と同様である．

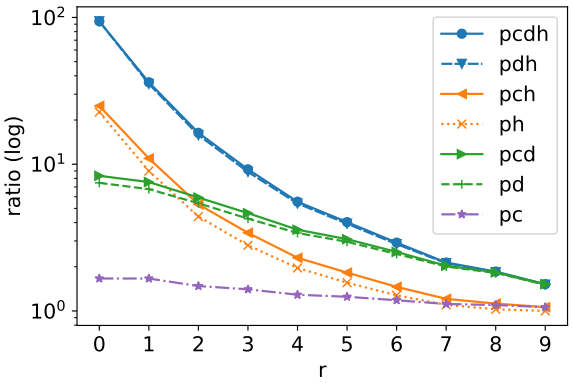


図 6: 枝刈りごとの効果.

枝刈りごとの効果を調べるため，$n = 12$ の点集合に対し，凸包内部の点の個数 $r = 0, \ldots, 9$ を変化させたときのノード数を調べた．結果を図 6 に示す．図の横軸は凸包内部の点の個数 r であり，縦軸は枝刈りありのときの状態数に対する枝刈りなしのときの状態数の比の対数である．以下では凸包，次数，連結性による枝刈りをそれぞれ ph, pd, pc と書き，たとえば連結性と次数の枝刈りを合わせたものを pcd のように書くことにする．図では pc, pd, ph, pdh を点線で示し，pd, ph, pdh のそれぞれに pc を追加した pcd, pch, pcdh を同色の実線で描いている．枝刈りを 1 つだけ用いる pc, pd, ph の効果を見ると，pc は他の枝刈りに比べて効果が小さい．$r \leq 1$ のときは ph が最も効果があるが，r の増加につれて pd よりも ph の方が効果が減っていく勢いが速いため逆転し，$r \geq 2$ のときは pd が最も効果がある．なお，$r = 9$ のときは $k = 3$ であり凸包の分断が起こらないため ph は一切効果がない．pd, ph, pdh に pc を追加した pcd, pch, pcdh は pc を追加しないときに比べて大きな差はないが若干効果が上がった．

7.2 手法間の比較

simple polygonization 列挙について，中畑らの手法を山中らの手法および全探索法と比較する．中畑らの手法は DAG 構築後 DAG 上の深さ優先探索によりすべての simple polygonization を出力する．山中らの手法は surrounding polygon 全体に対する家系木を深さ優先探索するが，家系木における深さ最大の葉が simple polygonization に対応するので，そのような葉に到達するたびに simple polygonization を出力する．全探索法は頂点の順列を全通り試し，simple polygonization に対応するものをすべて出力する．本節では中畑らの手法に枝刈りを加えないものを DP，加えたものを DP_PRUNE と呼び，山中らの手法を RVS，全探索法を NAIVE と呼ぶ．

入力の点集合としては次の 2 種類を用いた．1 つ目は単

表 1: 実験結果．表の各セルは上から順に，枝刈りをしなかったときの状態数，枝刈りをしたときの状態数，完全に枝刈りができたと仮定したときの状態数を表す．$k = 3, 4, \ldots, 12$ は凸包上の点の個数，$r = 0, 1, \ldots, 9$ は凸包内部の点の個数である．小数部分は切り捨てている．

k	r									
	0	1	2	3	4	5	6	7	8	9
	5	19	70	245	820	2591	8178	25588	80819	244420
3	4	15	53	176	575	1757	5561	17174	53648	160680
	4	10	24	62	148	348	806	1852	4179	9299
	15	56	197	628	2006	6935	20635	63618	222264	-
4	7	30	108	362	1168	3839	11616	36300	119787	-
	5	14	37	93	220	513	1157	2536	5706	-
	47	158	535	1688	5797	15464	57169	162030	-	-
5	13	56	209	706	2456	7279	24767	75793	-	-
	6	19	52	130	314	714	1611	3753	-	-
	133	481	1443	4828	15486	49303	148491	-	-	-
6	22	105	387	1407	4630	15839	50826	-	-	-
	7	24	69	176	435	998	2268	-	-	-
	425	1268	4337	12918	37915	122895	-	-	-	-
7	40	194	757	2614	8820	30488	-	-	-	-
	8	29	88	228	561	1287	-	-	-	-
	1250	3279	10965	33536	95246	-	-	-	-	-
8	71	323	1351	5207	17160	-	-	-	-	-
	9	35	109	292	717	-	-	-	-	-
	2554	8570	30898	95659	-	-	-	-	-	-
9	114	613	2571	10364	-	-	-	-	-	-
	10	40	130	358	-	-	-	-	-	-
	8494	23243	78315	-	-	-	-	-	-	-
10	210	1082	4789	-	-	-	-	-	-	-
	11	46	156	-	-	-	-	-	-	-
	21287	76845	-	-	-	-	-	-	-	-
11	359	2125	-	-	-	-	-	-	-	-
	12	53	-	-	-	-	-	-	-	-
	56844	-	-	-	-	-	-	-	-	-
12	604	-	-	-	-	-	-	-	-	-
	13	-	-	-	-	-	-	-	-	-

位正方形内のランダムな n 点の集合である．この点集合は $10 \leq n \leq 16$ をみたす各 n に対して 100 ケースずつ生成した．2 つ目は n 点の double chain [3] であり，双曲線上に等間隔に点を置いたような配置である．double chain は simple polygonization の個数 $\#\mathsf{simp}(P)$ に対して現在知られている最良の下界を達成する点集合であり [3]，ランダムな点集合より $\#\mathsf{simp}(P)$ が大きくなることが予想される．double chain において，$\#\mathsf{simp}(P)$ は n にのみ依存し，具体的な点の座標に依存しない [3]．しかし中畑らの手法は具体的な点の座標を枝刈りのために利用するため，点集合全体をランダムに回転させたものを $4 \leq n \leq 16$ をみたす偶数の各 n について 100 ケースずつ生成した．

計算時間の比較を図 7, 8 に示す．図 7 はランダムな点集合，図 8 は double chain に対するグラフである．グラフの横軸は入力の点の個数，縦軸は計算時間（秒）の対数である．タイムアウトは 10 分とし，10 分以上かかったケースは 10 分かかったと仮定して計算時間の平均値をとった．

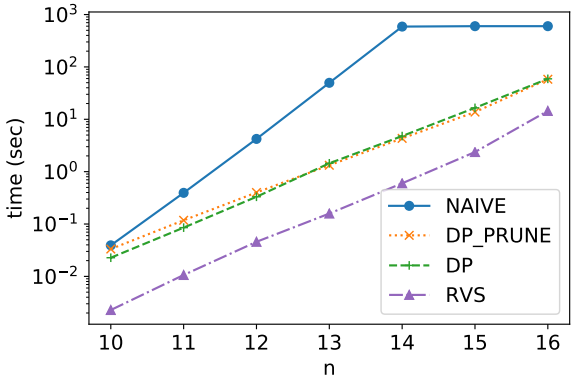


図 7: ランダムな点集合に対する計算時間の比較．

入力の点集合がどちらの種類でも，計算時間は概ね RVS < DP < DP_PRUNE < NAIVE となる傾向が見られた．DP と DP_PRUNE については枝刈りにより不要なノードの作成が減り DP_PRUNE < DP となることを期待したが，今回の実験でその傾向が見られたのはランダムな点

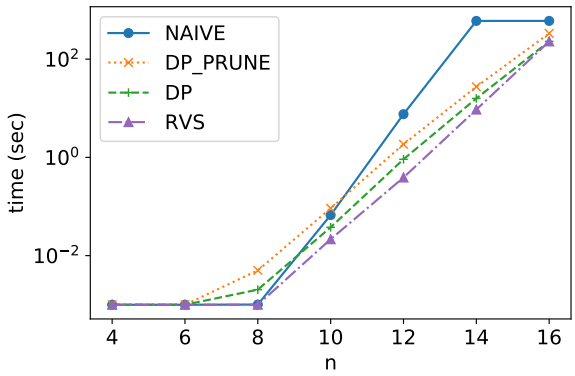


図 8: double chain に対する計算時間の比較.

集合の $n \geq 13$ のときのみだった．ランダムな点集合では $n \leq 12$ のとき DP $<$ DP_PRUNE, $n \geq 13$ のとき DP $>$ DP_PRUNE と計算時間が逆転している．これは n が小さい範囲では枝刈りで減るノード数が小さく，計算時間のオーバーヘッドが大きいためである．double chain では DP_PRUNE の DP に対するオーバーヘッドがランダムな点集合より顕著に見られる．この理由は double chain の凸包上の点が常に 4 点と少ないためである．

NAIVE は $n \geq 14$ のすべてのケースでタイムアウトとなったが，他の手法はすべてのケースで 10 分以内に終了した．DP, DP_PRUNE と RVS を比較すると，ランダムな点集合では前者の方が後者より $n = 10$ で約 15 倍，$n = 16$ で約 5 倍の時間がかかった．両者の差は n が大きくなるほど小さくなる傾向が見られるため，n がより大きい範囲では前者の方が後者より速くなる可能性がある．入力が double chain のときはランダムな点集合のときより DP, DP_PRUNE と RVS との差が小さい．この理由は，double chain においては $\#\mathsf{simp}(P)$ や $\#\mathsf{surp}(P)$ が大きいため RVS は時間がかかるが，DP, DP_PRUNE は状態をまとめることで効率よく処理できているからだと考えられる．また $n = 16$ では，DP, DP_PRUNE が DAG の構築にかかる時間のみを見ると，RVS の時間より短かった．DAG は列挙だけでなくランダムサンプリングや最適化にも使える [10] ため，それらの目的のためには RVS ですべて列挙するよりも DP や DP_PRUNE で構築された DAG を使うほうがよい．

8. おわりに

本稿では simple polygonization を列挙する中畑らの手法に対する枝刈りを提案した．枝刈りを用いることで，凸包内部の点が定数個のとき計算量の上界を指数的に削減できることを理論的に示した．また，山中らの手法についても同様の入力について高速に動作することを裏付ける理論的解析を与えた．中畑らの手法を実装し，枝刈りの性能を評価するとともに，山中らの手法および全探索法と比較した．

中畑らの手法は simple polygonization だけでなく非交差な全域木など他の構造にも適用可能であることが示されている．それらに対する実験的評価が今後の課題として挙げられる．また，中畑らの手法が多項式時間で終了する，あるいは真に指数時間かかる入力を示すことも重要である．山中らの手法については，凸包内部の点が定数個のとき多項式時間で終了することが定理 1 から従う．また，double chain において simple polygonization の個数は（したがって surrounding polygon の個数も）$\Omega(4.64^n)$ [3] となるから，double chain が入力のとき山中らの手法は指数時間かかる．より挑戦的な課題として，simple polygonization に対し指数時間の前処理を用いない真に多項式遅延の列挙アルゴリズムが存在するかは，依然として未解決である．

謝辞 本研究は JSPS 科研費 JP18H04091, JP18K11153, JP19J21000, JP19K11812, JP20H00605, JST CREST JPMJCR18K3 の助成を受けたものです．

参考文献

[1] Erik Demaine. Simple polygonizations. `https://erikdemaine.org/polygonization/`. Accessed: February 22, 2021.
[2] Micha Sharir, Adam Sheffer, and Emo Welzl. Counting plane graphs: Perfect matchings, spanning cycles, and Kasteleyn's technique. *J. Comb. Theory, Ser. A*, 120(4):777–794, 2013.
[3] Alfredo García Olaverri, Marc Noy, and Javier Tejel. Lower bounds on the number of crossing-free subgraphs of K_N. *Comput. Geom.*, 16(4):211–221, 2000.
[4] Dániel Marx and Tillmann Miltzow. Peeling and nibbling the cactus: Subexponential-time algorithms for counting triangulations and related problems. In *Proceedings of the 32nd Symposium on Computational Geometry*, volume 51 of *LIPIcs*, pages 52:1–52:16, 2016.
[5] Chong Zhu, Gopalakrishnan Sundaram, Jack Snoeyink, and Joseph S. B. Mitchell. Generating random polygons with given vertices. *Comput. Geom.*, 6:277–290, 1996.
[6] Christian Sohler. Generating random star-shaped polygons. In *Proceedings of the 11th Canadian Conference on Computational Geometry*, 1999.
[7] Joseph S. B. Mitchell and Joseph O'Rourke. Computational geometry column 42. *Int. J. Comput. Geom. Appl.*, 11(5):573–582, 2001.
[8] Emo Welzl. Counting simple polygonizations of planar point sets. In *Proceedings of the 23rd Canadian Conference on Computational Geometry*, 2011.
[9] Katsuhisa Yamanaka, David Avis, Takashi Horiyama, Yoshio Okamoto, Ryuhei Uehara, and Tanami Yamauchi. Algorithmic enumeration of surrounding polygons. *Discret. Appl. Math.*, 2020.
[10] Yu Nakahata, Takashi Horiyama, Shin-ichi Minato, and Katsuhisa Yamanaka. Compiling Crossing-free Geometric Graphs with Connectivity Constraint for Fast Enumeration, Random Sampling, and Optimization. *arXiv e-prints*, page arXiv:2001.08899, January 2020.
[11] David Avis and Komei Fukuda. Reverse search for enumeration. *Discret. Appl. Math.*, 65(1-3):21–46, 1996.
[12] Manuel Wettstein. Counting and enumerating crossing-free geometric graphs. *J. Comput. Geom.*, 8(1):47–77, 2017.