IEICE TRANS. FUNDAMENTALS, VOL. E80-A, NO. 4 APRIL 1997

PAPER Special Section on Discrete Mathematics and Its Applications

Parallel Algorithms for Maximal Linear Forests

Ryuhei UEHARA† *and* Zhi-Zhong CHEN††, *Nonmembers*

SUMMARY The maximal linear forest problem is to find, given a graph $G = (V, E)$, a maximal subset of V that induces a linear forest. Three parallel algorithms for this problem are presented. The first one is randomized and runs in $O(\log n)$ expected time using n^2 processors on a CRCW PRAM. The second one is deterministic and runs in $O(\log^2 n)$ time using n^4 processors on an EREW PRAM. The last one is deterministic and runs in $O(\log^5 n)$ time using n^3 processors on an EREW PRAM. The results put the problem in the class NC.

key words: *parallel algorithms, randomized parallel algorithms, graph algorithms, linear forests, maximal matchings, maximal independent sets*

1. Introduction

Since Karp and Wigderson showed that the maximal independent set (MIS) problem is in the class NC[11], much work has been devoted to the study of parallel complexity of maximality problems. A typical maximality problem on graphs is to find either a maximal vertex-induced subgraph (MVIS) or a maximal edge-induced subgraph (MEIS) that satisfies a specified graph property. A common feature of the MVIS or MEIS problems is that they can be solved by simple greedy algorithms but are very difficult from the parallel point of view. So far, only a few MVIS or MEIS problems have been shown to be in NC or RNC[1],[3]–[5],[8],[10],[12],[16],[18]. Parallel complexity of many natural MVIS or MEIS problems remains unknown. Among them is the MVIS problem associated with the property "acyclic." In [6], NC algorithms are given for several special cases of this problem. However, parallel complexity of this problem still remains open. Note that this problem is equivalent to the problem of computing a *maximal forest* in a given graph. In this paper, we show that a related problem, namely, the problem of computing a *maximal linear forest* in a given graph is in NC. We call this problem the MLF problem. Note that the MLF problem is equivalent to the MVIS problem associated with the property "acyclic and maximum-degree ≤ 2."

In addition to its correspondence to the problem of computing a maximal forest, there are two other reasons for us to be interested in parallel complexity of the MLF problem. First, the maximal path set (MPS) problem is studied in [5]. This problem is the edge-analogue of the MLF problem, i.e., the MEIS problem associated with the property "acyclic and maximum-degree ≤ 2." In [5],[20], it is shown that the MPS problem can be used to design parallel approximation algorithms for the shortest superstring problem, which has applications in DNA sequencing and data compression[17],[19]. On the other hand, the MPS problem can be efficiently reduced to the MLF problem as follows: To solve the MPS problem for a given graph G, it suffices to solve the MLF problem for the edge graph of G. Thus, efficient parallel algorithms for the MLF problem can be used to design parallel approximation algorithms for the shortest superstring problem, too. Second, Shoudai and Miyano[18] showed that the MVIS or MEIS problem associated with certain *local* properties can be solved in NC by an elegant reduction to the MIS problem. They also remarked that their idea of using the MIS problem (and its variants) does not seem to work if the graph property concerned is not local. Note that the property "acyclic and maximum-degree ≤ 2" is not local. One of our algorithms (the last one) for the MLF problem is a sophisticated reduction to a variant of the MIS problem (namely, the problem of finding an MIS in a given hypergraph of dimension 3). This result disproves Shoudai and Miyano's remark mentioned above.

Our RNC algorithm for the MLF problem has a similar structure to that of Luby's RNC algorithm for the MIS problem[14]. Namely, given a graph G, both Luby's algorithm and ours proceed in stages; in each stage, their main jobs are to compute a certain independent set I in a certain (augmented) subgraph of G and to delete (from G) the vertices in I and (part of) their neighbors. In Luby's algorithm, the expected number of edges deleted in each stage is a constant fraction of the number of edges in G [14]. However, our algorithm does not have this property. Instead, we define a *potential function* Φ and prove that in each stage, $\Phi(G)$ decreases by a constant fraction *on average*. This is the key for us to show that the algorithm runs very fast. It runs in $O(\log n)$ expected time using n^2 processors on a CRCW PRAM. Our first NC algorithm for the MLF problem is obtained by derandomizing the RNC algo-

Manuscript received August 27, 1996.

†The author is with the Center for Information Science, Tokyo Woman's Christian University, Tokyo, 167 Japan.

††The author is with the Department of Mathematical Sciences, Tokyo Denki University, Saitama-ken, 350-03 Japan.

rithm. It runs in $O(\log^2 n)$ time using n^4 processors on an EREW PRAM. Our second NC algorithm is a little more efficient than the first. It runs in $O(\log^5 n)$ time using n^3 processors on an EREW PRAM. This algorithm is obtained by a sophisticated reduction to the problem of finding an MIS in a given hypergraph of dimension 3. Finally, we remark that the MLF problem restricted to bipartite graphs or sparse graphs (such as planar graphs) can easily be reduced to the MPS problem and hence has more efficient NC algorithms.

Recall that the EREW PRAM is the parallel model where the processors operate synchronously and share a common memory, but no two of them are allowed simultaneous access to a memory cell (whether the access is for reading or for writing in that cell). The CRCW PRAM differs from the EREW PRAM in that both simultaneous reading and simultaneous writing to the same cell are allowed; in case of simultaneous writing, the processor with lowest index succeeds. The rest of the paper is organized as follows. In Sect. 2, we give basic definitions and reduce the MLF problem to a simpler problem. We then present the RNC algorithm and its derandomization in Sect. 3. In Sect. 4, we describe the more efficient NC algorithm. Section 5 concludes the paper and states an *interesting* open question related to this work.

2. Reducing the Problem to a Simpler One

Throughout this paper, we will be dealing only with undirected graphs without loops or multiple edges. Let $G = (V, E)$ be a graph. The *neighborhood* of a vertex v in G, denoted $N_G(v)$, is the set of vertices in G adjacent to v. The *degree* of a vertex v in G is $|N_G(v)|$, and denoted by $d_G(v)$. Vertices of degree 0 are called *isolated vertices*. For $U \subseteq V$, let $N_G(U) = \cup_{u \in U} N_G(u)$, and $G[U]$ be the graph (U, F), where $F = \{\{u, v\} \mid u, v \in U \text{ and } \{u, v\} \in E\}$. By a *path*, we always mean a simple path. Note that a single vertex is considered as a path (of length 0). A subset U of V is called a *linear forest set* (LFS) if $G[U]$ is a forest in which each connected component is a path. Intuitively speaking, if U is an LFS, then $G[U]$ is a collection of vertex-disjoint paths. A *maximal linear forest set* (MLFS) in G is an LFS that is not properly contained in another LFS. The *maximal linear forest* (MLF) problem is to find, given a graph G, an MLFS in G.

To reduce the MLF problem to a simpler problem, we need additional definitions. A graph $G = (V, E)$ is said to be *proper* if V can be partitioned into two subsets X and Y such that for every $y \in Y$, (a) $X \cup \{y\}$ is an LFS in G and (b) $|X \cap N_G(y)| = 2$. To explicitly show this partition of V, we write $G = (X \cup Y, E)$. Hereafter, by a proper graph $G = (X \cup Y, E)$, we always mean that for every $y \in Y$, $X \cup \{y\}$ is an LFS in G and $|X \cap N_G(y)| = 2$. A *valid* MLFS in a proper graph $G = (X \cup Y, E)$ is an MLFS U in G with $X \subseteq U$. The *restricted* MLF problem is to find, given a proper graph $G = (X \cup Y, E)$, a valid MLFS in G. In the rest of this section, we describe an NC reduction from the MLF problem to the restricted MLF problem.

Algorithm 1

Input: A graph $G = (V, E)$ with n vertices.

Output: A proper graph $H = (X \cup Y, E_H)$ such that $X \cup Y \subseteq V$, $H = G[X \cup Y]$, and every valid MLFS in H is an MLFS in G.

1. Initialize X to be an MIS in G.
2. Perform the following steps twice:
   - 2.1. Compute W, the set of all $v \in V - X$ such that $X \cup \{v\}$ is an LFS in G and $|N_G(v) \cap X| = 1$.
   - 2.2. Construct a graph $G' = (W, E_{G'})$, where $E_{G'}$ consists of all $\{v_1, v_2\}$ such that $\{v_1, v_2\} \in E$ or $N_G(v_1) \cap N_G(v_2) \cap X \neq \emptyset$.
   - 2.3. Compute an MIS I in G'.
   - 2.4. Add the vertices in I to X.
3. Set Y to be the set of those $v \in V - X$ such that $X \cup \{v\}$ is an LFS in G and $|N_G(v) \cap X| = 2$.

Let X_0 be the MIS in G found at step 1. For $i = 1, 2$, let W_i, G'_i, I_i, and X_i be the contents of the variables W, G', I, and X at the end of the ith execution of the steps 2.1 through 2.4, respectively.

Lemma 1: X_2 is an LFS in G.

Proof: X_0 is an MIS in G and is hence an LFS in G. We next prove that X_1 is an LFS in G. First note that $X_1 = X_0 \cup I_1$. By steps 2.1 through 2.3, each $v \in I_1$ has degree 1 in the graph $G[X_1]$. This together with the acyclicness of $G[X_0]$ implies that $G[X_1]$ is acyclic. We claim that every $u \in X_0$ has degree at most 2 in the graph $G[X_1]$. This can be seen by noting that each $u \in X_0$ is adjacent to at most one vertex $v \in I_1$ in the graph $G[X_1]$ and that $X_0 \cup \{v\}$ is an LFS in G for all $v \in I_1$. The claim implies that X_1 is an LFS in G. A similar discussion shows that X_2 is also an LFS in G. □

Lemma 2: For all $v \in V - X_2$, $X_2 \cup \{v\}$ is not an LFS in G or $|N_G(v) \cap X_2| \geq 2$.

Proof: Since X_0 is an MIS in G, it holds that $W_2 \subseteq W_1$. From this, we can observe that $V - X_2$ is the union of $V - (W_1 \cup X_0)$, $W_1 - (W_2 \cup I_1)$, and $W_2 - I_2$ which are pairwise disjoint. Let v be an arbitrary vertex in $V - X_2$. One of the following three cases must occur.

Case 1: $v \in V - (W_1 \cup X_0)$. Then, by the definition of W_1, $X_0 \cup \{v\}$ is not an LFS in G or $|N_G(v) \cap X_0| \geq 2$. Since $X_0 \subseteq X_2$, it follows that $X_2 \cup \{v\}$ is not an LFS in G or $|N_G(v) \cap X_2| \geq 2$.

Case 2: $v \in W_1 - (W_2 \cup I_1)$. Then, by the definition of W_2, $X_1 \cup \{v\}$ is not an LFS in G or $|N_G(v) \cap X_1| \geq 2$.

Since $X_1 \subseteqq X_2$, it follows that $X_2 \cup \{v\}$ is not an LFS in G or $|N_G(v) \cap X_2| \geqq 2$.

Case 3: $v \in W_2 - I_2$. Since $v \in W_1$, $|N_G(v) \cap X_0| = 1$. Let u be the unique vertex in $N_G(v) \cap X_0$. Because $v \in W_1 - I_1$ and I_1 is an MIS in G_1', there is a $w \in I_1$ with $\{v, w\} \in E_{G_1'}$. By the construction of G_1', $\{v, w\}$ belongs to E or there is some $x \in X_0$ with $x \in N_G(v) \cap N_G(w)$. In the former case, $w \in N_G(v) \cap I_1$ and $\{u, w\} \subseteqq N_G(v) \cap X_1$, implying that $|N_G(v) \cap X_2| \geqq 2$. Thus, we may assume the latter case. Then, we have $x = u$ since $x \in X_0 \cap N_G(v) = \{u\}$. Therefore, $w \in N_{G[X_1]}(u)$. On the other hand, since $v \in W_2 - I_2$ and I_2 is an MIS in G_2', there is a $w' \in I_2$ with $\{v, w'\} \in E_{G_2'}$. By the construction of G_2', $\{v, w'\}$ belongs to E or there is some $y \in X_1$ with $y \in N_G(v) \cap N_G(w')$. In the former case, $w' \in N_G(v) \cap I_2$ and $\{u, w'\} \subseteqq N_G(v) \cap X_2$, implying that $|N_G(v) \cap X_2| \geqq 2$. Thus, we may assume the latter case. Then, we have $y = u$ since $|N_G(v) \cap X_1| = 1$ and $\{u, y\} \subseteqq X_1 \cap N_G(v)$. Therefore, $w' \in N_{G[X_2]}(u)$. Recalling that $w \in N_{G[X_1]}(u)$, we now have $\{w, w'\} \subseteqq N_{G[X_2]}(u)$. Hence, $\{v, w, w'\} \subseteqq N_{G[X_2 \cup \{v\}]}(u)$. Obviously, $v \neq w$ and $v \neq w'$. Since $w \in I_1$, $w' \in I_2$, and $I_1 \cap I_2 = \emptyset$, it follows that $w \neq w'$. Therefore, $|N_{G[X_2 \cup \{v\}]}(u)| \geqq 3$. This implies that $X_2 \cup \{v\}$ is not an LFS in G. □

Theorem 3: The MLF problem can be reduced to the restricted MLF problem in $O(\log n)$ expected time with n^2 processors on a CRCW PRAM, in $O(\log^2 n)$ time with n^4 processors on an EREW PRAM, or in $O(\log^3 n)$ time with n^2 processors on an EREW PRAM.

Proof: By Lemma 1 and Lemma 2, Algorithm 1 is clearly correct. This gives a reduction from the MLF problem to the restricted MLF problem. It remains to analyze the complexity of Algorithm 1. Steps 1 and 2.3 can be done in $O(\log n)$ expected time with n^2 processor on a CRCW PRAM [14], in $O(\log^2 n)$ time with n^4 processor on an EREW PRAM [14], or in $O(\log^3 n)$ time with n^2 processor on an EREW PRAM [9]. It is not so difficult to see that the other steps can be done in $O(\log n)$ time using n^2 processors on an EREW PRAM. This establishes the theorem. □

In the next sections, we present parallel algorithms for solving the restricted MLF problem.

3. An RNC Algorithm and Its Derandomization

3.1 Description of the Algorithm

The top-level structure of the RNC Algorithm is as follows.

Algorithm 2

Input: A proper graph $G = (X \cup Y, E)$.

Output: A valid MLFS U in G.

1. Set $U := X$ and construct a new graph $G' = (Y, E')$, where E' is the set of all edges $\{y_1, y_2\}$ such that $\{y_1, y_2\} \in E$ or $X \cap N_G(y_1) \cap N_G(y_2) \neq \emptyset$.
2. While G' is not empty, perform the following steps:
   - 2.1. Select an independent set I in G' such that each connected component of $G[U \cup I]$ contains at most one vertex of I.
   - 2.2. Add the vertices in I to U.
   - 2.3. Delete from G' all vertices y such that $y \in I$ or $U \cup \{y\}$ is not an LFS in G.
3. Output U.

An easy induction shows that Algorithm 2 correctly computes a valid MLFS of G. We need to specify how to implement the steps 2.1 and 2.3. In order to perform the two steps fast, the algorithm maintains an array R for which the following is an *invariant*: For each vertex $u \in U$, $R[u] = u$ if $d_{G[U]}(u) = 0$, $R[u] = v$ if $d_{G[U]}(u) = 1$, where v is the other vertex of degree 1 in the connected component of $G[U]$ containing u. (*Comment:* Vertices u in U with $d_{G[U]}(u) = 2$ are irrelevant to R.) It is necessary to insert a new step between steps 1 and 2 in Algorithm 2 to initialize R so that the invariant is true before the first execution of the while-loop. Since the details are trivial, we omit them.

To implement step 2.1, we need a definition and a lemma due to Luby [14]. Following [13], we say that a vertex v in a simple graph H is *good* if $\sum_{u \in N_H(v)} \frac{1}{d_H(u)} \geqq \frac{1}{3}$. The following lemma is known:

Lemma 4: [13], [14]. Given a graph H, we can compute an independent set J in $O(1)$ expected time with $n+m$ processors on a CRCW PRAM such that for each good vertex v in H, the probability that $v \in N_H(J)$ is no less than a positive constant.

To implement step 2.1, we use the following three substeps:

- 2.1.1. Compute an independent set J of G' as in Lemma 4.
- 2.1.2. In parallel, for each vertex $u \in U$ with $d_{G[U]}(u) \leqq 1$, select u if $u = R[u]$, and randomly select one of u and $R[u]$ if $u < R[u]$.
- 2.1.3. Set $I := \{y \in J \mid$ the two vertices in $U \cap N_G(y)$ are both selected at step 2.1.2$\}$.

Lemma 5: For the set I computed in step 2.1.3, each connected component of $G[U \cup I]$ contains at most one vertex of I.

Proof: Fix an arbitrary connected component C of $G[U]$. Recall that C is a path. By Algorithm 2, only the endpoint(s) of C can be adjacent to a vertex of I in G. Moreover, if C has two endpoints, then exactly one of the two is selected in step 2.1.2. Thus, by the definition of I, at most one vertex u of C can be adjacent to a vertex of I in G, and this vertex u must be an endpoint of C. To show the lemma, it suffices to show

that $|N_G(u) \cap I| = 1$. Towards a contradiction, assume that $|N_G(u) \cap I| \geq 2$. Let y_1 and y_2 be two vertices in $N_G(u) \cap I$. Note that either $u \in X$ or $u \in U - X$. In the former case, $\{y_1, y_2\}$ must be an edge in G' by step 1, contradicting that I is an independent set in G'. In the latter case, u must be in Y and hence $|N_G(u) \cap X| = 2$. Combining this with the fact that $X \subseteq U$, we see that u can not be an endpoint of C, a contradiction. This completes the proof. □

To update R and to implement step 2.3, we use the following substeps:

2.3.1. In parallel, for each $y \in I$, find the two vertices u_1 and u_2 in $N_G(y) \cap U$ and set $R[R[u_1]] := R[u_2]$ and $R[R[u_2]] := R[u_1]$.

2.3.2. Delete all vertices in I from G'.

2.3.3. In parallel, for each $u \in U$ with $\mathrm{d}_{G[U]}(u) = 2$, delete from G' all vertices in $N_G(u)$.

2.3.4. In parallel, for each vertex y in G', if the two vertices u_1 and u_2 in $N_G(y) \cap U$ satisfies that $R[u_1] = u_2$, then delete y from G'.

We claim that R is correctly updated in step 2.3.1. This simply follows from the fact that the two vertices in $N_G(y) \cap U$ both have degree at most 1 in $G[U]$.

Lemma 6: The vertices of G' deleted in step 2.3.3 or 2.3.4 are exactly those vertices y in G' such that $U \cup \{y\}$ is not an LFS in G.

Proof: It is clear that the vertices y of G' deleted in step 2.3.3 or 2.3.4 satisfy that $U \cup \{y\}$ is not an LFS in G. Let y be a vertex such that $U \cup \{y\}$ is not an LFS in G. Then, one of the following three cases must occur; (1) there is a vertex $u \in N_G(y) \cap U$ with $\mathrm{d}_{G[U \cup \{y\}]}(u) > 2$; (2) the two vertices in $N_G(y) \cap U$ are the two endpoints of a single path in $G[U]$; and (3) $\mathrm{d}_{G[U \cup \{y\}]}(y) > 2$. In the cases (1) and (2), y is deleted in step 2.3.3 and 2.3.4, respectively. We claim that the case (3) is impossible. Towards a contradiction, assume that the case (3) occurs. Then, since $|N_G(y) \cap X| = 2$, there must exist a vertex $u \in (U - X) \cap N_G(y)$. Since $U - X \subseteq Y$, u was added to U in an earlier iteration of the while-loop in Algorithm 2. In that iteration, y must have been deleted in step 2.3.3 because $\mathrm{d}_{G[U]}(u) = 2$. This contradicts that y still remains in G' in the current iteration. □

3.2 Complexity Analysis

Suppose that the while-loop in Algorithm 2 is executed t times. For $1 \leq i \leq t$, let U_i and G'_i be respectively the contents of the variables U and G' after the ith iteration, and let I_i and J_i be the two independent sets (in G'_i) computed in the ith iteration. For convenience, let U_0 and G'_0 be the contents of the variables U and G' before the first iteration. Our main task in this subsection is to prove an upper bound on the expected value of t. We begin with a helpful lemma.

Lemma 7: For every $0 \leq i \leq t-1$ and every good vertex v in G'_i, $\Pr[v \in N_{G'_i}(I_{i+1})]$ is no less than a positive constant c_1.

Proof: Fix an arbitrary good vertex v in G'_i. By Lemma 4, the probability that $v \in N_{G'_i}(J_{i+1})$ is no less than a positive constant. In other words, the probability that some neighbor of v in G'_i belongs to J_{i+1} is no less than a positive constant. On the other hand, if y is a neighbor of v and y is in J_{i+1}, then the probability that $y \in I_{i+1}$ is no less than $\frac{1}{4}$ by step 2.1.2. Therefore, it seems that the probability that some neighbor of v in G'_i belongs to I_{i+1} is no less than a positive constant. However, this intuitive inference is not correct and we need a rigorous proof. Fortunately, by step 2.1.2, we have the fact that $\Pr[y \in I_{i+1} \mid y \in J_{i+1}] \geq \frac{1}{4}$ for every vertex y in G'_i. Using this fact, we can modify the proof of Lemma 4 given in [13] to prove that $\Pr[v \in N_{G'_i}(I_{i+1})]$ is no less than a positive constant. Since the details are not so difficult, we omit them. □

We proceed to the proof of the fact that the expected value of t is $O(\log n)$. To show the fact, we use a potential function argument. Before describing our potential function, we observe that for every $0 \leq i \leq t$, each vertex y in G'_i satisfies that $N_G(y) \cap U_i$ is equal to $N_G(y) \cap X$. We denote the two vertices in $N_G(y) \cap X$ by x_1^y and x_2^y. Now, we are ready to define our potential function Φ as follows: For every $0 \leq i \leq t$,

$$\Phi(G'_i, U_i) = \sum_{\text{edge } e = \{y,z\} \text{ in } G'_i} \phi(e, U_i),$$

where $\phi(e, U_i) = (2 - \mathrm{d}_{G[U_i]}(x_1^y))(2 - \mathrm{d}_{G[U_i]}(x_2^y))(2 - \mathrm{d}_{G[U_i]}(x_1^z))(2 - \mathrm{d}_{G[U_i]}(x_2^z))$.

For a random variable Z, let $\mathcal{E}Z$ denote the expected value of Z, and let $\mathcal{E}(Z \mid B)$ denote the expected value of Z given that event B occurs.

Lemma 8: For every $0 \leq i \leq t-1$, $\mathcal{E}(\Phi(G'_i, U_i) - \Phi(G'_{i+1}, U_{i+1})) \geq c_2 \cdot \Phi(G'_i, U_i)$ for some positive constant c_2.

Proof: Fix an arbitrary i with $0 \leq i \leq t-1$. Let m'_i be the number of edges in G'_i. For each edge $e = \{y, z\}$ in G'_i, let $Z_e = \phi(e, U_{i+1})$ if e is also in G'_{i+1} and $Z_e = 0$ otherwise, and let B_e be the event that $\{y, z\} \cap N_{G'_i}(I_{i+1}) \neq \emptyset$. Then, we have

$$\Phi(G'_i, U_i) - \Phi(G'_{i+1}, U_{i+1}) = \sum_{\text{edges } e \text{ in } G'_i} (\phi(e, U_i) - Z_e).$$

Fix an arbitrary edge $e = \{y, z\}$ in G'_i. Note that $\phi(e, U_i) \geq 1$. Assume that event B_e occurs. Then, $y \in N_{G'_i}(I_{i+1})$ or $z \in N_{G'_i}(I_{i+1})$. By symmetry, we may assume that $y \in N_{G'_i}(I_{i+1})$. Let w be a vertex in I_{i+1} with $y \in N_{G'_i}(w)$. Then, by the construction of G', either $\{y, w\}$ is an edge in G or $X \cap N_G(y) \cap N_G(w) \neq \emptyset$. In the former case, the edge $\{y, w\}$ is not contained in G'_{i+1} since it must be deleted from G' in step 2.3.3 during the

$i+1$st iteration, and hence $\phi(e,U_i)-Z_e=\phi(e,U_i)\geqq 1$. In the latter case, $\{x_1^y,x_2^y\}\cap\{x_1^w,x_2^w\}\neq\emptyset$ and we can simply show that $\phi(e,U_i)-Z_e=\phi(e,U_i)-\phi(e,U_{i+1})\geqq 1$. Thus, in both cases, we have $\phi(e,U_i)-Z_e\geqq 1$. This implies that $\mathcal{E}(\phi(e,U_i)-Z_e\mid B_e)\geqq 1$. Therefore, $\mathcal{E}(\phi(e,U_i)-Z_e)\geqq\mathcal{E}(\phi(e,U_i)-Z_e\mid B_e)\Pr[B_e]\geqq\Pr[B_e]$. If in addition e is good, then by Lemma 7, $\mathcal{E}(\phi(e,U_i)-Z_e)\geqq\Pr[B_e]\geqq c_1$ for some constant $c_1>0$. Combining this with the fact that G_{i+1} is a subgraph of G_i, we now have

$$\begin{aligned}
&\mathcal{E}(\Phi(G_i',U_i)-\Phi(G_{i+1}',U_{i+1}))\\
&\quad=\mathcal{E}\Bigg(\sum_{\text{edges } e \text{ in } G_i'-G_{i+1}'}(\phi(e,U_i)-Z_e)\Bigg)\\
&\quad=\sum_{\text{edges } e \text{ in } G_i'-G_{i+1}'}\mathcal{E}(\phi(e,U_i)-Z_e)\\
&\quad\geqq\sum_{\text{good edges } e \text{ in } G_i'-G_{i+1}'}\mathcal{E}(\phi(e,U_i)-Z_e)\\
&\quad\geqq\sum_{\text{good edges } e \text{ in } G_i'-G_{i+1}'}c_1\geqq\frac{m_i'}{2}c_1.
\end{aligned}$$

The last inequality follows from the fact that at least half the edges in any simple graph are good [13], [15]. On the other hand, we have $\Phi(G_i',U_i)\leqq 16m_i'$. Thus, $\mathcal{E}(\Phi(G_i',U_i)-\Phi(G_{i+1}',U_{i+1}))\geqq\frac{c_1}{32}\Phi(G_i',U_i)$. This completes the proof. □

Lemma 9: $\mathcal{E}t=O(\log n)$.

Proof: Note that $\Phi(G_0',U_0)\leqq 16\binom{n}{2}$ and that the while-loop is iterated until $\Phi(G_i',U_i)<1$. Thus, by Lemma 8 and Theorem 1.3 in [15], we immediately have that $\mathcal{E}t\leqq\int_1^{16\binom{n}{2}}\frac{1}{c_2x}dx=O(\log n)$. This completes the proof. □

Theorem 10: An MLFS in an n-vertex graph G can be found in $O(\log n)$ expected time with n^2 processors on a CRCW PRAM.

Proof: By Theorem 3, it suffices to prove that Algorithm 2 runs in $O(\log n)$ expected time with n^2 processors. Since there are only $O(n)$ edges between X and Y in G, step 1 takes $O(\log n)$ time with n^2 processors. Thus, by Lemma 9, it remains to show that steps 2.1 through 2.3 can be implemented in $O(1)$ time with n^2 processors.

By Lemma 4, step 2.1.1 can be done in $O(1)$ time with n^2 processors. Steps 2.1.2 and 2.1.3 use no more resources than step 2.1.1. Thus, the implementation of step 2.1 takes $O(1)$ time with n^2 processors. Implementing step 2.2 in $O(1)$ time using n processors is trivial. Now, consider the implementation of step 2.3. Recall that during any iteration of the while-loop, each vertex y in G' satisfies that $N_G(y)\cap U=N_G(y)\cap X$. Hence, the algorithm may compute $N_G(y)\cap U$ before entering the while-loop, and this takes $O(\log n)$ time with n^2 processors. Therefore, step 2.3.1 can be done in $O(1)$ time with n processors. The implementation of step 2.3.2 is trivial. It is not difficult to implement step 2.3.3 in $O(1)$ time with n^2 processors. Step 2.3.4 uses no more resources than step 2.3.1. In total, the implementation of step 2.3 takes $O(1)$ time with n^2 processors. This completes the proof. □

3.3 Derandomization

In this section, we obtain an NC algorithm for the MLF problem by derandomizing the randomized algorithm in Sect. 3.1. Recall that the randomized algorithm consumes random bits in steps 2.1.1 and 2.1.2. We now consider how many random bits are sufficient. It has been shown in [13], [14] that step 2.1.1 needs only $2\log n+O(1)$ random bits. We claim that step 2.1.2 needs only 1 random bit. To see this, we replace steps 2.1.2 and 2.1.3 with the following steps:

2.1.2'. Partition J into two subsets J_1 and J_2 such that both $U\cup J_1$ and $U\cup J_2$ are LFS's in G. (*Comment:* This step is possible because each connected component of $G[U\cup J]$ is a cycle or path.)

2.1.3'. Randomly set I to be J_1 or J_2.

It is easy to see that even if we do the above replacement, the algorithm remains correct, and more importantly, Lemma 7 and Lemma 8 still hold. Step 2.1.3' needs only 1 random bit, and hence the modified algorithm uses only $2\log n+O(1)$ random bits. Therefore, we can derandomize it using standard techniques. Since the details are rather standard, we omit them. Summarizing up, we obtain the following theorem:

Theorem 11: An MLFS in an n-vertex graph G can be found in $O(\log^2 n)$ time with $O(n^4)$ processors on an EREW PRAM.

4. A More Efficient NC Algorithm

We have already obtained an NC algorithm for the MLFS problem in Sect. 3. This algorithm runs very fast. However, it uses too many processors. In this section, we present a little more efficient NC algorithm using a completely different approach.

We begin with several definitions. 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: A proper graph $G=(X\cup Y,E)$.

Output: A valid MLFS U in G.

1. Set $U := X$ and $W := Y$.
2. While $W \neq \emptyset$, perform the following steps:
   - 2.1. 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 $d_{G[U]}(u) = 1$; (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 proper subset of $\{w_1, w_2, w_3\}$ is contained in E_H 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 $d_{G[U]}(u) = 0$; (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.
   - 2.2 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.)
   - 2.3. Let $C_1, \ldots, C_k$ be the connected components of $G[U \cup S]$ that are cycles. For each $1 \leqq i \leqq k$, 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 vertex-disjoint paths from x_i to y_i in C_i both contain a vertex in $S - \{x_i, y_i\}$.
   - 2.4. For each $1 \leqq i \leqq k$, 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 $A := \cup_{1 \leqq i \leqq k}((N_G(x_i) \cup N_G(u_{i,1}) \cup N_G(u_{i,2})) \cap W) - S$ and $B := \cup_{1 \leqq i \leqq k}((N_G(y_i) \cup N_G(v_{i,1}) \cup N_G(v_{i,2})) \cap W) - S$.
   - 2.5. If $|A| \leqq |B|$, add the vertices in $S - \{x_1, \ldots, x_k\}$ to U and then set $W := \{w \in A - B : U \cup \{w\}$ is an LFS in $G\}$; otherwise, add the vertices in $S - \{y_1, \ldots, y_k\}$ to U and then set $W := \{w \in B - A : U \cup \{w\}$ is an LFS in $G\}$.
3. Output U.

Let t be the number of executions of the while-loop in Algorithm 3. In case $t = 0$, Algorithm 3 is clearly correct. Thus, we may assume $t \geqq 1$. For $1 \leqq j \leqq t$, let W_j, U_j, H_j, S_j, A_j, and B_j denote the contents of the variables W, U, H, S, A, and B 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 1, respectively. Clearly, for $1 \leqq j \leqq t$, $W_j \subsetneqq W_{j-1}$, $U_{j-1} \subsetneqq U_j$, and $U_j \subsetneqq U_{j-1} \cup S_j$.

Lemma 12: For all $0 \leqq j \leqq t$ and all $w \in W_j$, $U_j \cup \{w\}$ is an LFS in G and $|U_j \cap N_G(w)| = 2$.

Proof: This is done by induction on j. Clearly, $U_0 \cup \{w\}$ is an LFS in G and $|U_0 \cap N_G(w)| = 2$ for all $w \in W_0$. Let $j > 0$ and assume that $U_{j-1} \cup \{w\}$ is an LFS in G and $|U_{j-1} \cap N_G(w)| = 2$ for all $w \in W_{j-1}$.

Let us first prove that no vertex has degree $\geqq 3$ in the graph $G[U_{j-1} \cup S_j]$. Fix an arbitrary vertex $u \in S_j$. Then, $d_{G[U_{j-1} \cup \{u\}]}(u) = 2$ by the inductive hypothesis. Since S_j is an independent set in H_j, $N_G(u) \cap S_j = \emptyset$. Thus, $d_{G[U_{j-1} \cup S_j]}(u) = 2$. Next, fix an arbitrary vertex $u \in U_{j-1}$. By the inductive hypothesis, $d_{G[U_{j-1}]}(u) \leqq 2$. Since S_j is an independent set in H_j, the construction of H_j guarantees that there are at most $2 - d_{G[U_{j-1}]}(u)$ vertices of S_j adjacent to u in the graph $G[U_{j-1} \cup S_j]$. This implies that $d_{G[U_{j-1} \cup S_j]}(u) \leqq 2$. Therefore, no vertex has degree $\geqq 3$ in the graph $G[U_{j-1} \cup S_j]$.

By the inductive hypothesis, $G[U_{j-1} \cup S_j]$ cannot contain a cycle in which only one vertex of S_j appears. Moreover, $G[U_{j-1} \cup S_j]$ cannot contain a cycle in which two or three vertices of S_j appear because S_j is an independent set in H_j. Hence, each connected component of $G[U_{j-1} \cup S_j]$ is either a cycle containing at least 4 vertices in S_j or a path. From this and Algorithm 3, it is easy to see that U_j is an LFS in G. Let w be an arbitrary vertex in W_j. By step 2.5, $U_j \cup \{w\}$ is an LFS in G. Moreover, $|U_{j-1} \cap N_G(w)| = 2$ by the inductive hypothesis. Therefore, $|U_j \cap N_G(w)| = 2$. □

Lemma 13: U_t is an MLFS in G.

Proof: By Lemma 12, we only need to prove the maximality of U_t. Let w be an arbitrary vertex in $(X \cup Y) - U_t$. Then, $w \in W_{j-1} - (W_j \cup U_j)$ for some $1 \leqq j \leqq t$. Without loss of generality, we may assume $|A_j| \leqq |B_j|$. Then, $W_j \subsetneqq A_j - B_j$ and $U_j = U_{j-1} \cup (S_j - \{x_1, \ldots, x_k\})$, where $x_1, \ldots, x_k$ are the vertices chosen in step 2.3 during the jth iteration. Thus, one of the following three cases must occur:

Case 1: $w = x_i$ for some $1 \leqq i \leqq k$. Then, the cycle C_i is completely contained in $G[U_j \cup \{w\}]$. Thus, $U_j \cup \{w\}$ is not an LFS in G.

Case 2: $w \in B_j$. Then, $w \in N_G(y_i) \cup N_G(v_{i,1}) \cup N_G(v_{i,2})$ for some $1 \leqq i \leqq k$. Since no two vertices of S_j can be adjacent in the graph $G[U_{j-1} \cup S_j]$, the choice of x_i and y_i guarantees that none of $v_{i,1}$ and $v_{i,2}$ is a neighbor of x_i in the cycle C_i. Thus, each of y_i, $v_{i,1}$, and $v_{i,2}$ has degree 2 in the graph $G[U_j]$. This implies that at least one of y_i, $v_{i,1}$, and $v_{i,2}$ must have degree 3 in the graph $G[U_j \cup \{w\}]$. Therefore, $U_j \cup \{w\}$ is not an LFS in G.

Case 3: $w \in W_{j-1} - (S_j \cup A_j \cup B_j)$. Since S_j is an MIS in H_j, there is a hyperedge W' in H_j such that $w \in W'$ and $W' - \{w\} \subsetneqq S_j$. If $(W' - \{w\}) \cap \{x_1, \ldots, x_k\} = \emptyset$, then $U_j \cup \{w\}$ is clearly not an LFS in G by the construction of H_j. Thus, we may assume that $(W' - \{w\}) \cap \{x_1, \ldots, x_k\} \neq \emptyset$. Then, one of the following two subcases must occur.

Subcase 3.1: $|W'| = 2$. Let x_i $(1 \leqq i \leqq k)$ be the unique vertex in $W' - \{w\}$. Since W' is a hyperedge in H_j and $w \notin A_j$, $G[U_{j-1} \cup \{w, x_i\}]$ has a cycle in which both w and x_i appear. Let u be a neighbor of w in

this cycle. Clearly, $u \in U_{j-1}$. Furthermore, u must also be contained in C_i and cannot be $u_{i,1}$ or $u_{i,2}$. Thus, $d_{G[U_j]}(u) = 2$ and so $d_{G[U_j \cup \{w\}]}(u) = 3$. This implies that $U_j \cup \{w\}$ is not an LFS in G.

Subcase 3.2: $|W'| = 3$. Then, there is some x_i ($1 \leqq i \leqq k$) with $x_i \in W' - \{w\}$. Let v be the vertex in $W' - \{w, x_i\}$. Since W' is a hyperedge in H_j and $w \notin A_j$, $G[U_{j-1} \cup W']$ has a cycle C in which w, x_i, and v appear. Note that neither $\{w, v\}$ nor $\{w, x_i\}$ can be an edge in G (and thus in C) or else $W' = \{w, v, x_i\}$ could not have been a hyperedge in H_j. Let u be the neighbor of w in C such that x_i can be reached from u without traversing w or v. Clearly, $u \in U_{j-1}$. Furthermore, u must also be contained in C_i and cannot be $u_{i,1}$ or $u_{i,2}$. Thus, $d_{G[U_j]}(u) = 2$ and so $d_{G[U_j \cup \{w\}]}(u) = 3$. This implies that $U_j \cup \{w\}$ is not an LFS in G. □

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

Proof: It suffices to show that $|W_j| \leqq |W_{j-1}|/2$ for all $1 \leqq j \leqq t$. Fix an arbitrary integer j with $1 \leqq j \leqq t$. Without loss of generality, we may assume $|A_j| \leqq |B_j|$. By Algorithm 3, $W_j \subseteqq A_j - (A_j \cap B_j)$ and $A_j \cup B_j \subseteqq W_{j-1}$. Now, it is easy to see that $|W_j| \leqq |W_{j-1}|/2$. □

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

Proof: By Theorem 3 and Lemma 13, it suffices to prove that Algorithm 3 runs in $O(\log^5 n)$ time with n^3 processors on an EREW PRAM. Now, by Lemma 14, we need only to prove that steps 2.1 through 2.5 of Algorithm 3 can be done in $O(\log^4 n)$ time with n^3 processors on an EREW PRAM.

Before constructing the hypergraph H in step 2.1, Algorithm 3 first computes the connected components of $G[U]$ and the degree of each vertex in $G[U]$. This takes only $O(\log n)$ time using n processors on an EREW PRAM. Using this information and that $|U \cap N_G(w)| = 2$ for all $w \in W$, it is easy to construct H in $O(\log n)$ time with n^3 processors on an EREW PRAM. Note that H has only n^3 hyperedges. Thus, step 2.2 can be done in $O(\log^4 n)$ time with n^3 processors on an EREW PRAM[7]. Obviously, steps 2.3 through 2.5 can be done in $O(\log n)$ time with n^2 processors on an EREW PRAM. Hence, steps 2.1 through 2.5 can be done in $O(\log^4 n)$ time with n^3 processors on an EREW PRAM. □

5. Concluding Remarks

Restricting to bipartite graphs or sparse graphs (such as planar graphs), the MLF problem can be easily reduced to the MPS problem and hence has more efficient NC algorithms than Algorithm 2 and 3. Since the details are rather easy, we omit them here.

In the remainder of this section, we point out an interesting open question related to this work. We say that a bipartite graph $G = (X, Y, E)$ is *special* if every $y \in Y$ has exactly 3 neighbors (in X). Consider the following problem: Given a special bipartite graph $G = (X, Y, E)$, we wish to find a maximal subset S of Y such that $G[X \cup S]$ is acyclic. We call this problem the *maximal triangular cactus* (MTC) problem. The MTC problem can be viewed as an extension of the restricted MLF problem. We describe an interesting application of the MTC problem below.

The *maximum planar subgraph* problem is, given a graph $G = (V, E)$, to find a maximum subset F of E such that the graph (V, F) is planar. This problem has applications in circuit layout and graph drawing, but is unfortunately NP-hard. Thus, it is of interest to design approximation algorithms for this problem. Note that the maximum planar subgraph of an n-vertex graph contains at most $3n - 5$ edges and that a spanning tree has exactly $n-1$ edges. From this fact, we obtain a trivial approximation algorithm which always computes a planar subgraph whose size is at least $\frac{1}{3}$ optimal. Only recently, Călinescu et al. [2] gave the first approximation algorithm better than the trivial one.

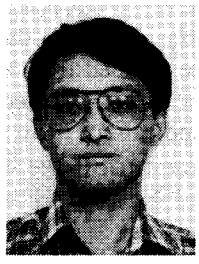
To describe Călinescu et al.'s approximation algorithm, we need several definitions. A *triangular cactus* is a graph whose cycles (if any) are triangles and such that all edges appear in some cycle. A *triangular cactus in a graph* G is a spanning subgraph of G which is a triangular cactus. A triangular cactus H in G is *maximal* if there is no other triangular cactus in G which is a supergraph of H. It is easy to see that a maximal triangular cactus in a given graph can be computed in polynomial time. We are now ready to describe Călinescu et al.'s approximation algorithm. Given a (connected) graph G, their algorithm computes a maximal triangular cactus H in G, and then computes a spanning tree T in the graph obtained from G by merging each connected component of H into a single vertex. The output of the algorithm is the union of H and T. In [2], it is shown that the size of the planar subgraph output by this approximation algorithm is at least $\frac{7}{18}$ optimal. We wish to parallelize this algorithm. To reach our goal, we only need to answer the question whether a maximal triangular cactus in a given graph can be computed in NC.

We claim that the problem of computing a maximal triangular cactus in a given graph can simply be reduced to the MTC problem. The reduction is as follows. Given a graph $G = (V, E)$, we construct a *special* bipartite graph $G' = (X, Y, E')$, where $X = V$, Y is the set of all triangles in G, and E' consists of all $\{x, y\}$ such that the vertex x appears in the triangle y. Clearly, there is a one-to-one correspondence between maximal triangular cactuses in G and maximal subsets S of Y such that $G'[X \cup S]$ is acyclic. Thus, the claim holds. Therefore, Călinescu et al.'s approximation algorithm can be parallelized if the MTC problem can be solved in NC. However, we do not know whether the MTC

problem is in NC or not. This is an open question.

References

[1] R. Anderson, "A parallel algorithm for the maximal path problem," Proc. 17th ACM Symp. on the Theory of Computing, pp.33–37, ACM, 1985.
[2] G. Călinescu, C.G. Fernandes, U. Finkler, and H. Karloff, "A better approximation algorithm for finding planar subgraphs," Proc. 7th ACM-SIAM Symp. on Discrete Algorithms, pp.16–25, ACM, 1996.
[3] Z.-Z. Chen, "A randomized NC algorithm for the maximal tree cover problem," Information Processing Letters, vol.40, pp.241–246, 1991.
[4] Z.-Z. Chen, "The maximal f-dependent set problem for planar graphs is in NC," Theoretical Computer Science, vol.143, pp.309–318, 1995.
[5] Z.-Z. Chen, "Parallel constructions of maximal path sets and applications to short superstrings," Theoretical Computer Science, vol.161, pp.1–21, 1996. A preliminary version was presented at ICALP '96.
[6] Z.-Z. Chen and X. He, "Parallel algorithms for maximal acyclic sets," Proc. Aizu International Symposium on Parallel Algorithm/Architecture Synthesis, pp.169–175, IEEE, 1995. to appear in Algorithmica.
[7] E. Dahlhaus, M. Karpinski, and P. Kelsen, "An efficient parallel algorithm for computing a maximal independent set in a hypergraph of dimension 3," Information Processing Letters, vol.42, pp.309–313, 1992.
[8] K. Diks, O. Garrido, and A. Lingas, "Parallel algorithms for finding maximal k-dependent sets and maximal f-matchings," ISA '91 Algorithms, pp.385–395, Lecture Notes in Computer Science, vol.557, Springer-Verlag, 1991.
[9] M. Goldberg and T. Spencer, "Constructing a Maximal Independent Set in Parallel," SIAM J. Disc. Math., vol.2, no.3, pp.322–328, 1989.
[10] A. Israeli and Y. Shiloach, "An improved parallel algorithm for maximal matching," Information Processing Letters, vol.22, no.57–60, 1986.
[11] R.M. Karp and A. Wigderson, "A fast parallel algorithm for the maximal independent set problem," Journal of American Computing Machinery, vol.32, no.4, pp.762–773, 1985.
[12] P. Kelsen and V. Ramachandran, "On finding minimal 2-connected subgraphs," Proc. 2nd Ann. ACM-SIAM Symp. on Discrete Algorithms, pp.178–187, ACM, 1991.
[13] D.C. Kozen, "The Design and Analysis of Algorithms," Springer-Verlag, 1992.
[14] M. Luby, "A simple parallel algorithm for the maximal independent set problem," SIAM Journal on Computing, vol.15, no.4, pp.1036–1053, 1986.
[15] R. Motwani and P. Raghavan, "Randomized Algorithms," Cambridge, 1995.
[16] D. Pearson and V.V. Vazirani, "Efficient sequential and parallel algorithms for maximal bipartite sets," Journal of Algorithms, vol.14, pp.171–179, 1993.
[17] H. Peltola, H. Soderlund, J. Tarhio, and E. Ukkonen, "Algorithms for some string matching problems arising in molecular genetics," Proc. 2nd IFIP Congress, pp.53–64, 1983.
[18] T. Shoudai and S. Miyano, "Using maximal independent sets to solve problems in parallel," Theoretical Computer Science, vol.148, no.57–65, 1995.
[19] J. Storer, "Data Compression: Methods and Theory," Computer Science Press, 1988.
[20] R. Uehara, Z.-Z. Chen, and X. He, "Fast RNC and NC algorithms for finding a maximal set of paths with an application," Proc. 2nd Ann. Intern. Computing and Combinatorics, pp.209–218, Lecture Notes in Computer Science, vol.1090, Springer-Verlag, 1996.



Ryuhei Uehara was born in Osaka Prefecture, Japan, on September 7, 1965. He received the B.E. and M.S. degrees both from the University of Electro-Communications in 1991 and 1993, respectively. He is now Assistant of the Center for Information Science of Tokyo Woman's Christians University. His current interest is in the Theory of Computational Complexity, and Algorithms and Data Structures.

Zhi-Zhong Chen was born in China, on January 9, 1964. He received the B.E. degree from the Northwest Telecommunication Engineering Institute (China) in 1985, the M.S. and Ph.D. degrees both from the University of Electro-Communications in 1989 and 1992, respectively. He is now Associate Professor of the Department of Mathematical Sciences of Tokyo Denki University. His current interest is in Parallel Algorithms and Approximation Algorithms.