数理解析研究所講究録
第1554巻 2007年 145-152

3正則グラフの巡回セールスマン問題に対する厳密アルゴリズムの改善

岩間 一雄 (Kazuo Iwama), 中島 拓也 (Takuya Nakashima)
京都大学 情報学研究科 (School of Informatics, Kyoto University)
{iwama, tnakashima}@kuis.kyoto-u.ac.jp

概要

3正則グラフ (cubic graph) に対する巡回セールスマン問題はNP完全問題であるが, n 頂点3正則グラフに対する厳密アルゴリズムはEppsteinによる $O(2^{n/3})$ 時間のアルゴリズムが知られている. 本論文ではこれの最悪のケースの割合を減らすことで $O(2^{31n/96})$ 時間で解けるようにアルゴリズムを改善した.

1 まえがき

巡回セールスマン問題 (TSP) は最も有名なNP完全問題の一つである [2].

一般のグラフに対するTSPのアルゴリズムとしては1962年に発表された動的計画法を用いて頂点数 n のグラフについて $O(n^2 2^n)$ 時間で解くアルゴリズムが知られているが [1], それ以降改善されたアルゴリズムは見つかっていない [7].

また, 一般のグラフではなく特殊なグラフに対するTSPに関してもいくつか研究が行なわれている. たとえば平面グラフにおけるTSPは $O(n^{3/2}2^{6.903\sqrt{n}})$ 時間で解く事ができる [9].

与えられるグラフの最大次数を3に制限したTSPはその一つで, この問題も一般のTSPと同じくNP完全問題である [2]. この問題に対してはEppstein[8] が $O(2^{n/3}) \approx 1.260^n$ 時間で解けることを示している. 本論文ではこれを改良することによって $O(2^{31n/96}) \approx 1.251^n$ で解ける事を示す.

[8] ではバックトラック法によってハミルトン閉路の探索を行なっているが, 一回の再帰呼び出しにおいて多数の枝についてハミルトン閉路として使用するかどうかを決定する事で高速に問題を解いている.

本研究ではその際の探索途中の頂点の状態と最悪のケースについて調べ, 探索においていくつかの制限がかけられることを発見した. これに基づき, ハミルトン閉路として使用するかどうか決める枝を選ぶ際に優先順位をつけ, 探索において最悪のケースが少なくなるようにアルゴリズムの改善を行った.

その結果, 探索全体にかかる計算時間を改善する事ができた.

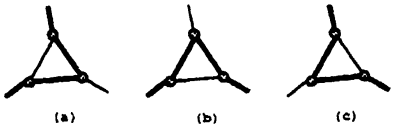

図 1: Treatment of 3-cycles.

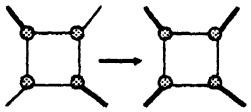

図 2: Treatment of 4-cycles.

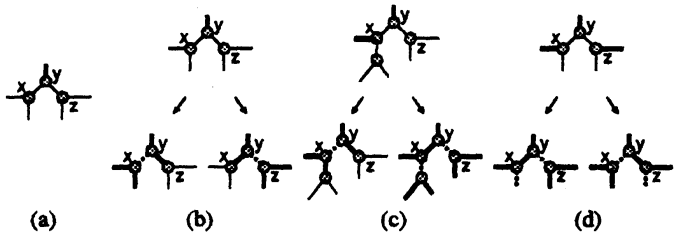

図 3: Single branching of the algorithm

2 Eppstein のアルゴリズム

まず, 与えられた n 頂点 3 正則グラフ G に対し, 全ての 3-cycle をひとつの頂点に縮約する. ハミルトン閉路は 3-cycle においては Fig. 1 (a), (b), または (c) のようになるので次数 3 のひとつの頂点に縮約できる.

また, 4-cycles を特殊に取り扱う: Fig. 2 が示すように, 対角の枝が選択されていればその他の対角の枝も自動的に選択する. これにより残された 4-cycle は最終ステップでまとめて処理される (詳細は [8] 参照).

アルゴリズムのメイン部分は Fig. 3 のように表せる. 既に解の一部として選択されている枝 (Fig. 3 (a) では選択されている枝は太線で示されている) に隣接している頂点 y を選び, 枝 yx または yz を解に追加する. ここで, 頂点 x および z の状態によってケース (b), (c) および (d) に分かれる.

全ての隣接枝が選択されていない頂点を "フリー" な頂点であると表記する. 頂点 x や z がフリーであると仮定すると枝 yx または yz が解として追加され, 最低でも 3 本および 3 本の枝が解に追加される. (c) のように, それらのうちひとつ, (ここでは x) がフリーでないとすると同様に 2 本及び 5 本の枝が解に追加される. (d) のようにどちらもフリーでない場合, 最低でも 4 本および 4 本が追加される. しかし, 最後のケースは Fig. 4 のように 6-cycle の一部であった場合のみは 3 本しか追加されない.

ハミルトン閉路は n 本の枝から構成されるので, 本アルゴリズムの計算時間 $T(n)$) は以下のように表せる.

$$T(n) \leq 2T(n-3) \tag{1}$$

$$T(n) \leq T(n-2) + T(n-5) \tag{2}$$

$$T(n) \leq 2T(n-4) \tag{3}$$

ここでは (1) が最悪のケースなので $T(n) = 2^{n/3}$ となる. これが [8] のメインとなる結果である.

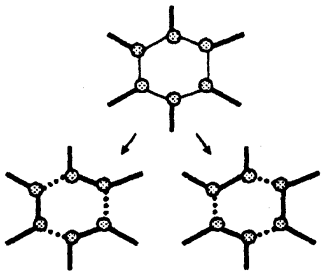

図 4: 6-cycles

3 新しいアルゴリズム

再起呼び出しにおいて, 最悪のケースは Fig. 3 (b) のケースである. しかし, このケースは 1 回の呼び出しで フリー な頂点を 4 つ減少させるので, バックトラックツリーにおける根から葉への各パスの中では高々$n/4$ 回しか起こらない.

先行研究においてはこのように最悪のケースに制限が加わることを考慮していないが, 本研究においては最悪のケースによる分岐の回数を減らすことで最悪計算時間を改善している.

残念ながら, 最悪のケースは Fig. 3 (b) 以外に Fig. 4 のように 6-cycle の周囲の枝全てが選択されており, かつ 6-cycle の枝すべてが選択されていない状態というものが存在し, このほかにもアイデアは単体では動作しない.

しかし, 最悪のケースは上記 2 ケースのみであるため, この合計回数を減らすことが出来れば最悪計算時間は改善される. 6-cycle が最悪のケースとなるのは, 6-cycle の内部の枝が 6-cycle の周囲の枝全てが選択されるまで選択されない場合のみである. したがって, 以下のように新しく枝を解に追加する箇所に関して優先順位をつけることにより 6-cycle が最悪のケースとなる回数を減らすことができる.

Fig. 9 に [8] のオリジナルのアルゴリズムを示す. F は既に解として選択された枝の集合である. われわれのアルゴリズムは, これの Step.3 (b) を以下の (b1) および (b2) に置き換えたものである:

(b1) そのような 4-cycle が存在せず, $G\backslash F$ が生きた 6-cycle を持ち, かつその 6-cycle が cycle 上の頂点 y で選択された cycle 外の枝と接している場合, cycle 上の y の隣接枝を z とする. そのような 6-cycles が複数存在した場合, 隣接枝のうち選択されているものの数が最も多いものを選択する.

(b2) そのような 4-cycle も 6-cycle も存在せず, かつ F が空でない場合, F から枝 uy を y が最低でもひとつのフリーでない頂点 z に隣接しているように取る (yz は $G\backslash F$ 上の枝となる). そのような枝 uy が存在しない場合, uy は F の任意の枝とし, yz は $G\backslash F$ 上の任意の隣接枝とする.

4 アルゴリズムの解析

A 6-cycle が "生きている" とは, cycle の 6 本の枝全てが選択されていないことを示す. 6-cycle が生きていないとき, "死んでいる" と表記する. $C(i), 0 \leq i \leq 6$, は生きた 6-cycle が 6-cycle に隣接する周囲 6 本の枝のうち i 本が既に選択されている状態を示す.

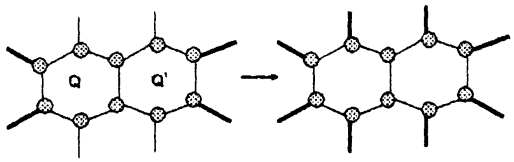

図 5: Case 1

また, Fig. 3 (b) で示された分岐を A-分岐, Fig. 4 で示された分岐を B-分岐, その他の分岐を D-分岐と呼ぶ. A-分岐と B-分岐が繰り返し呼ばれるのが最悪のケースである (Fig. 3 (b) は周囲に既に選択された枝があれば 3 本よりも多くの枝を選択する. この場合, この分岐は A-分岐ではなく D-分岐とする.)

以上の場合, 以下の lemma が成立する. :

Lemma 1. *P をバックトラックツリーの一本のパスとし, P において $C(6)$ (たとえば Q) を処理するとすると, Q に隣接する枝のうち少なくとも 3 本は D-分岐により選択された枝である.*

Proof. Q は 本アルゴリズムの開始時点においては $C(0)$ であり, 例えば $C(0) \to C(3) \to C(5) \to C(6)$ のような状態変化を経て $C(6)$ となる. もし, 一度に $C(0)$ から $C(3)$ になったとしたら, $C(3)$ から $C(6)$ ではどれかの 6-cycle(s) によって D-分岐が呼ばれるため本 lemma は成立する (前節の 3 (b1) 参照). $C(3), C(4)$ および $C(5)$ のときには D-分岐しか呼ばれないことは場合を列挙すれば簡単に示せる.

さらに, Q が $C(2)$ から $C(6)$ に変化したとすると, これは D-branch であり本 lemma が成立する.

よって Q が $C(4)$ または $C(5)$ を通して $C(6)$ になった場合について議論すればよい. 他の場合は簡単に示せるので, 以下では Q が一旦 $C(4)$ になったあとで $C(6)$ 担った場合について考える.

Q が $C(4)$ であるとすると, その前の状態は $C(0), C(1)$ または $C(2)$ である.(上で述べたように $C(3)$ の場合は考慮しない).

もし Q が $C(0)$ から $C(4)$ に直接変化したとすれば, 4 本枝が追加されているので D-branch である. Q が $C(1)$ から $C(4)$ に変化した場合も同様に省略できる. よって, 以下では Q は $C(2)$ から $C(4)$ に直接変化し, 最終的に $C(6)$ となった場合について調べればよい. $C(2)$ から $C(4)$ への変化が D-分岐によるものであればこの lemma は成立する.

Q が $C(2)$ から $C(4)$ へ D-branch 以外によって変化する状況を考える. この分岐は B-分岐では明らかに不可能であるため, A ー分岐によるものになる. さらに, A-分岐が生じるため $C(3)$ 以上の 6-cycle は存在しない. (我々は lemma に従わない $C(4)$ がこのアルゴリズムの探索パス中において初めて現れる場合を考えている. 以前に現れた $C(4)$ はこの段階では全て死んでいるものとする.) このような状態においてわれわれが注意しなければいけないのは Q が $C(4)$ となるとき, 同時に $C(4)$ になる Q' が存在している場合である. なぜなら, Q が単体で $C(4)$ になった場合, 次のステップでその 6-cycle は死んでしまい, $C(6)$ にならないからである.

一回の A-分岐が Q と Q' の 2 つの $C(4)$ を生成したとする. まず, Q と Q' が切り離されているか cycle の枝を共有していないという状況は起こりえない. なぜならそれらが切り離されるか枝を共有していなければ 1 回の A-分岐によって 4 本の枝を追加しなければいけないからである.

つまり, Q と Q' は選択された共通の枝を持ち, かつ Q および Q' 自身の枝は選択されていない. よって, Q と Q' は何本かの枝を共有する.

Q と Q' が $C(2)$ で枝を共有している状態を列挙すると, 共有している枝の本数ごとに Fig. 5 ,

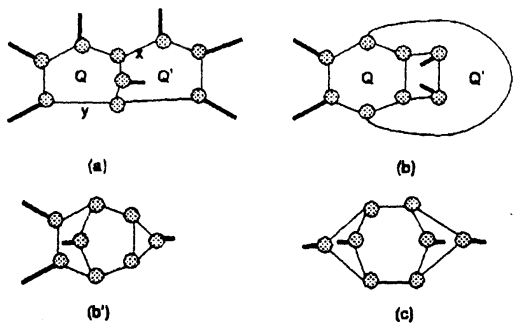

図 6: Case 2

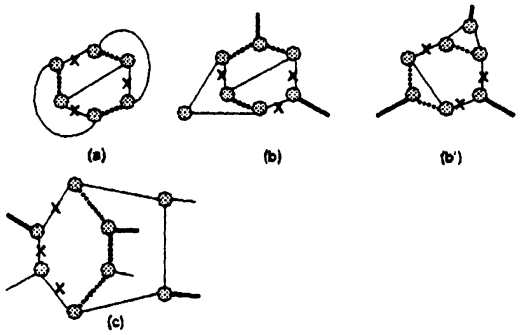

図 7: Case 3

Fig. 6 (a) (b) (b') (c), Fig. 7 (a) (b) (b') (c) , Fig. 8 (a) (b) のいずれかである. (x 枝が共有枝, Q の枝が直線, Q' 枝が破線で示されている).

これはいずれも 3-cycle を含む, 4-cycle の対角が選択されている等構造上 2 つの $C(6)$ を作ることができないか, または $C(4)$ を作る途中で D-分岐が発生するため 本 lemma を満たす.

以上, すべての場合について本 lemma は成立する □

また, 本アルゴリズムの正当性については, [8] からのアルゴリズム自体の変更は少なく, [8] の正当性を阻害していないため, 自明である.

以上の結果を元に, 本アルゴリズムの計算量を解析する.

Theorem 1. 本アルゴリズムは $O(n^{31n/96})$ で解く事ができる.

Proof. 本アルゴリズムにおいて計算量に直接影響してくる再帰呼び出し部分について要約すると以下のようになる (4-cycle の処理を含めたその他の部分についての詳細は [8] を参照).

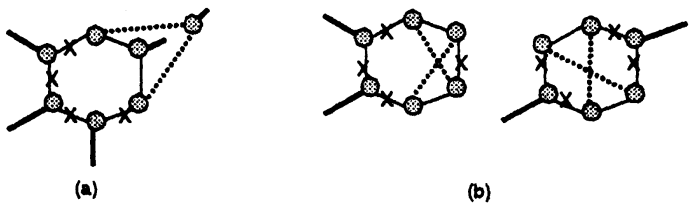

図 8: Case 4

(1) Fig. 3 (b) の場合, どちらの分岐においても新しく 3 本の枝が選択され, フリーな頂点が 4 つ減少する.

(2) Fig. 4 の場合, どちらの分岐においても新しく 3 本の枝が選択され, フリーな頂点の数は変わらない.

(3) Fig. 3 (c) の場合, 片方の分岐においては 2 本, もう片方の分岐においては 5 本の枝が新しく選択され, どちらの分岐においてもフリーな頂点が 2 つ以上失われる.

(4) Fig. 3 (d) の場合, どちらの分岐においても新しく 4 本以上の枝が選択され, フリーな頂点が 2 つ以上減少する. ただし, 8-cycle 以上の大きさの偶数長の cycle で cycle の枝が一本も選択されておらず, かつ cycle の隣接枝が全て選択されている場合にはフリーな頂点の減少はない.

ここで, $T(n,a,b,f)$ を G に対するバックトラックツリーにおける根から葉までのパスの数とする. 各引数は (i) あと n 本の枝を選択する必要がある, (ii) Fig. 3 (b) 型の分岐がこれから a 回起きる, (iii) the Fig. 4 型の分岐がこれから b 回起きる, (iv) G は f 個のフリーな頂点を持つ, ということを表す.

するとケース (1) から ケース (4) により以下のような関係が成立する :

$$T(n,a,b,f) \le \max \begin{cases} 2T(n-3,a-1,b,f-4) \\ 2T(n-3,a,b-1,f) \\ T(n-5,a,b,f-2)+T(n-2,a,b,f-2) \\ 2T(n-4,a,b,f) \end{cases}$$

なお, 根においては $T(0,0,0,0)=1$ となる.

ここで $T(n,a,b,f)=2^{\frac{n+\frac{1}{2}(a+2b)+f/8}{4}}$ とすると, 以下のように上記条件を全て満たす:

$$\begin{aligned} 2T(n-3,a-1,b,f-4) &= 2^{\frac{(n-3)+\frac{1}{2}((a-1)+2b)+(f-4)/8}{4}+1} = 2^{\frac{n+\frac{1}{2}(a+2b)+f/8}{4}} \\ 2T(n-3,a,b-1,f) &= 2^{\frac{(n-3)+\frac{1}{2}(a+2(b-1))+f/8}{4}+1} = 2^{\frac{n+\frac{1}{2}(a+2b)+f/8}{4}} \\ 2T(n-4,a,b,f) &= 2^{\frac{(n-4)+\frac{1}{2}(a+2b)+f/8}{4}+1} = 2^{\frac{n+\frac{1}{2}(a+2b)+f/8}{4}} \\ T(n-5,a,b,f-2)+T(n-2,a,b,f-2) &= 2^{\frac{(n-5)+\frac{1}{2}(a+2b)+(f-2)/8}{4}} + 2^{\frac{(n-2)+\frac{1}{2}(a+2b)+(f-2)/8}{4}} \\ &= (2^{\frac{-5-2/8}{4}} + 2^{\frac{-2-2/8}{4}})2^{\frac{n+\frac{1}{2}(a+2b)+f/8}{4}} \\ &> (1.07)2^{\frac{n+\frac{1}{2}(a+2b)+f/8}{4}} > 2^{\frac{n+\frac{1}{2}(a+2b)+f/8}{4}} \\ T(0,0,0,0) &= 2^0 = 1 \end{aligned}$$

ここで, 根から葉までのパスにおいて A-分岐 a 回, B-分岐 b 回で選択される枝の数は $3a+3b$ 本で, D-分岐により選択される枝は lemma.1 より $3b$ 本以上である. 選択される枝の数は n 本なので

$$3a+3b+3b \le n$$

となり, $a+2b \le n/3$ が成立する. また, 明らかに $f \le n$ なので

$$2^{\frac{n+\frac{1}{2}(a+2b)+n/8}{4}} \le 2^{\frac{n+\frac{1}{2}n/3+n/8}{4}} = 2^{31n/96}.$$

□

参考文献

[1] M. Held and R.M. Karp, A dynamic programming approach to sequencing problems, SIAM Journal on Applied Mathematics, Volume 10, 1962, 196-210

[2] Michael R. Garey and David S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W H Freeman, 1979

[3] Richard J. Lipton and Robert E. Tarjan, A separator theorem for planar graphs, SIAM Journal on Applied Mathematics, Volume 36, 1979, pp. 177-189.

[4] Richard J. Lipton and Robert E. Tarjan, Applications of a planar separator theorem, SIAM Journal on Computing, Volume 9, 1980, 615-627

[5] R. Z. Hwang, R. C. Chang, R. C. T. Lee, The Searching over Separators Strategy To Solve Some NP-Hard Problems in Subexponential Time, Algorithmica, Volume 9, 1993, 398-423.

[6] K. Iwama and S. Tamaki, Improved Upper Bounds for 3-SAT, 15th annual ACM-SIAM Symposium on Discrete Algorithms, Proc. SODA 2004, January, 2004, 328-329

[7] Gerhard J. Woeginger, Exact Algorithms for NP-Hard Problems: A Survey, Lecture Notes in Computer Science, Volume 2570, Jan 2003, 185-207

[8] David Eppstein, The Traveling Salesman Problem for Cubic Graphs, Lecture Notes in Computer Science, Volume 2748, Sep 2003, 307-318

[9] Frederic Dorn, Eelko Penninkx, Hans L. Bodlaender, and Fedor V. Fomin, Efficient Exact Algorithms on Planar Graphs:Exploiting Sphere Cut Branch Decompositions, ESA 2005, LNCS 3669, 2005, 95-106

図 9: Eppstein のアルゴリズム.

1. Repeat the following steps until one of the steps returns or none of them applies:
   (a) If G contains a vertex with degree zero or one, return None.
   (b) If G contains a vertex with degree two, add its incident edges to F.
   (c) If F consists of a Hamiltonian cycle, return the cost of this cycle.
   (d) If F contains a non-Hamiltonian cycle, return None.
   (e) If F contains three edges meeting at a vertex, return None.
   (f) If F contains exactly two edges meeting at some vertex, remove from G that vertex and any other edge incident to it; replace the two edges by a single forced edge connecting their other two endpoints, having as its cost the sum of the costs of the two replaced edges costs.
   (g) If G contains two parallel edges, at least one of which is not in F, and G has more than two vertices, then remove from G whichever of the two edges is unforced and has larger cost.
   (h) If G contains a self-loop which is not in F, and G has more than one vertex, remove the self-loop from G.
   (i) If G contains a triangle xyz, then for each non-triangle edge e incident to a triangle vertex, increase the cost of e by the cost of the opposite triangle edge. Also, if the triangle edge opposite e belongs to F, add e to F. Remove from G the three triangle edges, and contract the three triangle vertices into a single supervertex.
   (j) If G contains a cycle of four unforced edges, two opposite vertices of which are each incident to a forced edge outside the cycle, then add to F all non-cycle edges that are incident to a vertex of the cycle.
2. If $G\backslash F$ forms a collection of disjoint 4-cycles, perform the following steps.
   (a) For each 4-cycle C_i in $G\backslash F$, let H_i consist of two opposite edges of C_i, chosen so that the cost of H_i is less than or equal to the cost of $C_i\backslash H_i$.
   (b) Let $H = \cup_i H_i$. Then $F \cup H$ is a degree-two spanning subgraph of G, but may not be connected.
   (c) Form a graph $G' = (V', E')$, where the vertices of V' consist of the connected components of $F \cup H$. For each set H_i that contains edges from two different components K_j and K_k, draw an edge in E' between the corresponding two vertices, with cost equal to the difference between the costs of C_i and of H_i.
   (d) Compute the minimum spanning tree of (G', E').
   (e) Return the sum of the costs of $F \cup H$ and of the minimum spanning tree.
3. Choose an edge yz according to the following cases:
   (a) If $G\backslash F$ contains a 4-cycle, two vertices of which are adjacent to edges in F, let y be one of the other two vertices of the cycle and let yz be an edge of $G\backslash F$ that does not belong to the cycle.
   (b) If there is no such 4-cycle, but F is nonempty, let xy be any edge in F and yz be an adjacent edge in G F.
   (c) If F is empty, let yz be any edge in G.
4. Call the algorithm recursively on $G, F \cup \{yz\}$.
5. Call the algorithm recursively on $G\backslash\{yz\}, F$.
6. Return the minimum of the set of at most two numbers returned by the two recursive calls.