アルゴリズム 42－6
（1994．11．18）

極大パス集合を求めるＮＣアルゴリズムと文字列の圧縮への応用

陳 致中

東京電機大学 理工学部 数理学科

与えられた（無向または有向）グラフの極大パス集合を求めるＮＣアルゴリズムを提案する。このアルゴリズムを用いて、最短超文字列問題の並列近似アルゴリズムを新しく与える。この並列近似アルゴリズムは $\frac{1}{3+\epsilon}$ $(\epsilon > 0)$ の圧縮率を持つ。これは、今までの並列近似アルゴリズムによる最高圧縮率 $\frac{1}{4+\epsilon}$ を改善した。また、与えられた（無向または有向）グラフの極大誘導パス集合を求めるＮＣアルゴリズムを提案する。

NC Algorithms for Finding a Maximal Set of Paths with Application to Compressing Strings

Zhi-Zhong Chen

Department of Mathematical Sciences, Tokyo Denki University
Hatoyama, Saitama 350-03, Japan.
Email address: chen@r.dendai.ac.jp

It is shown that the problem of finding a maximal set of paths in a given (undirected or directed) graph is in *NC*. This result is then used to obtain an *NC* approximation algorithm for the shortest superstring problem with a compression ratio of $\frac{1}{3+\epsilon}$ for any $\epsilon > 0$. This improves on the previous best ratio of $\frac{1}{4+\epsilon}$. It is also shown that the problem of finding a maximal set of *induced* paths in a given (undirected or directed) graph is in *NC*.

1 Introduction

In [2], Aggarwal and Anderson considered the problem of finding a maximal set of paths with *specified* endpoints in a given (undirected or directed) graph. To be precise, let us give a brief review of their problem. Following [1], we call their problem the *maximal paths problem*. Given an undirected graph $G = (V, E)$ and two sets $S \subseteq V$ and $T \subseteq V$ with $S \cap T = \emptyset$, the maximal paths problem requires the computation of a maximal set of vertex-disjoint paths in G such that one of the endpoints of each path is a vertex in S and the other is a vertex in T; furthermore, the remaining path does not contain any vertices of S or of T. The problem for directed graphs is difined similarly. In [2], Aggarwal and Anderson showed that the maximal paths problem is in RNC. Soon thereafter, Goldberg *et al.* gave a sublinear-time parallel algorithm for the same problem [9]. However, it still remains open whether the maximal paths problem is in NC.

In section 2, we consider the problem of finding a maximal set of paths with *unspecified* endpoints in a given (undirected or directed) graph. Let us make our problem clearer. Given an undirected graph $G = (V, E)$, our problem requires the computation of a maximal subset F of E such that the subgraph of G induced by F is a forest in which each connected component is a path. We call this problem the *maximal path set* (MPS) problem. The problem for directed graphs is defined similarly. Note that the maximal paths problem can be stated in a similar way: Given an undirected graph $G = (V, E)$ and two sets $S \subseteq V$ and $T \subseteq V$ with $S \cap T = \emptyset$, find a maximal subset F of E such that the subgraph of G induced by F is a forest in which each connected component is a path from a vertex in S to a vertex in T. Although it still remains open whether the maximal paths problem is in NC, we show that the MPS problem is in NC. More precisely, we show that the MPS problem can be solved in $O(\log^4 n)$ time with a linear number of processors on an ARBITRARY CRCW PRAM or in $O(\log^3 n)$ time with $O(n^5)$ processors on an EREW PRAM. Our algorithm can be extended to solving the following more general problem: Given an n-vertex undirected graph $G = (V, E)$ and a positive integer $l \leq n - 1$, find a maximal subset F of E such that the subgraph of G induced by F is a forest in which each connected component is a path of length $\leq l$. Note that if we fix the input integer l to be 1, the generalized problem becomes the maximal matching problem [13]. Thus, our result also shows that certain natural generalizations of the maximal matching problem are still in NC.

In section 3, we give an interesting application of the NC algorithm obtained in section 2. Let $S = \{s_1, \cdots, s_n\}$ be a set of n strings over an alphabet Σ. A *superstring* of S is a string s over Σ such that s contains each string $s_i \in S$ as a substring, i.e., s can be written as $u_i s_i v_i$ for some strings u_i and v_i over Σ. A *shortest superstring* of S is a superstring of S that has minimum length over all superstrings of S. The *shortest superstring problem* (SSP) is to find a shortest superstring of a given set of strings. SSP has many important applications [15, 16, 17] but is unfortunately NP-hard [7, 8]. This has motivated many researchers to find approximation algorithms with good performance guarantees for SSP [3, 5, 18, 20]. To evaluate the quality of an approximation algorithm A for SSP, two measures are usually used. One is the *approximation ratio*, defined to be $max\{\frac{|A(S)|}{opt(S)} : S$ is a set of strings$\}$, where $|A(S)|$ is the length of the superstring found by algorithm A and $opt(S)$ is the length of the shortest superstrings of S. The other is the *compression ratio*, defined to be $min\{\frac{|S|-|A(S)|}{|S|-opt(S)} : S$ is a set of strings$\}$, where $|S|$ is the sum of the lengths of the strings in S. Here, we are only interested in the compression ratio and want to design an NC approximation algorithm for SSP with a good compression ratio. Previously, only one NC approximation algorithm for SSP had been given [5]. This algorithm achieves a compression ratio of $\frac{1}{4+\epsilon}$ for

any $\epsilon > 0$. In section 3, using our NC algorithm for finding a maximal set of paths in a given directed graph, we present a new NC approximation algorithm for SSP. The algorithm achieves a compression ratio of $\frac{1}{3+\epsilon}$ for any $\epsilon > 0$. This improves on the previous best ratio due to Czumaj *et al.* [5].

In section 4, we consider the *vertex* counterpart of the MPS problem, namely, the problem of finding a maximal set of *induced* paths with unspecified endpoints in a given (undirected or directed) graph. More precisely, we consider the following problem: Given an undirected graph $G = (V, E)$, find a maximal subset U of V such that the subgraph of G induced by U is a forest in which each connected component is a path. We call this problem the *maximal induced path set* (MIPS) problem. The problem for directed graphs is defined similarly. Although the MIPS problem looks very similar to the MPS problem, designing an NC algorithm for the former is surprisingly a little more difficult. Indeed, our NC algorithm for the MIPS problem is a sophisticated NC reduction to the problem of finding a maximal independent set in a given hypergraph of dimension 3. It runs in $O(\log^5 n)$ time using $O(n^3)$ processors on an EREW PRAM. Restricting to bipartite graphs or sparse graphs (such as planar graphs), the MIPS problem can easily be reduced to the MPS problem and hence has more efficient NC algorithms.

For the definitions of the ARBITRARY CRCW PRAM and EREW PRAM models and complexity classes such as NC and RNC, the reader is referred to [14].

2 Finding a maximal set of paths

In this section, we only prove that a maximal set of paths in a given undirected graph can be found in NC. However, at the end of this section, we will point out that this result can easily be extended to directed graphs.

Throughout this section, unless stated otherwise, a graph is always undirected and simple. By a *path*, we always mean a simple path. Note that a single vertex is considered as a path (of length 0). Let $G = (V, E)$ be a graph. A set M of edges in G is called a *matching* if no two edges in M share an endpoint. A matching in G is said to be *maximal* if it is not a proper subset of another matching. For $F \subseteq E$, let $G[F]$ denote the graph (V, F). A set F of edges in G is called a *path set* if $G[F]$ is a forest in which each connected componet is a path. A *maximal path set* (MPS) in G is a path set that is not properly contained in another path set.

Our main result in this section is the following algorithm for finding an MPS in a given graph. In the algorithm, we assume that the edges in the input graph are linearly ordered, say, by indexing them with numbers between 1 through m, where m is the number of edges in the input graph. This assumption is not essential to our result.

Algorithm 1
Input: An n-vertex graph $G = (V, E)$.
Output: An MPS F in G.
1. Initialize F to be the empty set.
2. While $E \neq \emptyset$, perform the following steps:
  2.1. Construct a new graph K as follows. Corresponding to each connected component p in $G[F]$, K contains a vertex w_p. For p_1 and p_2 in $G[F]$, K contains the edge $\{w_{p_1}, w_{p_2}\}$ iff E contains some edge $\{v_1, v_2\}$ such that v_1 is contained in p_1 and v_2 is contained in p_2.
  2.2. Find a maximal matching M in K.
  2.3. In parallel, for each edge $\{w_{p_1}, w_{p_2}\} \in M$, add to F the smallest edge $\{v_1, v_2\} \in E$ such that v_1 is contained in p_1 and v_2 is contained in p_2, and then remove the edge $\{v_1, v_2\}$ from E.
  2.4. Remove from E all edges e such that $F \cup \{e\}$ is not a path set in G.
3. Output F.

Let t be the number of executions of the while-loop in Algorithm 1. In case $t = 0$, the

input graph G does not contain any edge and so Algorithm 1 is clearly correct and takes constant time. Thus, we may assume that $t \geq 1$. For $1 \leq i \leq t$, let E_i, F_i, K_i, and M_i denote the contents of the variables E, F, K, and M after the ith execution of the while-loop, respectively. For convenience, let $F_0 = \emptyset$ and E_0 be the edge set of the input graph G. For $1 \leq i \leq t$, let $M'_i = F_i - F_{i-1}$. Note that $E_t = \emptyset$ and $|M_i| = |M'_i|$ for $1 \leq i \leq t$. Let $0 \leq i \leq t-1$. An *augmentation* of F_i is a set of some edges in E_i. An augmentation A of F_i is said to be *valid* if $F_i \cup A$ is a path set in G.

Lemma 1. F_t is an MPS.

Lemma 2. $t = O(\log n)$.

Proof. For $0 \leq i \leq t-1$, let α_i be the size of a maximum valid augmentation of F_i. Let us first prove that $|M'_{i+1}| \geq \frac{\alpha_i}{4}$. Fix an integer i with $0 \leq i \leq t-1$. Let β be the size of a maximum matching in the graph K_{i+1}. We want to show that $\alpha_i \leq 2\beta$. Let A be a maximum valid augmentation of F_i. Since A is valid, each connected component of $G[F_i \cup A]$ must be a path. For each connected component p of $G[F_i \cup A]$ that contains at least one edge of A, we start at an endpoint of p and traverse p toward the other endpoint while labeling the edges of A on p by 0 and 1 alternately (the first edge of A on p is labeled 0). Let B be the set of those edges of A labeled 0. Clearly, $|B| \geq \frac{|A|}{2} = \frac{\alpha_i}{2}$. Moreover, it is easy to see that corresponding to B, there is a matching of size $|B|$ in K_{i+1}. This implies that $|B| \leq \beta$ and in turn that $\alpha_i \leq 2\beta$. On the other hand, since M_{i+1} is a maximal matching in K_{i+1}, at least one endpoint of each edge in any maximum matching must be matched in M_{i+1}. This implies that $|M_{i+1}| \geq \frac{\beta}{2}$. Using $\alpha_i \leq 2\beta$, we now have $|M'_{i+1}| = |M_{i+1}| \geq \frac{\alpha_i}{4}$.

By the proof of Lemma 1, M'_{i+1} is a valid augmentation of F_i for $0 \leq i \leq t-1$. Thus, $\alpha_{i+1} + |M'_{i+1}| \leq \alpha_i$ for $0 \leq i \leq t-1$. Since $|M'_{i+1}| \geq \frac{\alpha_i}{4}$, we now have that $\alpha_{i+1} \leq \frac{3}{4}\alpha_i$ for $0 \leq i \leq t-1$. Combining this with the fact that α_0 is no more than the number of edges in G, we obtain $t = O(\log n)$. ∎

Theorem 3. Given an n-vertex m-edge graph G, an MPS of G can be found in $O(\log^4 n)$ time with $O(n+m)$ processors on an ARBITRARY CRCW PRAM or in $O(\log^3 n)$ time with $O(n^5)$ processors on an EREW PRAM.

The following corollary will be used in section 4.

Corollary 4. Given an n-vertex m-edge graph G and a path set F' in G, an MPS F in G with $F' \subseteq F$ can be found in $O(\log^4 n)$ time with $O(n+m)$ processors on an ARBITRARY CRCW PRAM or in $O(\log^3 n)$ time with $O(n^5)$ processors on an EREW PRAM.

Let l be a positive integer. A path of length $\leq l$ is called an *$l^{\leq}$-path*. Let $G = (V, E)$ be a graph. A set F of edges in G is called an *$l^{\leq}$-path set* if $G[F]$ is a forest in which each connected componet is an $l^{\leq}$-path. A *maximal $l^{\leq}$-path set* (MPS(l)) in G is an $l^{\leq}$-path set that is not properly contained in another $l^{\leq}$-path set. Clearly, a matching in G corresponds to a $1^{\leq}$-path set in G and a maximal matching in G corresponds to an MPS(1) in G. Thus, the following corollary shows that certain generalizations of the maximal matching problem are still in NC.

Corollary 5. Given an n-vertex m-edge graph $G = (V, E)$ and a positive integer $l \leq n-1$, an MPS(l) F in G can be found in $O(\log^4 n)$ time with $O(n+m)$ processors on an ARBITRARY CRCW PRAM or in $O(\log^3 n)$ time with $O(n^5)$ processors on an EREW PRAM.

Below, we point out that Theorem 3 can be extended to digraphs. This extension will be used in the next section. Let $D = (V, A)$ be a digraph. For $F \subseteq A$, let $D[F]$ denote the digraph (V, F). A *directed path set* (DPS) in D is a subset B of A such that $D[B]$ is an acyclic digraph in which the indegree and outdegree of each vertex are both at most 1. Intuitively speaking, if B is a DPS in D, then $D[B]$ is a collection of vertex-disjoint directed paths. A

maximal directed path set (MDPS) in D is a DPS that is not properly contained in another DPS.

Corollary 6. Given an n-vertex m-arc digraph $D = (V, A)$ and a DPS B in D, an MDPS F in D with $B \subseteq F$ can be found in $O(\log^4 n)$ time with $O(n)$ processors on an ARBITRARY CRCW PRAM or in $O(\log^3 n)$ time with $O(n^5)$ processors on an EREW PRAM.

3 NC-approximation of shortest superstrings

For a string s, let $|s|$ denote the length of s. Let s and t be two strings, and let v be the longest string such that $s = uv$ and $t = vw$ for some non-empty strings u and w. $|v|$ is called the *overlap* between s and t and is denoted by $ov(s,t)$. By $s \circ t$, we denote the string uvw.

Let $S = \{s_1, s_2, \cdots, s_n\}$ be a set of strings. As in previous studies, we assume that S is *substring free*, i.e., no string in S is a substring of any other. By $opt(S)$, we denote the length of the shortest superstring of S. Define $|S| = \Sigma_{i=1}^{n}|s_i|$. The *overlap graph* of S is the arc-weighted digraph $OG(S) = (V, A, ov)$, where $V = \{1, 2, \cdots, n\}$, $A = \{(i,j) : 1 \leq i \neq j \leq n\}$, and $ov(i,j) = ov(s_i, s_j)$. For a path $p = i_0$, e_1, i_1, $\cdots$, e_k, i_k in $OG(S)$, the *weight* of p is $\Sigma_{j=1}^{k} ov(e_j)$ and is denoted by $w(p)$.

Fact 1 [20] Let $p = i_0$, e_1, i_1, $\cdots$, e_{n-1}, i_{n-1} be a Hamiltonian path with maximum weight in $OG(S)$. Then, $w(p) = |S| - opt(S)$ and $s_{i_0} \circ s_{i_1} \circ \cdots \circ s_{i_{n-1}}$ is a shortest superstring.

By Fact 1, we can find a shortest superstring by computing a Hamiltonian path with maximum weight in $OG(S)$. Unfortunately, it does not seem that there is an efficient algorithm for computing a Hamiltonian path with maximum weight in $OG(S)$. However, Turner showed that the following simple sequential algorithm finds a Hamiltonian path with weight at least $\frac{|S|-opt(S)}{2}$ in $OG(S)$ [20].

Algorithm GREEDY
Input: $OG(S) = (V, A, ov)$.
1. Initialize B to be the empty set.
2. While the digraph (V, B) is not a Hamiltonian path in $OG(S)$, perform the following: Add to B the largest arc e such that $B \cup \{e\}$ is a DPS in $OG(S)$ but $B \cup \{e'\}$ is not a DPS in $OG(S)$ for all arcs e' with $ov(e) < ov(e')$. (Note: we here assume that the arcs of equal weights in $OG(S)$ are linearly ordered.)
3. Output the digraph (V, B).

For a weighted complete digraph D, let $OptPath(D)$ be the maximum weight of a Hamiltonian path in D.

Fact 2 [20] Algorithm GREEDY finds a Hamiltonian path with weight at least $\frac{OptPath(OG(S))}{2}$ in $OG(S)$.

Fact 3 [20] Even if the input to GREEDY is changed to an arbitrary weighted complete digraph D, GREEDY finds a Hamiltonian path in D with weight at least $\frac{OptPath(D)}{3}$.

By Fact 1 and Fact 2, GREEDY achieves a compression ratio of $\frac{1}{2}$. To the best of our knowledge, $\frac{1}{2}$ is still the best known compression ratio. Thus, it seems very difficult to design an NC approximation algorithm for SSP with a better compression ratio. To design an NC approximation algorithm for SSP, one way seems to be parallelize GREEDY. However, Czumaj *et al.* proved that computing the output of GREEDY is hard for P [5]. Consequently, a complete different approach seems to be necessary. In [5], Czumaj *et al.* gave an NC approximation algorithm for SSP with a compression ratio of $\frac{1}{4+\epsilon}$ for any $\epsilon > 0$. Here, we present an NC approximation algorithm for SSP with a compression ratio of $\frac{1}{3+\epsilon}$ for any $\epsilon > 0$.

Algorithm 2
Input: $OG(S) = (V, A, ov)$.
1. Let $c = 1 + \frac{\epsilon}{3}$. In parallel, for each arc $e \in A$, set $lev(e) = \lceil \log_c ov(e) \rceil$ if $ov(e) > 1$, and set $lev(e) = 0$ otherwise.

2. Compute $MaxLev = max\{lev(e) : e \in A\}$.
3. Set $B = \emptyset$ and $CurLev = MaxLev$.
4. While $CurLev \geq 0$, perform the following steps:
   4.1. Construct an unweighted digraph $D = (V, E)$ by setting $E = B \cup \{e \in A : lev(e) = CurLev\}$.
   4.2 Compute an MDPS F in D with $B \subseteq F$ and then update B to be F.
   4.3 Decrease $CurLev$ by 1.
5. Output the digraph (V, B).

Lemma 7. Algorithm 2 finds a Hamiltonian path with weight at least $\frac{OptPath(OG(S))}{3+\epsilon}$ in $OG(S)$.

Proof. The proof is very similar to that of Lemma 3 in [5]. Let p_{out} be the output of Algorithm 2. Since $OG(S)$ is a complete (weighted) digraph, p_{out} is certainly a Hamiltonian path in $OG(S)$. We next show that $w(p_{out}) \geq \frac{OptPath(OG(S))}{3+\epsilon}$.

Define a weighted digraph $H = (V, A, w_H)$ by setting $w_H(e) = c^{lev(e)}$ if e is on p_{out} and setting $w_H(e) = ov(e)$ otherwise. Let $\succ$ be a total order satisfying the following condition: If either $w_H(e) > w_H(e')$, or $w_H(e) = w_H(e')$ and e is contained in p_{out} but e' is not contained in p_{out}, then $e \succ e'$. When $e \succ e'$, we say that e is *larger* than e'.

Obviously, if the arcs in H are sorted in nondecreasing order using $\succ$, then given H as input to both Algorithm GREEDY and Algorithm 2 (in place of $OG(S)$), the two algorithms work in the same way. Thus, by Fact 3, Algorithm 2 on input H finds a Hamiltonian path of weight at least $\frac{OptPath(H)}{3}$. On the other hand, the weight of the Hamiltonian path found by Algorithm 2 on input H is less than $c \cdot w(p_{out})$ because $w_H(e) < c \cdot ov(e)$ for all $e \in A$. Therefore, $c \cdot w(p_{out}) > \frac{OptPath(H)}{3} \geq \frac{OptPath(OG(S))}{3}$ since $w_H(e) \geq ov(e)$ for all $e \in A$. Recall that $c = 1 + \frac{\epsilon}{3}$. We now have $w(p_{out}) > \frac{OptPath(OG(S))}{3c} = \frac{OptPath(OG(S))}{3+\epsilon}$. ∎

Lemma 8. Algorithm 2 runs in $O(\log^4 n \cdot \log_{1+\epsilon/3} |S|)$ time using $O(\ |S|^2)$ processors on an ARBITRARY CRCW PRAM or in $O(\log^3 n \cdot \log_{1+\epsilon/3} |S|)$ time using $O(\ |S|^2 + n^5)$ processors on an EREW PRAM.

Theorem 9. There is an NC approximation algorithm for SSP with a compression ratio of $\frac{1}{3+\epsilon}$ for any $\epsilon > 0$. It runs in $O(\log^4 n \cdot \log_{1+\epsilon/3} |S|)$ time with $O(\ |S|^2)$ processors on an ARBITRARY CRCW PRAM or in $O(\log^3 n \cdot \log_{1+\epsilon/3} |S|)$ time with $O(\ |S|^2 + n^5)$ processors on an EREW PRAM.

4 Finding a maximal set of induced paths

In this section, we present an NC algorithm for finding a maximal set of induced paths in a given undirected graph. The extension to directed graphs is straightforward and is hence omitted here.

Hereafter, a graph is always undirected and simple. Let $G = (V, E)$ be a graph. For a vertex $v \in V$, $N_G(v)$ denotes the set $\{u : \{v, u\} \in E\}$ and $deg_G(v)$ denotes the cardinality of $N_G(v)$. For $U \subseteq V$, the *subgraph of G induced by U* is the graph (U, F) with $F = \{\{u, v\} \in E : u, v \in U\}$ and is denoted by $G[U]$. A set U of vertices in G is called an *induced path set* (IPS) if $G[U]$ is a forest in which each connected componet is a path. A *maximal induced path set* (MIPS) in G is a path set that is not properly contained in another IPS.

A *hypergraph* $H = (V, E)$ consists of a set V of *vertices* and a collection E of subsets of V called *hyperedges*. The *dimension* of H is the maximum size of a hyperedge in E. Clearly, an ordinary graph is a hypergraph of dimension 2. An *independent set* in H is a subset U of V that does not contain any hyperedge of E. A *maximal independent set* (MIS) in H is an independent set that is not properly contained in another independent set.

Algorithm 3
Input: An n-vertex graph $G = (V, E)$.
Output: An IPS U in G such that for all $v \in V - U$, $U \cup \{v\}$ is not an IPS in G or $|N_G(v) \cap U| \geq 2$.

1. Initialize U to be an MIS in G.
2. Compute V_1, the set of all $v \in V - U$ such that $U \cup \{v\}$ is an IPS in G and $|N_G(v) \cap U| = 1$.
3. Construct $K = (V_1, E_K)$, where E_K consists of all $\{v_1, v_2\}$ such that $\{v_1, v_2\} \in E$ or $N_G(v_1) \cap N_G(v_2) \cap U \neq \emptyset$.
4. Compute an MIS I in K
5. Add the vertices in I to U.
6. Compute V_1', the set of all $v \in V_1 - I$ such that $U \cup \{v\}$ is an IPS in G and $|N_G(v) \cap U| = 1$.
7. Construct $K' = (V_1', E_{K'})$, where $E_{K'}$ consists of all $\{v_1, v_2\}$ such that $\{v_1, v_2\} \in E$ or $N_G(v_1) \cap N_G(v_2) \cap U \neq \emptyset$.
8. Compute an MIS I' in K'.
9. Add the vertices in I' to U.
10. Output U.

In addition to the notations used in Algorithm 3, let U_i be the content of the variable U at the end of step i for $1 \leq i \leq 9$. Obviously, $U_1 = \cdots = U_4 \subseteq U_5 = \cdots = U_8 \subseteq U_9$.

Lemma 10. U_9 is an IPS in G.

Lemma 11. For all $v \in V - U_9$, $U_9 \cup \{v\}$ is not an IPS in G or $|N_G(v) \cap U_9| \geq 2$.

Lemma 12. Algorithm 3 is correct and runs in $O(\log^3 n)$ time with $O(n^2)$ processors on an EREW PRAM.

We now use Algorithm 3 to design an NC algorithm for finding an MIPS in a given graph.

Algorithm 4
Input: An n-vertex graph $G = (V, E)$.
Output: An MIPS U in G.
1. Use Algorithm 3 to find an IPS U in G such that for all $v \in V - U$, $U \cup \{v\}$ is not an IPS in G or $|N_G(v) \cap U| \geq 2$.
2. Set W to be the set of those vertices $v \in V - U$ such that $U \cup \{v\}$ is an IPS in G. (Comment: After this step, $|N_G(w) \cap U| = 2$ and $deg_{G[U]}(u) = 1$ for all $w \in W$ and all $u \in N_G(w) \cap U$.)
3. While $W \neq \emptyset$, perform the following steps:
   - **3.1.** Construct a hypergraph $H = (W, E_H \cup E_H')$ of dimension 3 as follows. E_H consists of all subsets $\{w_1, w_2\}$ of W such that at least one of the following (1), (2), and (3) holds: (1) $\{w_1, w_2\} \in E$; (2) there is some $u \in U$ such that $u \in N_G(w_1) \cap N_G(w_2)$ and $deg_{G[U]}(u) = 1$; (3) $G[U \cup \{w_1, w_2\}]$ has a cycle in which both w_1 and w_2 appear. E_H' consists of all subsets $\{w_1, w_2, w_3\}$ of W such that no $W' \in E_H$ is properly contained in $\{w_1, w_2, w_3\}$ and at least one of the following (a) and (b) holds: (a) there is some $u \in U$ such that $u \in N_G(w_1) \cap N_G(w_2) \cap N_G(w_3)$ and $deg_{G[U]}(u) = 0$; (b) $G[U \cup \{w_1, w_2, w_3\}]$ has a cycle in which all of w_1, w_2, and w_3 appear.
   - **3.2** Compute an MIS S in H. (Comment: It will be shown that each connected component of $G[U \cup S]$ is either a cycle containing at least 4 vertices in S or a path.)
   - **3.3.** Let $C_1, \cdots, C_k$ be the connected components of $G[U \cup S]$ that are cycles. For each $1 \leq i \leq k$, in parallel, choose two vertices x_i and y_i in C_i such that $x_i \in S$, $y_i \in S$, and the two possible paths from x_i to y_i in C_i both contain a vertex in $S - \{x_i, y_i\}$.
   - **3.4.** For each $1 \leq i \leq k$, let $u_{i,1}$, $u_{i,2}$ be the two neighbors of x_i in C_i and let $v_{i,1}$, $v_{i,2}$ be the two neighbors of y_i in C_i. Set $X = \cup_{1 \leq i \leq k}((N_G(x_i) \cup N_G(u_{i,1}) \cup N_G(u_{i,2})) \cap W) - S$ and $Y = \cup_{1 \leq i \leq k}((N_G(y_i) \cup N_G(v_{i,1}) \cup N_G(v_{i,2})) \cap W) - S$.
   - **3.5.** If $|X| \leq |Y|$, add the vertices in $S - \{x_1, \cdots, x_k\}$ to U and then set $W = \{w \in X - (X \cap Y) : U \cup \{w\}$ is an IPS in $G\}$; otherwise, add the vertices in $S - \{y_1, \cdots, y_k\}$ to U and then set $W = \{w \in Y - (X \cap Y) : U \cup \{w\}$ is an IPS in $G\}$.
4. Output U.

Let t be the number of executions of the

while-loop in Algorithm 4. In case $t = 0$, Algorithm 4 is clearly correct. Thus, we may assume $t \geq 1$. For $1 \leq j \leq t$, let W_j, U_j, H_j, S_j, X_j, and Y_j denote the contents of the variables W, U, H, S, X, and Y after the jth execution of the while-loop, respectively. For convenience, let W_0 and U_0 denote the contents of the variables W and U at the end of step 2, respectively. Clearly, for $1 \leq j \leq t$, $W_j \subseteq W_{j-1}$, $U_{j-1} \subseteq U_j$, and $U_j \subseteq U_{j-1} \cup S_j$.

Lemma 13. U_t is an MIPS in G.

Lemma 14. $t = O(\log n)$.

Theorem 15. An MIPS in an n-vertex graph G can be found in $O(\log^5 n)$ time with $O(n^3)$ processors on an EREW PRAM.

Acknowledgment The author is grateful to Professor Artur Czumaj for pointing out an error in an earlier version of this paper.

References

1. A. Aggarwal, Parallel Complexity of Computing a Maximal Set of Disjoint Paths, *Inform. Process. Lett.*, vol. 41, pp. 149-151, 1992.
2. A. Aggarwal and R. Anderson, A Random NC Algorithm for Depth-First Search, *Combinatorica* **8** (1988) 1-12.
3. A. Blum, T. Jiang, M. Li, J. Tromp, and M. Yannakakis, Linear Approximation of Shortest Superstrings, in: *Proc. 23th ACM Symp. on Theory of Computing* (ACM, 1991) 328-336.
4. R. Cole, Parallel Merge Sort, *SIAM J. Comput.* **17** (1988) 770-785.
5. A. Czumaj, L. Gasieniec, M. Piotrow, and W. Rytter, Parallel and Sequential Approximation of Shortest Superstrings, in: *Proc. 4th Scandinavian Workshop on Algorithm Theory*, Lecture Notes in Computer Science, Vol. 824 (Springer, Berlin, 1994) 95-106.
6. E. Dahlhaus, M. Karpinski, and P. Kelsen, An Efficient Parallel Algorithm for Computing a Maximal Independent Set in a Hypergraph of Dimension 3, *Inform. Process. Lett.* **42** (1992) 309-313.
7. J. Gallant, D. Maier, and J. Storer, On Finding Minimal Length Superstrings, *Journal of Computer and System Sciences* **20** (1980) 50-58.
8. M.R. Garey and D.S. Johnson, *Compters and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, New York, 1979).
9. A.V. Goldberg, S.A. Plotkin, and P.M. Vaidya, Sublinear-Time Parallel Algorithms for Matching and Related Problems, *J. of Algorithms*, vol. 14, pp. 180-213, 1993.
10. M. Goldberg, Parallel Algorithms for Three Graph Problems, *Congressus Numerantium*, vol. 54, pp. 111-121, 1986.
11. M. Goldberg and T. Spencer, Constructing a Maximal Independent Set in Parallel, *SIAM J. Disc. Math.*, vol. 2, pp. 322-328, 1989.
12. A. Israeli and A. Itai, A Fast and Simple Randomized Parallel Algorithm for Maximal Matching, *Inform. Process. Lett.* **22** (1986) 77-80.
13. A. Israeli and Y. Shiloach, An Improved Maximal Matching Parallel Algorithm, *Inform. Process. Lett.* **22** (1986) 57-60.
14. R.M. Karp and V. Ramachandran, *Parallel Algorithms for Shared Memory Machines*, in: J. van Leeuwen ed., *Handbook of Theoretical Computer Science Vol. A* (Elsevier, Amsterdam, 1990) 868-941.
15. M. Li, Towards a DNA Sequencing Theory (Learning a String), in: *Proc. 31th IEEE Symp. on Foundations of Computer Science* (IEEE, 1990) 125-134.
16. H. Peltola, H. Soderlund, J. Tarhio, and E. Ukkonen, Algorithms for Some String Matching Problems Arising in Molecular Genetics, in: *Proc. 2nd IFIP Congress* (1983) 53-64.
17. J. Storer, *Data Compression: Methods and Theory* (Computer Science Press, 1988).
18. J. Tarhio and E. Ukkonen, A Greedy Approximation Algorithm for Constructing Shortest Common Superstrings, *Theoretical Computer Science* **57** (1988) 131-145.
19. S.-H. Teng and F. Yao, Approximating Shortest Superstrings, in: *Proc. 34th IEEE Symp. on Foundations of Computer Science* (IEEE, 1993) 158-165.
20. J.-S. Turner, Approximation Algorithms for the Shortest Common Superstring Problem, *Information and Computation* **83** (1989) 1-20.