4E1-1

オイラー路の高速な列挙索引化アルゴリズム

A Fast Algorithm for Enumerating and Indexing Eulerian Paths

ムハマド　ホリルロハマン *1　　湊真一 *1*2
Muhammad Kholilurrohman　　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

*2湊離散構造処理系プロジェクト
ERATO Minato Project, Japan Science and Technology Agency

Although a mathematical formula for counting the number of Eulerian paths (cycles) of a directed graph is already known, no formula is yet known for enumerating such paths if the graph is an undirected one. A simple approach is to use a backtrack based algorithm for the latter kind of the graph, but it is inefficient since the time needed is proportional to the number of the paths. In this paper, a fast algorithm based on *Breadth-first search* (BFS) to enumerate all Eulerian paths of an undirected graph is proposed.

1. はじめに

オイラー路とは，与えられたグラフの全ての辺を 1 度だけ通る路のことである．出発点と到着点が一緒である場合はオイラー閉路という．与えられたグラフにオイラー路が存在する必要十分条件は，次数が奇数である頂点がちょうど 2 つあることである．一方，オイラー閉路が存在する必要十分条件は，グラフのすべての頂点の次数が偶数であること．特に日本では，オイラー路問題は一筆書き問題としてよく知られている．

有向グラフに関して，そのグラフに存在する全てのオイラー路の数は *BEST* 定理 [1, 2] を使って数式で表すことができる．一方，無向グラフに関しては，オイラー路の数を求める数式はまだ知られていない．バックトラック法を使えば，無向グラフ上のオイラー路の数を求めることができるが，解の個数に比例する時間がかかるため，少し大きなグラフになると非現実的な計算時間を必要とする．

本研究では，与えられた無向グラフのすべてのオイラー路 (又は閉路) を高速に列挙索引化するアルゴリズムを提案する．このアルゴリズムは，Knuth の Simpath 法 [3] を拡張したもので，有向非巡回グラフ (*Directed Acyclic Graph*: DAG) を用いて各辺の接続関係を圧縮して表現することにより，動的計画法を用いた高速な計算を実現した．いくつかの例題で実験した結果，本手法により，1 京通りをはるかに越えるような膨大な個数のオイラー路を，市販の PC でわずか数秒で正確に数え上げることができた．

2. 準備

本手法では，オイラー路を列挙するために与えられたグラフの辺を一つずつ処理していく．その度に処理済みの辺と未処理の辺が接続している頂点の集合ができ，それをフロンティア (*frontier*; 境界) と呼ぶ．例えば，図 1 のグラフにおいて，処理済みの辺が太線で未処理の辺が点線だとすると，現在のフロンティアが $\{v_4, v_5, v_6\}$ になる．

入力したグラフの頂点 v に繋がっている辺の数，つまり，頂点 v の次数を $deg(v)$ と書く．この頂点に関する *Edge pairing*(エッジペアーリング) というのは，繋がっている $deg(v)$ 本の辺を一度分解し，各辺がちょうど 1 つの他の辺に繋がるようにすることを意味する．例えば，頂点 v に繋がっている辺が e_1, e_2, e_3, e_4 だとすると，辺を分解したグラフは図 2 の右側のように示せる．そして，頂点 v に関する全ての Edge pairing のパターンは図 3 のようになる．

連絡先: 湊真一，北海道大学大学院情報科学研究科 (教授)，〒 060-0814 札幌市北区北 14 条西 9 丁目，011-706-7682, minato@ist.hokudai.ac.jp

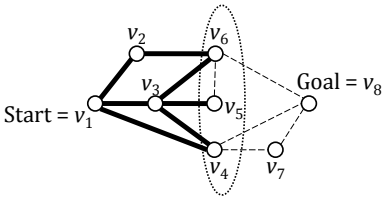

図 1: フロンティア

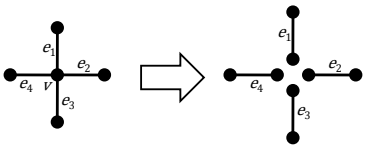

図 2: 頂点 v に繋がる辺を分解する

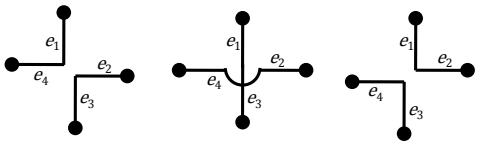

図 3: 頂点 v に関する全ての Edge pairing のパターン

3. 基本アイデア

与えられたグラフに対し，そのグラフの各頂点に関する全ての Edge pairing のパターンに基づいて場合分けをしながら幅優先で多分木を構築すれば，その多分木に全てのオイラー路が含まれることになる．

しかし，そのようにオイラー路を列挙すれば，多くのメモリ量や計算時間が必要となる．我々の目的は，動的計画法かつ幅優先でオイラー路になりえない部分パスの枝刈りや多分木の等価な節点，つまり同じ部分グラフを持った節点を共有して多分木ではなく，与えられたグラフに存在する全てのオイラー

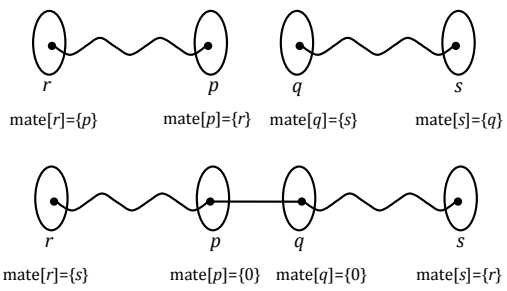

図 4: 選択 1

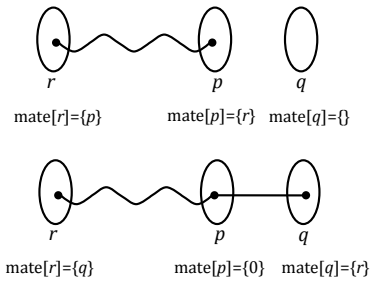

図 5: 選択 2

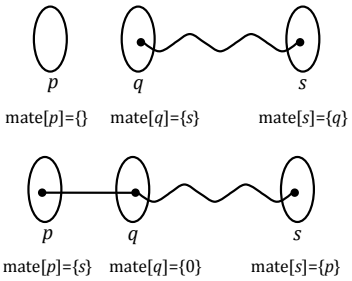

図 6: 選択 3

路を格納した多分決定グラフを作ることである．本手法では，多分決定グラフの根節点から出発した各経路がグラフの一つの異なったオイラー路に対応しているので，共有した節点の分だけ多分決定グラフの圧縮率が上がる．

4. 提案手法

オイラー路を構築するには辺を一つずつ加えていく．そうするといくつかの部分パス (断片) ができる．そして，一つの頂点に複数の部分パスが繋げられるので，その頂点上の接続情報を $mate$ (友達) だという概念を使って表す．この論文では頂点 v の $mate$ を重み付き集合として表し，$mate[v] = \{\alpha_1, \alpha_2, \cdots, \alpha_g\}$，各要素 $\alpha_i (1 \leq i \leq g)$ に重み $mate[v].w(\alpha_i)$ がつくと書く．

- $\alpha_i = 0$ は部分パスの途中を表す．
- $\alpha_i = u (u \neq 0)$ は，頂点 v が頂点 u にある部分パスに繋げられていることを意味する．

そして，重み付き集合のサイズ，つまり $\sum mate[v].w(\alpha_i)$ は今頂点 v に繋がっている部分パスの数に対応している．オイラー路には各頂点 v がちょうど $\frac{deg(v)}{2}$ 回現れるので，$mate$ の最終サイズも $\frac{deg(v)}{2}$ にしたい．

頂点 p だけから見ると，辺 p–q を加えたときに 2 つの選択がある．それは，p にあった部分パスに繋げる，又は p のどの部分パスにも繋げずに加える，の二つである．しかし，後者の選択は頂点 p に繋がっている部分パスの数が $\frac{deg(p)}{2}$ 以下のときにのみできる．頂点 q に関しても同じように 2 つの選択があるので，辺 p–q を加えたときに，実際には 4 つの選択がある．

1. 頂点 p にある一つの部分パスを頂点 q にある一つの部分パスに繋げる．
2. 頂点 p にある一つの部分パスを頂点 q に繋げる．
3. 頂点 p を頂点 q にある一つの部分パスに繋げる．
4. 頂点 p を頂点 q に繋げる．

オイラー路の構築で辺を加える度に，いくつかの頂点の $mate$ を更新する必要がある．その更新ルールは上記の選択によって違ってくる．例えば，選択 1 は図 4 に示されているが，頂点 p と頂点 r を繋げた部分パスを頂点 q と頂点 s を繋げた部分パスに辺 p–q を使って繋げることを表している．そのときに，頂点 r において，$mate[r].w(p)$ を一つ減らして，$mate[r].w(s)$ を一つ増やすことで $mate[r]$ の更新ができる．頂点 p, q, s も同様である．

次に，図 5 は選択 2 を表しているが，更新すべき $mate$ は頂点 r，頂点 p，と頂点 q の $mate$ となる．ここでは $mate[q]$ の更新ルールは単に $mate[q].w(r)$ を一つ増やすことだけである．さらに，図 6 は選択 3 を表している図であり，図 7 は選択 4 を表している図である．

多分決定グラフを構築するときの節点の共有を行うには，フロンティア上の頂点の $mate$ を比較すればよい．その理由はフロンティア上の頂点以外の接続情報は必ず同じだからである．二つの節点が同じ $mate$ 配列を持っていれば，その節点を共有できる．

本研究では頂点の $mate$ を表すために重み付き集合を使っているが，その理由は三つ挙げられる．

1. 多分決定グラフの節点の共有率を上げるため．
2. 同じことを一括まとめて処理することで計算時間を短くするため．
3. サイクルを持った部分グラフ (未完成のオイラー路) を削除するため．

まず，共有率を上げる理由については，例えば頂点 v の $mate[v]$ を整数の配列として表すこととする．さらに，多分決定グラフから 2 つの節点があり，一つ目の節点の $mate[v] = \{1, 0, 3\}$ とし，もう一つの節点の $mate[v] = \{0, 3, 1\}$ とする．そうすると，二つの節点が異なる $mate$ を持っているので，二つの節点を共有することができない．一方，$mate$ を重み付き集合として表すと，両方ともの $mate$ が $\{0, 1, 3\}$ になり，共有できる可能性がある．

2 番目の理由については，例えば $mate$ をマルチセット (multiset) として表すこととする．さらに，$mate[p] = \{3, 3\}$ であり，$mate[q] = \{7, 7, 7\}$ であるとする．そのときに，辺 p–q を加えれば，$2 \times 3 = 6$ の繋ぎ方があり，それを一つずつ考慮しなければいけない．しかし，その 6 通りの繋ぎ方を持った子節点たちは結局同じ $mate$ 配列を持つので，共有されることになる．一方，$mate$ を重み付き集合として表すと，一つだけの繋ぎ方になり，節点から子節点に $2 \times 3 = 6$ 本の辺に繋げるようにすればよい．

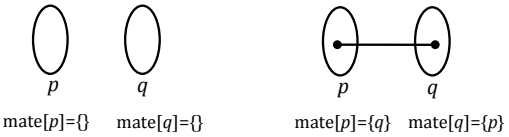

図 7: 選択 4

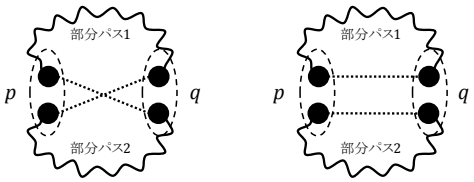

図 8: 例外

しかし，これには例外がある．それは，$mate[p] = \{q\}$ で，$mate[q] = \{p\}$ のときに辺 p–q を加えれば，サイクルが作られてしまう場合がある．例えば，$mate[p].w(q)$ が 2 で，$mate[q].w(p)$ も 2 であることは図 8 に示されている．左側の図は異なる部分パスを辺 p–q (破線) に繋いだ二つの繋ぎ方であり，右側の図は同じ部分パスを繋いだ二つの繋ぎ方である．そこで，右側の図に示される繋ぎ方にはサイクルができてしまうので，オイラー路は作れない．つまり，多分決定グラフに対応する節点と子節点は $mate[p].w(q) \times mate[q].w(p)$ ではなく，$mate[p].w(q) \times mate[q].w(p) - mate[p].w(q)$ 本の辺に繋げることになる．これは $mate$ を重み付き集合として表す 3 番目の理由になる．

例として，提案手法を使って図 12 の左端の「中」型グラフに存在する全てのオイラー路を列挙する手順を図 9 に示す．そして，実際の処理中の $mate$ は図 10 に示されている．

多分決定グラフが完成すれば，オイラー路の数を数える方法は次の通りになる．まず，多分決定グラフの一番下の節点の値は 1 だと定義する．次に，他の節点の値は，その節点の子節点の値の総和である．最後に，オイラー路の数は多分決定グラフの根節点 (一番上の節点) の値となる．例えば，図 10 の各節点の右上に書かれている数字はその節点の値である．

提案手法によって構築した多分決定グラフは n 番目のオイラー路を得るための索引とみなすことができる．まず多分決定グラフの根節点から，n と各節点の値を比較しながらその多分決定グラフを辿っていく．そして，多分決定グラフ上の n 番目のオイラー路に対応するパスが分かれば，それを利用して実際にオイラー路を構築すればよい．この方法で，多分決定グラフに格納されている n 番目のオイラー路は $O(|E|)$ で見つけることができる．

5. 実験結果

本研究では 4GB メモリ Intel® Core™i3-2330M CPU @ 2.20GHz × 4 の計算機を用いていくつかのグラフを使って実験を行った．ここでは多分決定グラフの節点数，提案手法の実行時間 (秒単位)，バックトラック法による実行時間 (秒単位)，そして各グラフに存在するオイラー路 (又は閉路) の数を示す．

5.1 完全グラフ (Complete Graph $C(n)$)

任意の 2 頂点間に辺があるグラフは完全グラフという．ここで $C(n)$ は n 本の辺を持った完全グラフである．完全グラフに関する実験は以下の表 1 に示されている．

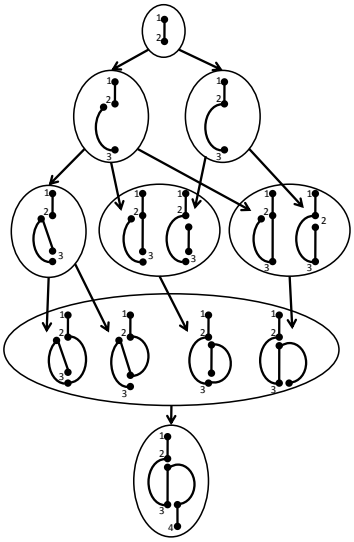

図 9: 「中」型グラフを処理する手順

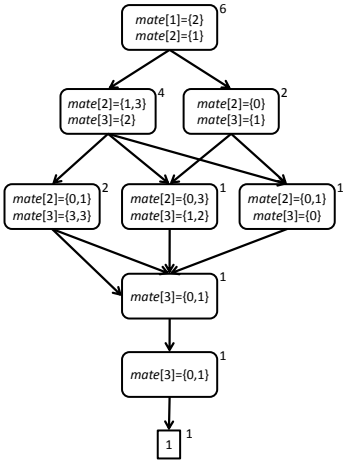

図 10: 「中」型グラフを処理する手順の $mate$

完全グラフに関して，グラフをどの辺の順序で処理しても，フロンティアのサイズが大きいので，オイラー路の計算は難しい．それでも，$n = 9$ の 3,646,080,228,084,940,800 のオイラー閉路の数は約 33 秒で高速に計算できた．

5.2 Aztec Diamond グラフ $A(n)$

$n = 1, 2$ と $n = 3$ に関する Aztec Diamond グラフ $A(n)$ は図 11 に示されているが，頂点 1 を始点とする全てのオイラー閉路の数を求める実験を行った．得られた実験結果は表 2 に示されている．

5.3 環状グラフ (Ring Graph $R(n)$)

環状グラフ $R(n)$ は図 12 に示されているように n 個の環を持ったグラフである．環状グラフには多重辺を含むので，多重グラフに属している．環状グラフにおける実験結果は表 3 に示している．

表 1: 完全グラフ $C(n)$ における実験結果

n	節点数	実行時間		オイラー閉路の数
		提案手法	バックトラック法	
3	7	0.000442	0.000318	2
5	50	0.020941	0.042312	528
7	2168	0.100832	time out	389928960
9	451162	33.361454	time out	3646080228084940800

表 2: Aztec Diamond グラフ $A(n)$ における実験結果

n	節点数	実行時間		オイラー閉路の数
		提案手法	バックトラック法	
1	8	0.000228	0.000368	2
2	40	0.015503	0.023900	80
3	286	0.032317	156.887235	264320
4	2164	0.071024	time out	67131225600
5	17271	0.500763	time out	1282298454848135168
6	148224	6.153548	time out	18239588354744044219224391680
7	1382302	73.527219	time out	192178269775153104174170778660103782400
8	14083862	942.330957	time out	1495157006436041186484738405257449073460914460033024

表 3: 環状グラフ $R(n)$ における実験結果

n	節点数	実行時間		オイラー閉路の数
		提案手法	バックトラック法	
1	9	0.002855	0.000860	6
5	33	0.007305	2.079651	7776
10	63	0.014786	time out	60466176
50	303	0.031448	time out	8082812774647640606431396004565362933 76
100	603	0.045438	time out	6.533186×10^{77}
500	3003	0.450163	time out	1.190214×10^{389}
1000	6003	1.603632	time out	1.416610×10^{778}
5000	30003	37.794911	time out	$5.704951 \times 10^{3890}$
10000	60003	150.083619	time out	$3.254647 \times 10^{7781}$

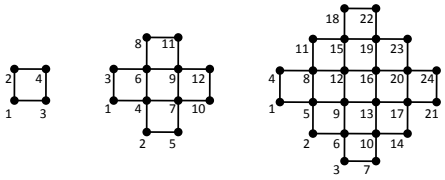

図 11: $n = 1, 2, 3$ の Aztec Diamond グラフ $A(n)$

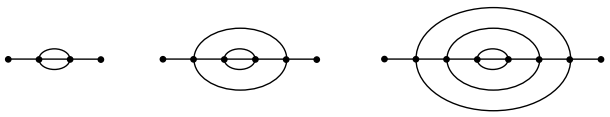

図 12: $n = 1, 2, 3$ の環状グラフ $R(n)$

6. おわりに

以上で，無向グラフに存在する全てのオイラー路 (又は閉路) を列挙索引するアルゴリズムを説明した．この問題は#P-complete であることが知られているので [4]，最悪の場合は，本手法でも指数的な時間がかかる．しかし，本論文で示したように，DAG がメモリに納まる範囲で小さく圧縮されるような例題では，短時間で計算できることが示された．場合によっては膨大な個数のオイラー路でも，小さな DAG で格納できる場合では，比較的短時間で計算できることがある．今後の課題として，本手法でオイラー路をうまく列挙できるグラフのクラスを明かにしたい．

参考文献

[1] van Aardenne-Ehrenfest, T., de Bruijn, N. G. Circuits and trees in oriented linear graphs. Simon Stevin 28, 203–217 (1951)

[2] Tutte, W. T., Smith, C. A. B.: On unicursal paths in a network of degree 4. American Mathematical Monthly 48, 233–237 (1941)

[3] D. E. Knuth : “The Art of Computer Programming,” volume 4, Binary Decision Diagrams, ZDDs to represent simple paths, pp.122, Addison-Wesley, (2008)

[4] Brightwell, G. R., Winkler, Peter : “Note on Counting Eulerian Circuits,” CDAM Research Report LSE-CDAM-2004-12 (2004)