2008－AL－118 (3)
2008／5／27

無向グラフにおいて１つのオイラー小径を求める並列アルゴリズム

多田 昭雄，松本 吉實，吉岡 大三郎

崇城大学 情報学部

概要 オイラー小径を求める逐次アルゴリズムとしてフラーリーのアルゴリズムがあるが，効率良い並列アルゴリズムは見当たらない．本稿では，CREW-PRAM 計算機モデルのもとで，無向連結グラフにおいて１つのオイラー小径を求める並列アルゴリズムを提案する．具体的には，はじめにグラフがオイラー小径をもつグラフか否かを判定する．次に，グラフの各節点の次数が高々２になるように節点をいくつかの新節点に分割し，そして新節点間を辺で連結して１つのオイラー小径を求める並列アルゴリズムである．提案する並列アルゴリズムの計算量は，節点数 n，辺数 m とするとき，プロセッサ数 $O(n+m)$ を用いて時間量が $O(\log(n+m))$ となる．

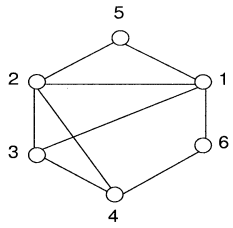
Parallel Algorithm for Finding an Eulerian Path in an Undirected Graph

Akio Tada, Yoshimi Matsumoto, Daisaburo Yoshioka
Faculty of Computer and Information Sciences, Sojo University

Abstract There is Fleury's algorithm as a sequential algorithm for finding an Eulerian path. But we cannot find a parallel algorithm for this problem. In this paper, we proposed an efficient parallel algorithm for finding an Eulerian path in an undirected graph with n vertices and m edges on a CREW-PRAM model. Namely, the proposed algorithm initially examines whether a given graph has an Eulerian path or not. Next, we divide each vertex into several new vertices which have at most 2 edge degrees, and connect between new vertices by edges. As the result, we find the Eulerian path in the given graph. The proposed algorithm requires $O(n+m)$ processors and $O(\log(n+m))$ time on a CREW-PRAM model.

1 はじめに

オイラー小径とはグラフが一筆書きできるときの経路である．オイラー小径をもつグラフの条件は奇数次数の節点がちょうど２つあるときか，すべての節点の次数が偶数のときである．オイラー小径を求める逐次アルゴリズムとしてフラーリーのアルゴリズム（Fleury's algorithm）[1] があるが，この問題に対する効率良い並列アルゴリズムは見当たらない．本稿では，CREW-PRAM 計算機モデルのもとで，無向連結グラフに対して基本的な並列アルゴリズムのみを使用して１つのオイラー小径を求める並列アルゴリズムを提案する．具体的には，はじめにグラフがオイラー小径をもつグラフか否かを判定する．次に，グラフの各節点の次数が高々２になるように節点をいくつかの新節点に分割 [2, 3] し，そして新節点間を辺で連結して１つのオイラー小径を求める並列アルゴリズムである．提案する並列アルゴリズムの計算量は，節点数 n，辺数 m とするとき，プロセッサ数 $O(n+m)$ を用いて時間量が $O(\log(n+m))$ となる．



(a) 入力グラフの例

	1	2	3	4	5	6	7	8	9
V1	1	1	1	1	2	2	2	3	4
V2	2	3	5	6	3	4	5	4	6

(b) 辺を表す配列

図 1: 所与の無向グラフ

2 提案する並列アルゴリズムの概要

対象とするグラフは，自己ループや多重辺を含む無向連結グラフ（節点の集合 V，辺の集合 E，$|V| = n$，$|E| = m$）とする．節点は次数の多い順に 1 から通し番号を付けて表す．もしも，各節点に任意の番号が与えられたときには，次数の多い順に節点を再番号付けしてグラフを再構成する．所与のグラフは各辺を配列 V1 と V2 の節点番号の形で表す．ただし，$V1[i] \leqq V2[i]$（$1 \leqq i \leqq m$）で，配列 V1 の節点は整列されていると仮定する．具体例として，図 1(a) に入力グラフの例を示し，図 1(b) にその入力配列の例を示す．

提案するアルゴリズムは次の 2 つのステージから構成される．

ステージ 1 与えられたグラフがオイラー小径をもつグラフか否かを判定する並列アルゴリズム

ステージ 2 グラフの節点を次数が高々 2 の新節点に分割して 1 つのオイラー小径を求める並列アルゴリズム

3 提案する並列アルゴリズムの詳細化

3.1 与えられたグラフがオイラー小径をもつグラフか否かを判定する並列アルゴリズム（ステージ 1）

オイラー小径をもつグラフか否かを判定するために，各節点の次数を調べる．入力グラフに次数が奇数の節点がちょうど 2 つあるか，グラフのすべての節点の次数が偶数かを判定し，これら以外のときはアルゴリズムを終了する．

すべての節点の次数が偶数の場合には，ある 1 つの辺を削除して 2 つの奇数次数をもつ節点をつくる．そして，以降の処理はグラフに奇数次数の節点が 2 つあるとして処理をすすめる．以下にステージ 1 の詳細を示す．

ステップ［1．1］ 各節点の次数を求める

はじめに，入力配列の V1 と V2 を 1 つの配列 V1V2[1..2m] として昇順に整列する．このとき，後の処理のために，先頭の辺の節点から V1 と V2 の要素を交互に並べ，さらに後に新節点に置き換えるために，インデックスを伴って整列する．

次に，整列された配列 V1V2 において，その要素の節点番号が異なる境界の辺番号を配列 B[1..n] の要素として格納する．そして，配列 B において，1 つ前の境界の辺番号との差から各節点の次数を配列 D[1..n] に求める．

ステップ［1．2］ オイラー小径をもつグラフか否かの判定

各節点の次数について，奇数次数の節点数が 0（すべての次数が偶数）または 2 の場合がオイラー小径をもつグラフである．判定は各節点の次数を 2 で割り余りを求める．そして，余りの合計が 0 または 2 がオイラー小径をもつグラフである．

余りの合計が 0 の場合，すなわちすべての次数が偶数の場合には，入力グラフの先頭の辺を一時削除し，以降の処理を奇数次数が 2 となるように統一する．

そして，奇数次数の節点の小さい方の節点番

```
procedure Stage-1(V1[1..m],V2[1..m]);
 { ステージ1の並列アルゴリズム }
begin
   { ステップ [1.1]  次数を求める }
   for e := 1 to m in parallel do begin
      V1V2[2(e-1)+1] ← V1[e];
      V1V2[2e] ← V2[e]
   end;
   Para_Sort( V1V2[1..2m]); { インデックスを伴って昇順に整列 }
   { 境界の辺番号を求める }
   for e := 1 to 2m-1 in parallel do
      if V1V2[e] ≠ V1V2[e+1] then B[V1V2[e] ← e;
   B[V1V2[2m]] ← 2m;
   D[1] ← B[1];
   for v := 2 to n in parallel do
      D[v] ← B[v] - B[v-1]; { 各節点の次数 }

   { ステップ [1.2]  オイラー小径をもつグラフか否かの判定 }
   for v := 1 to n in parallel do
      DD[v] ← D[v] mod 2;
   Para_Prefixsum( DD[1..n]); { 合計をDD[n]に求める }
   if ( DD[n] ≠ 0 AND DD[n] ≠ 2 ) then exit(0);
   if ( DD[n] = 0) then begin
      先頭の辺を削除してステップ [1.1] を再度実行
   end;
   DD[0] ← 0;
   for v := 1 to n in parallel do
      if ( DD[v] = 1 AND DD[v-1] ≠ 1) then odd ← v
end;
```

図 2: ステージ 1 のアルゴリズム

号を変数 odd に保存する.

ステージ 1 のアルゴリズムの概要を擬似コードによって図 2 に示す. なお擬似コードの中で定義した配列等の初期値はすべて 0 と仮定する.

ステージ 1 のアルゴリズムの正当性と計算量に関する考察

ステップ [1.1] では, 各節点の次数を求めるのに, 入力配列の要素である節点番号を昇順に整列して, 同一の節点番号の個数を求めているので, 次数を算定するアルゴリズムの正当性は明らかである.

整列に要する計算量は, プロセッサ数が高々 $2m$ 個なので $O(m)$ で, 時間量は並列整列アルゴリズム [6] を用いて $O(\log m)$ である. また, 整列を除く処理は定数時間でできる. 従って, このステップの計算量は, プロセッサ数が $O(m)$ で, 時間量は $O(\log m)$ である.

ステップ [1.2] では, オイラー小径をもつグラフか否かを判定するのに, 各節点の次数を 2 で割ってその余りを求めて, 奇数節点の数を計算して判定しているので, その正当性は自明である.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
V1V2	1	1	1	1	2	2	2	2	3	3	3	4	4	4	5	5	6	6

図 3: 整列された入力グラフの節点

	1	2	3	4	5	6
B	4	8	11	14	16	18

(a) 境界の辺番号

	1	2	3	4	5	6
D	4	4	3	3	2	2

(b) 各節点の次数

図 4: 各節点の次数

このステップの計算量は奇数節点の数を計算するときに並列累計和アルゴリズム [4] (Para_Prefixsum) を用いているので, プロセッサ数が $O(n)$ で, 時間量は $O(\log n)$ である.

以上の考察から, ステージ 1 の計算量はプロセッサ数が $O(m)$ で, 時間量は $O(\log m)$ である.

[具体例]

図 1 の具体例において, ステップ [1.1] で求めた入力配列が整列された配列 V1V2 を図 3 に示し, また境界の辺番号を表す配列 B と各節点の次数を表す配列 D をそれぞれ図 4(a) と (b) に示す.

次に, ステップ [1.2] のオイラー小径をもつグラフか否かの判定では, 各節点の次数は図 4(b) より奇数節点が 3 と 4 の 2 つであるので, オイラー小径をもつグラフと判定され, 辺数 odd には 3 が入る.

3.2 グラフの節点を次数が高々 2 の新節点に分割して 1 つのオイラー小径を求める並列アルゴリズム (ステージ 2)

オイラー小径を求める逐次アルゴリズムとしてフラーリーのアルゴリズム (Fleury's algorithm) が知られている. このアルゴリズムは複数の分岐があるとき辺の選択として橋になる辺をできるだけ避けるというものである.

本アルゴリズムでは辺を連結するときにフラーリーのアルゴリズムに近い形で並列に同時に連結できるようなアルゴリズムを提案する．

具体的には，はじめに所与のグラフの各節点を次数が高々２の新節点に分割する．そして分割された新節点の節点番号で入力配列の旧節点番号を置き換えて，１つのオイラー小径を求める並列アルゴリズムである．

旧節点番号を新節点番号に置き換えるとき，入力グラフにおいて，次数が多い節点から節点番号を割り当ててあるので，次数の多い順に辺が連結され，フラーリーのアルゴリズムに近似した並列アルゴリズムとなる．以下にステージ２の詳細を示す．

ステップ［２．１］　旧節点と新節点の対応表を作成する

はじめに，所与のグラフの各節点（旧節点と呼ぶ）を高々２の次数の新節点に分割するために，各節点の次数を示す配列Ｄの要素をそれぞれ２で割る．奇数次数は切り上げる．この配列Ｄの要素の値は旧節点が分割されてできる新節点の数を表す．

次に，配列Ｄより新節点数の累計和S[1..n]を求め，旧節点番号ｖと新節点番号ｕの対応表N[1..S[n]]を作成する．対応表Ｎにおいて，要素の値が旧節点番号で添字（インデックス）が新節点番号である．

ステップ［２．２］　入力配列を新節点番号に置き換える

図1(b)の入力配列の要素が示す旧節点番号を新節点番号に置き換える．はじめに，ステップ[1.1]で求めた旧節点番号が整列された配列V1V2の要素を同じくステップ[1.1]で求めた境界の辺番号を示す配列Ｂとステップ[2.1]で求めた新節点数の累計和Ｓを用いて新節点番号に置き換える．配列V1V2の各要素が示す旧節点番号は累計和Ｓから新節点の開始番号が分かり，配列Ｂから旧節点の同じ番号が何番目にあるかが分かるので，同時に並列に旧節点は新節点番号に置き換わる．新節点番号に置き換わった結果は配列NV1V2[1..2m]に格納する．

このとき，各旧節点は先頭から２つずつ同一番号で新節点に置き換わる．ただし，配列oddに格納された奇数節点はオイラー小径の先頭を示すために先頭の節点のみ（１つ）とし，以降は２つずつ同一番号で新節点に置き換わる．

そして，配列NV1V2より，インデックスを参照して，元の入力配列の節点番号を新節点番号に置き換えれば，１つのオイラー小径が形成される．

なお，入力グラフがすべて偶数の場合には，削除した辺を復元し，新節点に置き換える．

ステージ２のアルゴリズムの概要を図５に示す．

```
procedure Stage-2( var D[1..n], B[1..n], V1V2[1..2m]);
{ ステージ2の並列アルゴリズム }
begin
  { ステップ[2.1]　旧節点と新節点の対応表を作成する }
  for v := 1 to m in parallel do
    D[v] ← D[v]/2; { 新節点数 }
  Para_Prefixsum( D[1..n], S[1..n]); { 新節点数の累計和を求める }
  N[1] ← 1;
  for v := 2 to m in parallel do begin
    u ← S[v-1] + 1; N[u] ← v { 新節点uと旧節点vの対応表Nを作る }
  end;
  Boadcast-1(N[1..S[n]); { N[u]=0ならば旧節点番号で埋める }

  { ステップ[2.2]　入力配列を新節点番号に置き換える }
  B[0] ← 0; S[0] ← 0;
  for e := 1 to 2m in parallel do
    if odd = e then NV1V2[e] ← S[V1V2[e]-1] + [e+1 - B[V1V2[e] - 1] ]/2
          else NV1V2[e] ← S[V1V2[e]-1] + [e - B[V1V2[e] - 1] ]/2;
  配列NV1V2をインデックスを参照して入力グラフの配列V1とV2の旧節点番号を
  新節点番号に置き換える
  入力グラフがすべて偶数の場合には削除した辺を復元し新節点に置き換える
end;
```

図 5: ステージ２のアルゴリズム

ステージ２のアルゴリズムの正当性と計算量に関する考察

ステップ[2.1]では，新節点の個数は各節点の次数を２で割り，それらの数の累計和を求めている．累計和はS[各節点]に総数はS[n]に格納される．これらを基に，旧節点と新節点の対応表を作成しているので，正当性は明らかである．

計算量は，新節点の総数は高々$(n+m/2)$個

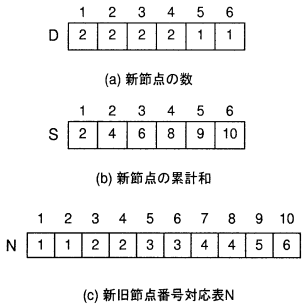


図 6: ステップ [2.1] の処理

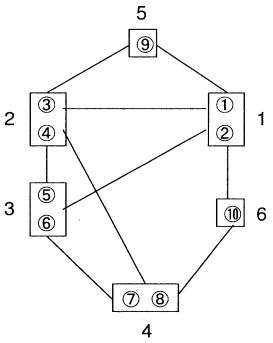


図 7: 分割された新節点

であるので，各新節点に 1 台のプロセッサを割り当てると，プロセッサ数が $O(n+m)$ で，時間量は対応表を作成するときにブロードキャスト (Broadcast-1) を用いているので，$O(\log(n+m))$ である．

ステップ [2.2] では，累計和 S と境界辺番号 B を用いて旧節点（配列 V1V2）を新節点（配列 NV1V2）に置き換え，さらに新節点（配列 NV1V2）をインデックスを用いて入力グラフ（配列 V1 と V2）に置き換えているので，正当性が自明である．

計算量は，節点の置き換えは同時に並列に定数時間でできるので，プロセッサ数は $O(m)$ で，時間量は $O(1)$ である．

従って，ステージ 2 の計算量はプロセッサ数が $O(n+m)$ で，時間量は $O(\log(n+m))$ である．

[具体例]

具体例において，ステップ [2.1] の処理を図 6 に示す．図 6(a) は旧節点が分割してできる新節点の数である．また，図 6(b) は新節点の累計和 S を示している．図 6(b) を利用して作成された新旧節点対応表 N を図 6(c) に示す．変数 odd には 3 が入っているので，旧節点 3 は 1 つの新節点のみに置き換わる．図 6(c) において，要素の値が旧節点番号で添字（インデックス）が新節点番号である．

従って，所与のグラフは図 7 のように旧節点がいくつかの新節点に分割される．図 7 において，矩形の外側の数字が旧節点番号で○内の数字が新節点番号を示す．

図 8: 新節点に置き換えられた入力配列

次にステップ [2.2] において，辺番号 B と累計和 S を使用して旧節点番号を新節点番号に置き換えた配列 NV1V2 を図 8(a) に示す．そして，元の入力配列の節点番号を新節点番号に置き換えた結果を図 8(b) に示す．

この結果を図 7 に基づいて表せば，図 9 のようになり，(5)–(1)–(3)–(6)–(7)–(4)–(9)–(2)–(10)–(8) からなる 1 つのオイラー小径が求められる．すなわち，元の入力グラフでは，3–1–2–3–4–2–5–1–6–4 となるオイラー小径である．

4 まとめ

無向グラフに対して，基本的な並列アルゴリズムのみを用いてオイラー小径を求める並列アルゴリズムを提案した．このアルゴリズムは

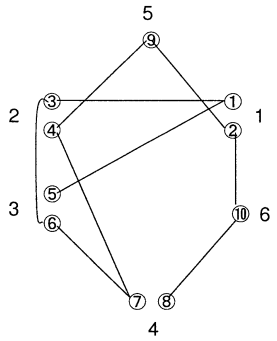


図 9: オイラー小径の例

CREW-PRAM 計算機モデルで，プロセッサ数が $O(n+m)$，計算時間が $O(\log(n+m))$ で実行できる．

オイラー小径を求めるフラーリーの逐次アルゴリズムの時間計算量は $\Theta(n+m)$ であるが，提案した並列アルゴリズムのコストは $O((n+m)\log(n+m))$ であるから，まだ最適アルゴリズムではない．しかしBrentの定理[4]を適用して最適化することは容易である．すなわちデータ構造として配列を用いているので，$\log(n+m)$ 個の配列の要素を1台のプロセッサに受け持たせることによって可能である．

参考文献

[1] 一森哲男：グラフ理論, pp.64–70, 共立出版株式会社 (2002).

[2] 多田昭雄，右田雅裕，中村良三：並列トポロジカル整列アルゴリズム，情報処理学会論文誌，Vol.45, No.4, pp.1102-1111 (2004).

[3] 右田雅裕，多田昭雄，糸川剛，中村良三：PERTチャートにおけるクリティカルパスを求める並列アルゴリズム，情報処理学会論文誌，Vol.47, No.7, pp.2212-2223 (2006).

[4] Gibbons, A. and Rytter, W.: *Efficient Parallel Algorithms*, Cambridge University Press, pp.6–18 (1988).

[5] Xavier, C. and Iyegar, S.S.: *Introduction to Parallel Algorithms*, Wiley-inter science, pp.108–140 (1998).

[6] Cole, R.: Parallel Merge Sort, *SIAM J. Comput.*, Vol.17, No.4, pp.770–785 (1988).