1D5-OS-02b-4

グラフの部分構造を表すZDD構築のための変数順序決定ヒューリスティクス

Variable Ordering Heuristics to Construct ZDDs Representing Substructure of Graphs

井上 祐馬 *1 Yuma Inoue　鈴木 浩史 *1 Hirofumi Suzuki　伊藤 華 *1 Hana Ito　湊 真一 *1 Shin-ichi Minato

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

Frontier method is a technique to compress and index a huge number of substructures in a given graph, such as spanning forests, into compact data structure named ZDD. ZDD construction requires ordering edges in a given graph, and selection of edge orders dramatically effects the size of ZDD and the efficiency of frontier method. In this paper, we propose edge ordering methods based on graph heuristic search to construct a small ZDD, and conduct computational experiments to evaluate our method.

1. はじめに

1.1 研究背景

グラフは地図やデータ間の関係など，様々な場面で現れるデータ表現である．グラフの基本問題として，特定の条件を満たす部分構造を抽出することが挙げられる．例えば，地図のある地点と別のある地点を結ぶ経路を求めたり，関係グラフから結びつきの強い部分構造を見つけて知識発見に繋げる，などの例がある．グラフに含まれる特定の部分構造の全列挙は，最適化やサンプリング等の幅広いグラフ解析への足がかりとなる．

1.2 研究目的

グラフの様々な部分構造を列挙する枠組みとして，フロンティア法 [5] が注目を浴びている [3, 4, 13]．フロンティア法は解集合を ZDD [7] という圧縮データ構造に保存することで，列挙対象の数に直接依存しない列挙を実現できるメリットがある．フロンティア法をより高速・省メモリにすることは適用範囲の拡大に繋がり，ひいては種々のグラフ解析への応用を実現する．フロンティア法ではグラフの辺に順序をつけて，その順に従って処理を進めるが，本研究ではその順序がフロンティア法の効率に大きく影響することに着目し，良い辺順序を見つけることを目的とする．

1.3 本研究の貢献

本研究ではまず，良い頂点の順序付けを求めるアルゴリズムを提案する．フロンティア法では，フロンティアと呼ばれる頂点集合を更新しながら処理を進めるが，その時々の集合サイズが計算量に影響しているため，集合サイズを評価関数とするビームサーチによって順序付けを行う．次に，頂点の順序付けから良い辺の順序付けを求めるアルゴリズムを提案する．これらを組み合わせることで，既存の順序付けに比べ，フロンティア法が最大で数百倍高速になることを実験によって確認した．

2. グラフ構造列挙

2.1 グラフ

頂点集合 V，辺集合 $E \subseteq \{\{u, v\} \mid u, v \in V\}$ からなるグラフ G を $G = (V, E)$ と表記する．グラフ $H = (V', E')$ がグラフ $G = (V, E)$ の部分グラフであるとは，$V' \subseteq V, E' \subseteq E$ であることを言う．

連絡先: 井上祐馬，北海道大学大学院情報科学研究科，札幌市北区北 14 条西 9 丁目，011-706-6471，yuma@ist.hokudai.ac.jp

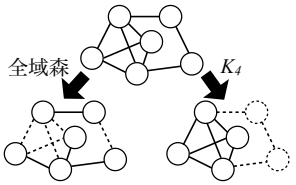

図 1: 無向グラフとその部分構造の例

2.2 グラフの部分構造列挙

グラフの部分構造とは，ある条件を満たす部分グラフを指す．グラフの部分構造の例として，例えば全域森 ($V' = V$ であるような，閉路のないグラフ) やクリーク (完全グラフ = すべての頂点間に辺があるグラフ) などがある (図 1)．頂点数が n の完全グラフを K_n と表記する．

グラフの部分構造列挙とは，与えられたグラフに含まれる条件を満たす部分グラフを全列挙することを言う．例えば K_n が与えられたとき，その部分グラフである $K_m (m \leq n)$ を列挙する問題などである．この例の場合，解は $\binom{n}{m}$ 個あり，指数的な数となる．一般に部分構造の個数はグラフのサイズに対して指数的になりうるため，全列挙することは計算資源上困難な場合が多い．

2.3 ZDD によるグラフ構造の圧縮表現

指数的な列挙対象を圧縮して表現することで，計算資源の節約が期待できる．部分グラフ集合を圧縮表現できるデータ構造の一つに，ZDD [7] がある．(図 2)

ZDD は二分決定木をベースとしており，各節点に入力グラフの各辺に対応するラベルを持つ．ある部分構造 $H = (V', E')$ が辺 e を含むのであれば e をラベルに持つ節点から伸びる 1 枝を辿り，含まないのであれば 0 枝を辿る．終端節点までのパスがそれぞれの部分構造 H に対応しており，終端節点が $\top$ であれば H が条件を満たす部分構造であることを，$\bot$ であれば条件を満たさないことを表す．この二分決定木を 2 つのルール

- 等価節点の共有: 同じラベル・ 子を持つ節点を共有
- 冗長節点の削除: 1 枝が $\bot$ 終端節点を指す節点を削除

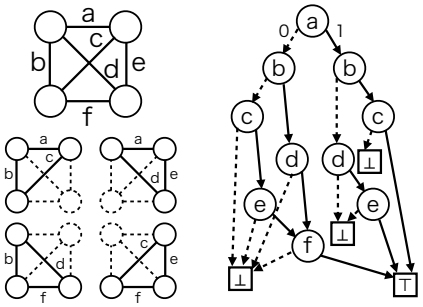

図 2: K_4 とそれに含まれる K_3 を表す ZDD

により圧縮したものが ZDD である．

ZDD のベースとなる二分決定木を構築するためには，グラフの辺の順序を固定する必要がある．辺の順序を変えることでできあがる圧縮後の ZDD の節点数も変化する．ZDD の節点数と辺の順序との関係は，次節にてより詳しく説明する．

3. フロンティア法と辺順序

3.1 フロンティア法

フロンティア法は，グラフの部分構造列挙の結果を表す ZDD を構築するアルゴリズムである [5]．解を直接列挙しないため，列挙対象の数に直接依存しない計算時間を実現できる．

フロンティア法では，ZDD の辺順序に従って辺を使う/使わないで場合分けをしながら処理していき，ベースとなる二分決定木を構築していく．その際，もう条件を満たす部分構造を構成できないような辺の選び方になったとき，途中で枝刈りをし，⊥ 終端節点に繋ぐ．また，今までの辺の選び方が違っていても，以降の辺の選び方がまったく同じになる場合がある．この場合，以降の ZDD 節点は同じものを共有して使うことができる．この共有により ZDD の圧縮を行いつつ，同じ計算を防いで効率化する．

以降の辺の選び方が同じになるかどうか判定するには，フロンティアと呼ばれる頂点集合の状態を見る．フロンティアとは，すでに処理済みの辺と未処理の辺との両方に接続する頂点の集合である．例えば図 3 において，1 から 5 までの辺を処理したとすると，フロンティアとなる頂点集合は $\{b, d, e\}$ である．いま全域森を列挙しているものとする．このとき，図 3 のどちらの状態も b と d は連結であり，e は b, d と非連結であるため，この後の全域森を作るために選べる辺集合は同じになる．よって，同じ状態とみなして以降作られる ZDD を共有する．

3.2 フロンティア法の計算量

フロンティアの状態が同じであるものは 1 度しか計算しないため，フロンティア法の計算量はフロンティアが取りうる状態数によって決まる．例えば前述の全域森の例では，各状態はフロンティア内の頂点の連結成分の状態であり，これは集合分割に対応する．n 個の集合の分割数はベル数 $B_n = \sum_{k=0}^{n-1} \binom{n-1}{k} B_k$ であることが知られているので，フロンティアサイズが F であるときの状態数は B_F で抑えられる．よって，i 番目の辺を処理するときのフロンティアサイズを F_i とすると，全域森を列挙するフロンティア法の計算量は，フロンティアの更新に要する時間を掛け合わせて $\mathrm{O}(\sum_{i=1}^{|V|} F_i B_{F_i})$ と見積もれる．B_n が n に対して指数的であるように，他の部分構造列挙に対するフロンティア法も，フロンティアサイズの指数関数によって計算時間・出力 ZDD サイズが抑えられる [12].

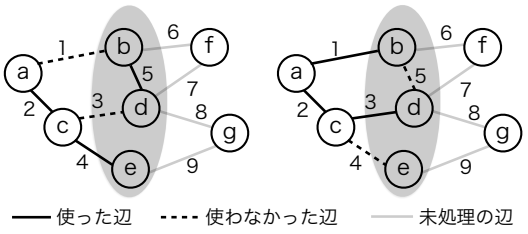

図 3: 全域木を列挙する途中状態のフロンティア

3.3 フロンティアサイズと辺順序

理論的に，フロンティア法の計算量や出力 ZDD サイズがフロンティアサイズの指数関数でバウンドできることがわかった．よって，各処理でのフロンティアサイズをできるだけ小さくすることがフロンティア法の効率化の鍵となる．

ここで，フロンティアの定義は分離点集合の定義と等価である．ある頂点順序 $v_1, \ldots, v_{|V|}$ の i における分離点集合は，$C_i(v_1, \ldots, v_{|V|}) = \{v_k \mid k \leq i, \exists j > i, \{v_k, v_j\} \in E\}$ と定義され，ここから分離点集合数 $\max_{1 \leq i \leq |V|} |C_i(v_1, \ldots, v_{|V|})|$ が定義される．すなわち，辺順序に対応した頂点順序を最適化し，分離点集合数をできるだけ小さくすることは，フロンティアサイズを小さくすることと等価である．一方，分離点集合数が最小となる頂点順序を求める問題は，グラフのパス幅が最小となるパス分解を求める問題と等価であり，一般のグラフについて NP 完全問題であることが知られている [6].

4. 辺順序付けアルゴリズム

4.1 既存手法

分離点集合数を厳密に最適化する手法として，分枝限定法を用いた手法が提案されており，80 頂点程度の地図グラフでは実用的な時間で動作することが示されている [1].

一方，ZDD と似たデータ構造である BDD を用いてネットワーク信頼性を評価する手法では，平面的グラフに関して，平面分離定理からフロンティアサイズが $\mathrm{O}(\sqrt{|V|})$ となる辺順序を求める手法が提案されている [9].

同じく BDD を用いたネットワーク信頼性の評価手法として，BFS や NDS などのグラフ探索を元に，経験的によい近似解を求めるヒューリスティクス手法が用いられている [8]．ここで BFS はある始点から幅優先探索で訪問した順に順序付けを行う手法であり，NDS は現在のフロンティアとの間の辺が最も多い頂点から選ぶ順序付けを行う手法である．また，我々の既存の結果として，処理中のフロンティアから早く抜けることができる頂点を優先して処理する貪欲法を提案しており，高速に比較的よい順序が得られることを実験的に示している [11]．これらの手法はグラフサイズにほぼ線形時間で計算可能なため，大規模なグラフでも計算が可能である．

ここで，既存のフロンティア法の応用先 [3, 4] や経験則から，フロンティア法が主にターゲットとするグラフは数百〜数千頂点程度の中規模グラフである．この観点から，分枝限定法を用いた厳密最適化はスケールしない．一方，既存のヒューリスティクス手法はむしろ高速すぎるため，もう少し時間をかけて高度な計算を行える余地がある．

4.2 提案手法

まず，良い頂点順序を求めるアルゴリズムとしてビームサーチを用いた手法を提案する．次に，その頂点順序から良い辺順序を求めるアルゴリズムを提案する．

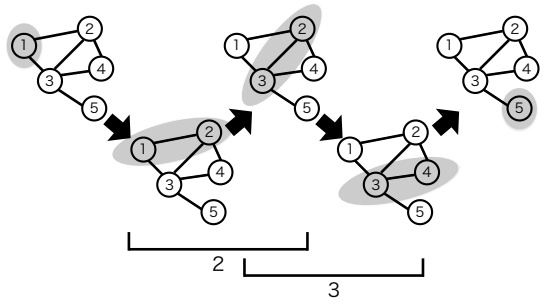

図 4: フロンティアの遷移と頂点に対応する区間

4.2.1 良い頂点順序を求めるアルゴリズム

良い頂点順序を用いる探索アルゴリズムとして，ビームサーチを用いる．ビームサーチとは，探索空間が広すぎて全探索が困難な状況で幅優先的な探索を行う際，評価関数を用意して各状態を評価し，評価値が高いトップ K の状態のみを展開することで枝刈りを行う探索アルゴリズムである．このとき K はビーム幅と呼ばれ，実行の際に指定するパラメータとなる．

今回評価関数は，それまでの頂点順序におけるフロンティアサイズの二乗和とした．これは，最大値を抑制しつつ，全体のフロンティアサイズも抑えるためである．

提案アルゴリズムの i ステップ目は以下のようになる：$i-1$ 頂点の順序付け K 個それぞれについて，現在のフロンティアに隣接した頂点から 1 つ選んで末尾に加える．1 つあたり最大で $|V|-i+1$ 通りの選択肢があるので、最大で $K(|V|-i+1)$ 通りの i 頂点の順序付けが得られる．各々の順序付けについて，フロンティアサイズの二乗和を計算し，評価値の高いもの K 個だけを次の候補とし，$i+1$ ステップ目に遷移する．

出力は $|V|$ ステップ終了後に最高評価値の頂点順序とした．

4.2.2 頂点順序から良い辺順序を求めるアルゴリズム

頂点順序が決まると，フロンティアの遷移が一意に定まる．ここで各頂点に着目すると，フロンティアに入ってから出て行く，という区間が各頂点につき必ず 1 つだけ存在する．(図 4) これは分離点集合とパス分解との等価性から言える．

よって，頂点順序に応じた辺順序を決める際，辺 $e=\{u,v\}$ を用いるタイミングは，頂点 u の区間と頂点 v の区間が重なる区間内であればいつでもよい．このような区間の中でフロンティアサイズが最小になるタイミングで e を用いれば，対応する状態数の上界が小さくなるため，効率がよいと考えられる．

5. 実験

5.1 実験設定

比較手法として，分枝限定法 (表中では B&B)，BFS，NDS，貪欲法 (表中では Greed) を用いた．このうち，分枝限定法は Sage [10] に実装されたものを，BFS，NDS，貪欲法，提案手法は著者らが C++言語で実装したものを用いた．提案手法 (表中では Beam) は，ビーム幅を 1000 で固定とした．

グラフデータには，ZDD が扱う対象に選ばれやすい地図グラフとして東京・愛知・大阪・福岡の市町村の隣接関係から得たグラフと，分枝限定法のベンチマークとして使われた Rome Graph のうち，100 頂点のグラフを用いた．地図グラフは平面的グラフ (に近い構造) であるため，分離点集合数と等価な概念であるパス幅が小さくなる (=O($\sqrt{|V|}$)) という特徴がある．

フロンティア法には Graphillion パッケージ [2] を利用し，列挙する部分構造は全域森と K_4 とした．

表 1: フロンティアサイズの最大と平均

		BFS	NDS	Greed	B&B	Beam
東京	最大	9	18	9	**7**	9
	平均	6.03	10.94	5.63	**5.03**	5.74
愛知	最大	13	10	**8**	**8**	**8**
	平均	8.03	6.24	5.49	5.76	**5.18**
大阪	最大	9	10	9	**7**	**7**
	平均	5.47	5.39	5.32	4.86	**4.68**
福岡	最大	11	14	8	**6**	**6**
	平均	6.01	7.84	4.47	4.41	**4.09**

表 2: 辺順序の計算時間 (ms)

	$\|V\|$	$\|E\|$	BFS	NDS	Greed	B&B	Beam
東京	56	135	0.1	0.3	0.7	7.0	122.7
愛知	62	157	0.3	0.2	0.3	9.7	136.6
大阪	69	161	0.5	0.2	0.2	14.9	119.9
福岡	56	123	0.1	0.2	0.1	3.8	102.5

計算機環境として，Intel® Core™ i7-3930K CPU 3.20GHz 6 Core，Ubuntu 14.04 LTS 64bit OS，64GB メモリの計算機を並列計算なしで用いた．また，実行時間に 10 分の制限時間をつけ，計測を行った．

5.2 実験結果と評価

5.2.1 都道府県グラフ

都道府県のグラフについて，各手法による辺順序のフロンティアサイズを表 1 に，順序付けに要した時間を表 2 に示す．

フロンティアサイズの最大値はどのグラフにおいても分枝限定法が最小であり，理論通りである．一方，ビームサーチもほとんどのケースで最小であり，評価関数のおかげで平均も分枝限定法より抑えられている傾向が見られる．NDS はどのケースにおいても最大値・平均値ともに大きく，BFS は貪欲法に近いこともあるが大きくなるケースも見られ，安定しない．

計算時間については、線形時間で計算できる BFS，NDS，貪欲法はほとんど無視できる程度だった．分枝限定法も文献 [1] にある通り，高々70 頂点程度の都道府県グラフでは十分高速に動作した．一方，ビームサーチは若干計算時間を要しているが，ビーム幅などのパラメーターにより調節も可能である．

フロンティア法の実行結果を表 3 に示す．各グラフに対して，最も効率的だった結果を太字にしてある．最も効率的だった結果とフロンティアサイズの最大・平均の最小が一致しており，理論計算量の上界を考慮することが実際の実行速度にもよい影響を与えていることがわかる．一方で，東京の全域森列挙では，BFS よりも平均フロンティアサイズがよいビームサーチの方が 5 倍近く遅くなっており，理論的な見積もりだけではない実際の枝刈り・共有の影響が出ているものと考えられる．

表 3 の K_4 列挙に関して，ZDD サイズが小さくどれも同じ大きさなのは，含まれる K_4 の数が少ないためである．実際，東京・愛知・大阪・福岡に含まれる K_4 の数はそれぞれ 0，5，0，6 であった．このように ZDD が最終的に小さくなるケースであっても，辺順序を考慮することでフロンティア法の計算時間が大きく変わることも見て取れる．

5.2.2 Rome Graph

貪欲法とビームサーチによる頂点順序付けを用いて Rome Graph の全域森を列挙した結果を表 4 に示す．分枝限定法は文献 [1] にある通り，辺順序付けの段階で 10 分の制限時間を超過した．BFS と NDS は辺順序付け後のフロンティア法の実行時間が 10 分を超過した．

表 3: 都道府県グラフに対するフロンティア法の実行時間と ZDD サイズ

		全域森					K_4				
		BFS	NDS	Greed	B&B	Beam	BFS	NDS	Greed	B&B	Beam
東京	時間 (sec)	0.0997	86.6441	0.1514	**0.0266**	0.4692	0.0153	2.5380	0.0174	**0.0050**	0.0380
	ZDD サイズ	79154	35690582	87581	**15188**	198354	1	1	1	1	1
愛知	時間 (sec)	29.7658	0.3704	0.2117	0.2574	**0.1041**	2.1823	0.0280	0.0209	0.0201	**0.0149**
	ZDD サイズ	9744239	125083	116023	98637	**58581**	30	30	30	30	30
大阪	時間 (sec)	0.0752	0.1371	0.3249	0.0372	**0.0290**	0.0127	0.0211	0.0311	0.0086	**0.0073**
	ZDD サイズ	61897	118025	204142	23339	**18098**	1	1	1	1	1
福岡	時間 (sec)	0.4104	116.5392	0.0256	0.0122	**0.0095**	0.0227	1.2015	0.0048	0.0038	**0.0036**
	ZDD サイズ	176699	1151688	19747	4394	**3642**	36	36	36	36	36

表 4: Rome Graph での全域森列挙の実行時間と ZDD サイズ

			Greed					Beam				
	$\|V\|$	$\|E\|$	前処理	最大	平均	実行時間	ZDD	前処理	最大	平均	実行時間	ZDD
grafo7785	100	136	0.2 ms	12	8.10	15.6 s	4429458	345.3 ms	10	6.72	2.1 s	1319436
grafo10183	100	132	0.5 ms	13	8.51	62.6 s	19582483	308.3 ms	12	7.23	6.2 s	5940898
grafo10586	100	137	0.2 ms	15	9.54	timeout	—	291.0 ms	12	7.46	8.1 s	6590789
grafo10861	100	138	0.2 ms	14	9.48	timeout	—	326.2 ms	11	7.54	5.8 s	5953148
grafo11194	100	127	0.5 ms	14	9.11	235.7 s	57684824	315.7 ms	10	6.90	1.2 s	833893

どのグラフにおいても，フロンティア法はビームサーチによる辺順序を用いた方が速く，貪欲法よりも約 10〜100 倍程度の高速化を実現した．得られた ZDD も最大で 70 倍ほど小さくなった．一方，辺順序付けそのものに要した時間は貪欲法に比べるとかなりかかっているが，その後のフロンティア法に要した時間の改善に比べ，十分無視できる時間といえる．

6. おわりに

6.1 結論

本研究では，グラフの部分構造を列挙するアルゴリズムであるフロンティア法をより効率的にすることを目的に，辺の順序付けに着目し，グラフの構造から良い辺順序を与えるアルゴリズムを提案した．提案アルゴリズムはフロンティア法の計算量に大きく影響するフロンティアサイズに着目し，フロンティアサイズを評価関数とするビームサーチによる順序付けを行った．実験の結果，目標とする中規模の地図グラフに対して最大で数百倍の効率改善を達成した．加えて，出力 ZDD サイズを小さくできたことは，後の応用の効率改善にも繋がる．

6.2 今後の課題

フロンティアサイズを抑えることを意識した辺順序を用いることで，実際にフロンティア法が効率的になることが実証できた一方で，東京の BFS の例のように，フロンティアサイズが少し大きくても高速に動作する例も確認された．出力 ZDD が理論的評価よりも小さくなる要因としては，枝刈りの効きやすさや，同一状態でない場合でも最終的に ZDD のルールで圧縮される場合などが考えられる．様々なグラフに対してさらに実験を行うことで，このような要因に関しても良いヒューリスティクスを発見し，導入したい．また，可能であれば，そのようなヒューリスティクスに関しても理論的に評価を行いたい．

謝辞

本研究は JSPS 科研費 15J01665 および 15H05711 の助成を受けたものです．

参考文献

[1] David Coudert, Dorian Mazauric, and Nicolas Nisse. Experimental evaluation of a branch and bound algorithm for computing pathwidth. In *Experimental Algorithms*, pp. 46–58. Springer, 2014.

[2] Takeru Inoue, Hiroaki Iwashita, Jun Kawahara, and Shin-ichi Minato. Graphillion: software library for very large sets of labeled graphs. *International Journal on Software Tools for Technology Transfer*, Vol. 18, No. 1, pp. 57–66, 2014.

[3] Takeru Inoue, Kyoya Takano, Toshio Watanabe, Jun Kawahara, Ryo Yoshinaka, Akihiro Kishimoto, Kazuhiko Tsuda, Shin-ichi Minato, and Yasuhiro Hayashi. Distribution loss minimization with guaranteed error bound. *IEEE Transactions on Smart Grid*, Vol. 5, No. 1, pp. 102–111, 2014.

[4] Hiroaki Iwashita, Yoshio Nakazawa, Jun Kawahara, Takeaki Uno, and Shin-ichi Minato. Efficient computation of the number of paths in a grid graph with minimal perfect hash functions. *TCS technical report TCS-TR-A-10-64, Division of Computer Science, Hokkaido University*, 2013.

[5] Jun Kawahara, Takeru Inoue, Hiroaki Iwashita, and Shin-ichi Minato. Frontier-based search for enumerating all constrained subgraphs with compressed representation. *TCS technical report TCS-TR-A-14-76, Division of Computer Science, Hokkaido University*, 2014.

[6] Nancy G Kinnersley. The vertex separation number of a graph equals its path-width. *Information Processing Letters*, Vol. 42, No. 6, pp. 345–350, 1992.

[7] Shin-ichi Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. *ACM/IEEE Design Automation Conference*, pp. 272–277, 1993.

[8] Yuchang Mo, Liudong Xing, Farong Zhong, Zhusheng Pan, and Zhongyu Chen. Choosing a heuristic and root node for edge ordering in BDD-based network reliability analysis. *Reliability Engineering & System Safety*, Vol. 131, pp. 83–93, 2014.

[9] Kyoko Sekine. Counting the number of paths in a graph via BDDs. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, Vol. 80, No. 4, pp. 682–688, 1997.

[10] The Sage Developers. *Sage Mathematics Software (Version 7.0)*, 2016. `http://www.sagemath.org`.

[11] 伊藤華, 井上祐馬, 湊真一. ZDD のトップダウン構築における変数順序付け法の実験と考察. 情報科学技術フォーラム講演論文集, Vol. 14, No. 1, pp. 115–116, 2015.

[12] 高野圭司. フロンティア法から生成される ZDD の幅解析. 数理解析研究所講究録, Vol. 1849, pp. 77–82, 2013.

[13] 湊真一（編）. 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, 4 月 2015.