数理解析研究所講究録
第 876 巻 1994 年 16-28

Finding Maximal Cycle-Free Subgraphs in Parallel

Zhi-Zhong Chen
Department of Information Engineering, Mie University, Tsu-shi, Mie 514

Abstract: The problem of finding a maximal vertex-induced subgraph without cycles in a given graph is investigated. Although the parallelizability of this problem is well doubted, it is shown here that this problem can be solved efficiently in parallel if the input graph is restricted to one whose diameter is small relative to the number of vertices in it. Moreover, the problem of finding a maximal vertex-induced subgraph without cycles of a fixed length k in a given graph is also considered. It is observed that this problem is solvable in RNC. Unfortunately, the RNC algorithm is very inefficient. Efficient NC algorithms are then given for small k's.

1 Introduction

Since Karp and Wigderson showed that the maximal independent set (MIS for short) problem is solvable in NC [16], much work has been devoted to the study of maximality problems (see [5] for a precise definition of maximality problems). Most maximality problems are defined on graphs (e.g., the MIS problem). Such a problem is usually associated with some property on graphs and is either a maximal vertex-induced subgraph (MVIS for short) problem or a maximal edge-induced subgraph (MEIS for short) problem. The MVIS (MEIS) problem associated with property π is to find a maximal subset of vertices (resp., edges) of a given graph whose induced subgraph satisfies π. Here, we are interested in only those properties π that checking π is computationally easy (say, in NC) and π is hereditary (i.e., whenever a set S of vertices or edges satisfies π, then all subsets of S satisfy π). For such a property π, the MVIS (or MEIS) problem associated with π can be solved trivially in sequential. However, it is an important open question whether every MVIS (or MEIS) problem associated with such a property is solvable in NC or RNC. So far, only specific MVIS (or MEIS) problems have been shown to be solvable in NC or RNC [2,3,4,8,12,13,14,15,17,18,20]. Many natural and important MVIS (or MEIS) problems still remain unresolved. Among them is the MVIS problem associated with the property "there is no cycle". We name this problem as the *maximal forest* (MF for short) problem. The MF problem is natural in the sense that the MEIS

problem associated with the property "there is no cycle" is the famous problem of computing a spanning forest in a given graph. Although the latter problem is known to have efficient parallel algorithms, the MF problem has not been shown to be in NC. In this paper, we prove several results about the parallelizability of the MF problem.

In this paper, we propose two parallel algorithms for the MF problem. The first algorithm is in fact designed for a more general problem: Given a graph G and a nonnegative integer l, find a maximal subset of vertices in G whose induced subgraph is a forest with diameter $\leq l$. For convenience, we call this problem the *generalized maximal forest* (GMF for short) problem. If we fix the input integer l to $n-1$ where n is the number of vertices in G, then the GMF problem becomes the MF problem. Moreover, if we fix the input integer l to 0, then the GMF problem becomes the MIS problem. The two facts guarantee that the GMF problem is natural in its own right. Our algorithm for the GMF problem runs in $O(l(T_{MIS}(n)+\log n))$ time with $P_{MIS}(n)+O(n^{2.376})$ processors on an EREW PRAM, where n is the number of vertices in the input graph and $T_{MIS}(n)$ is the time needed to find a MIS in an n-vertex graph using $P_{MIS}(n)$ processors on an EREW PRAM. This result tells us that the MF problem is in NC if the input graph is restricted to one whose diameter is small comparing with the number of vertices in it. The result also shows that certain generalized versions of the MIS problem are still in NC. The second algorithm for the MF problem runs in $O(\sqrt{n}\log^3 n)$ time with $O(n)$ processors on a given n-vertex planar graph. This algorithm is based on the fact that planar graphs have separators of small size.

As mentioned above, it seems difficult to design an efficient parallel algorithm for the MF problem. A natural question is to ask how about the MVIS problem associated with the property "there is no cycle of a fixed length k". For convenience, we here denote this problem by CYC_k. CYC_k has been investigated by Miyano in a different context [19]. Since CYC_k can be easily reduced to the problem of finding a maximal independent set in a hypergraph of dimension k, we observe that CYC_k can be solved in RNC by using a recent result of Kelsen [17]. Unfortunately, this direct use of Kelsen's result only gives us an RNC^{91} algorithm for CYC_4 and even more inefficient RNC algorithms for CYC_k when $k > 4$. In this paper, we show how to solve CYC_k more efficiently for small k's. Two results are obtained. The first is that CYC_3 can be solved in $O(\log^4 n)$ time with $O(nm)$ processors on an EWEW PRAM (n and m denote hereafter the numbers of vertices and edges in the input graph, respectively). The second result is that for $4 \leq k \leq 7$, CYC_k can be solved in poly-logarithmic time with $O(m^{k-2}+m^2 n^{\lfloor \frac{k}{2} \rfloor - 1})$ processors on an EWEW

PRAM if the input graph is restricted to a sparse one.

2 Preliminaries

Throughout this paper, a graph is a usual undirected graph without multiple edges and self-loops. Let $G = (V, E)$ be a graph. We sometimes write $V = V(G)$ and $E = E(G)$. For a vertex $v \in V$, $N_G(v)$ denotes the set $\{u : \{v, u\} \in E\}$. We denote by $dist_G(u, v)$ the distance between two vertices u, v in G. 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 $G[U]$. For $U \subseteq V$, $||U||$ denotes the number of vertices in U. By a path, we mean a simple path. The *length* of a path is the number of edges it traverses. The *diameter* of G is the length of the longest paths in G. A cycle of length k in G is called a *k-cycle*. A *maximal sugraph of G without k-cycles* is a subgraph $G[V']$ such that $G[V']$ contains no k-cycle but $G[V' \cup \{v\}]$ contains a k-cycle for every $v \in V - V'$. A *maximal forest of G with diameter $\leq l$* is a subgraph $G[V']$ such that $G[V']$ contains no path of length $l + 1$ nor a cycle but $G[V' \cup \{v\}]$ contains a path of length $l + 1$ or a cycle for every $v \in V - V'$. A *maximal forest of G* is a maximal forest of G with diameter $\leq ||V|| - 1$. G is *d-sparse* if for every $U \subseteq V$, $G[U]$ has at most $||U|| \log^d ||U||$ edges.

A *hypergraph* $H = (V, E)$ consists of a finite set V of vertices and a family E of non-empty subsets of V called *hyperedges*. The *dimension* of H is the maximum size of a hyperedge in E. Thus, a graph is a hypergraph of dimension 2. An *independent set* in H is a subset of V containing no hyperedge of H. A *maximal independent set* (MIS) in H is an independent set in H that is not properly contained in some other independent set.

As the model of computation, we choose the EREW PRAM, in which concurrent reads or concurrent writes of the same memory location are disallowed (for a discussion of the various PRAM models, see [1]).

3 Finding a maximal forest

In this section, we give two parallel algorithms for finding a maximal forest in a given graph. The first algorithm has an NC implementation if the output forest is required to have a small diameter. The second algorithm runs in sublinear time if the input graph is restricted to a planar one.

3.1 The first algorithm

We here show how to find, given a graph G and a nonnegative integer l, a maximal forest with diameter $\leq l$.

Input: A graph $G = (V, E)$ and a nonnegative integer l.
Output: A maximal subset of V whose induced subgraph is a forest of diameter $\leq l$.

1. If $l = 0$, then output a MIS of G and halt.
2. Compute a maximal subset U of V such that $G[U]$ is a forest of diameter $\leq l - 1$.
3. Set $V' = V - U$;
4. Remove from V' all vertices v such that $G[U \cup \{v\}]$ is not a forest of diameter $\leq l$.
5. Construct a graph $G' = (V', E')$ as follows: for every two vertices v_1, v_2 in V', $\{v_1, v_2\} \in E'$ iff $\{v_1, v_2\} \in E$ or there is a connected component C in $G[U]$ such that $N_G(v_1) \cap V(C) \neq \emptyset$ and $N_G(v_2) \cap V(C) \neq \emptyset$.
6. Compute a MIS I' of G' and add it to U.
7. Remove from V' all vertices in I' and all vertices $v \in V'$ such that $G[U \cup \{v\}]$ is not a forest of diameter $\leq l$.
8. Construct a graph $G'' = (V', E'')$ as follows: for every two vertices v_1, v_2 in V', $\{v_1, v_2\} \in E'$ iff $\{v_1, v_2\} \in E$ or there is a connected component C in $G[U]$ such that $N_G(v_1) \cap V(C) \neq \emptyset$, $N_G(v_2) \cap V(C) \neq \emptyset$, and $V(C) \cap I' = \emptyset$.
9. Compute a MIS I'' of G'' and add it to U.
10. Output U and halt.

In order to show the correctness of the algorithm, we need two lemmas.

Lemma 3.1 Let K be a graph and let K_1, K_2 be two vertex-disjoint subgraphs of K. Then, if K_1 and K_2 contain paths of length l and belong to the same connected component of K, then K contains a path of length $l + 1$.

Proof. Easy and thus omitted. ∎

Lemma 3.2 Let T be a tree, and let u_1, u_2 be two vertices of T such that for each $v \in V(T)$, $max\{dist_T(u_1, v), dist_T(u_2, v)\} \leq dist_T(u_1, u_2)$. Then, the path from u_1 to u_2 is one of the longest paths in T.

Proof. See Lemma 3.1 in [4]. ∎

Now we are ready to show the correctness of the algorithm. Since the algorithm is obviously correct when $l = 0$, we assume that $l > 0$. In addition to the notations used in the algorithm, let U_i be the content of the variable U at the end of step i for $5 \leq i \leq 9$, and let V'_i be the content of the variable V' at the end of step i for $3 \leq i \leq 9$. Obviously, $V'_{i+1} \subseteq V'_i$ for $3 \leq i \leq 8$, $U_5 \subseteq U_6 = U_7 = U_8 \subseteq U_9$.

Lemma 3.3 U_9 (i.e., the output of the algorithm) is a maximal subset of V whose induced subgraph is a forest with diameter $\leq l$.

Proof. We first show that $G[U_9]$ is a forest of diameter $\leq l$. Obviously, $G[U_5]$ is a forest of diameter $\leq l$ by steps $2 \sim 5$. By steps 5 and 6, each connected component of $G[U_6]$ contains only one vertex of I'. Thus, $G[U_6]$ (and hence $G[U_7]$, $G[U_8]$) is a forest of diameter $\leq l$ by step 4. We next prove that $G[U_9]$ is also a forest of diameter $\leq l$. For convenience, we distinguish the connected components of $G[U_8]$ in the following way: a connected component T of $G[U_8]$ is said to be *of type* I if $V(T) \cap I' = \emptyset$, and is said to be *of type* II otherwise. By step 2, T must contain a path of length l if T is of type II. For each vertex $v \in V'_6$, if there are two connected components of $G[U_8]$ $(= G[U_6])$, say T_1 and T_2 of type II such that $N_G(v) \cap V(T_1) \neq \emptyset$ and $N_G(v) \cap V(T_2) \neq \emptyset$, then v must be removed from V' in step 7 by Lemma 3.1. Thus, for each vertex $v \in V'_7$, there is at most one T of type II such that $N_G(v) \cap V(T) \neq \emptyset$. On the other hand, by the maximality of I', for each vertex $v \in V'_7$, there must be a T of type II such that $N_G(v) \cap V(T) \neq \emptyset$. Therefore, for each vertex $v \in V'_7$, there is exactly one T of type II such that $N_G(v) \cap V(T) \neq \emptyset$. Let K be a connected component of $G[U_9]$. Then, by step 8, K must consist of one T of type II, some $T_1, \cdots, T_r$ of type I, and some vertices in I'' via which T is connected with $T_1, \cdots, T_r$. Moreover, K clearly contains no cycle by step 8. Let u_1 and u_2 be two vertices of T such that $dist_T(u_1, u_2) = l$. By steps 7 and 8, we easily see that for each $v \in V(K)$, $max\{dist_K(u_1, v), dist_K(u_2, v)\} \leq l = dist_K(u_1, u_2)$. Thus, by Lemma 3.2, K and (hence) $G[U_9]$ contains no path of length $> l$.

Next, we show the maximality of U_9. Obviously, no vertex removed from V' in step 4 or 7 can be added to U unless $G[U]$ contains a cycle or a path of length $> l$. So we are done if V'_9 is empty. Suppose V'_9 is not empty. Let v be an arbitrary vertex in V'_9. By the maximality of I'', there is a vertex u in U_9 such that $\{v, u\} \in E$ or both $N_G(v) \cap V(T) \neq \emptyset$ and $N_G(u) \cap V(T) \neq \emptyset$ for a common T of type I. Let T_v (T_u) be the unique connected component of type II in $G[U_7]$ containing a vertex in $N_G(v)$ (resp., $N_G(u)$). If $T_v = T_u$, then $G[U_9 \cup \{u\}]$ contains a cycle. On the other hand, if $T_v \neq T_u$, then $G[U_9 \cup \{u\}]$ must contain a path of length $> l$ by Lemma 3.1. ∎

Theorem 3.4 Suppose that a MIS in a given graph with n vertices and m edges can be found in $T_{MIS}(n,m)$ time using $P_{MIS}(n,m)$ processors. Then, a maximal forest with diameter l in a given n-vertex graph G can be found in $O(l(T_{MIS}(n,n^2)+\log n))$ time using $P_{MIS}(n,n^2)+O(n^{2.376})$ processors.

Proof. We prove this theorem by an induction on the input integer l. When $l=0$, we only need to compute a MIS of G, and thus the theorem holds. Assume the theorem to be true for $l-1$ with $l \geq 1$. To show that the theorem still holds for l, it suffices to prove that all steps except step 2 of the above algorithm can be implemented in $T_{MIS}(n,n^2)+O(\log n)$ time using $P_{MIS}(n,n^2)+O(n^{2.376})$ processors by Lemma 3.3 and the inductive hypothesis.

For steps 1 and 3, their implementation is trivial. We now consider step 4. Step 4 is divided into three substeps. In the first substep, the algorithm computes the connected components of $G[U]$, i.e., computes a function $f : U \rightarrow \mathbf{N}$ such that for every two vertices u, u' in U, $f(u)=f(u')$ iff u and u' are in the same connected component of $G[U]$ ($\mathbf{N}$ is the set of nonnegative integers). Since $G[U]$ is a forest, this substep can be done in $O(\log n)$ time using $O(\frac{n}{\log n})$ processors [9]. In the second substep, the algorithm does the following: for each $v \in V'$, check whether $G[U \cup \{v\}]$ contains a cycle and remove v from U if so. This substep can be done in $O(\log n)$ time with $O(n+m)$ processors by using the function f and a sorting procedure [6]. In the third substep, the algorithm computes, for each connected component C of $G[U]$ and each $u \in V(C)$, $diam_C(u) = max\{dist_C(u,u') : u' \in V(C)\}$. This substep can be done in $O(\log n)$ time using $O(\frac{n}{\log n})$ processors [4]. In the fourth substep, the algorithm does the following: for each v still remaining in V', check whether $G[U \cup \{v\}]$ contains a path of length $l+1$ and remove v from V' if so. Clearly, $G[U \cup \{v\}]$ contains a path of length $l+1$ iff there is a vertex $u \in U$ such that $\{v,u\} \in E$ and $diam_{C_u}(u)+1 \geq l+1$ or there are two vertices u_1, u_2 in U such that $\{v,u_1\} \in E$, $\{v,u_2\} \in E$, and $diam_{C_{u_1}}(u_1)+diam_{C_{u_2}}(u_2)+2 \geq l+1$, where C_u, C_{u_1}, and C_{u_2} are the connected components of $G[U]$ containing u, u_1, and u_2, respectively. Thus, the fourth substep can be done in $O(\log n)$ time with $O(n+m)$ processors by using the data computed in the third substep and a sorting procedure [6]. Therefore, step 4 can be done in $O(\log n)$ time with $O(n+m)$ processors. Almost the same discussions work for step 7.

We proceed to consider step 5. Step 5 is divided into three substeps. Let C_1, C_2, $\cdots$, C_k be the connected components of $G[U]$. Note that C_i's are available from step 4. In the first substep, the algorithm constructs the adjacency matrix M_K of a graph K defined as follows:

(a) $V(K) = V' \cup \{w_1, \cdots, w_k\}$, where w_i's are new vertices not in G;

(b) $E(K) = \{\{w_i, v\} : 1 \leq i \leq k,\ v \in V'$, and there is a vertex u in C_i such that $\{u, v\} \in E\}$.

This substep can easily be done in $O(\log n)$ time with $O(n^2)$ processors. Note that K is a bipartite graph. In the second substep, the algorithm computes $M_K \times M_K$. This substep can be done in $O(\log n)$ time with $O(n^{2.376})$ processors by using boolean matrix multiplication [7]. Let M'_K be the matrix obtained from M_K by removing those rows and columns corresponding to w_i's. In the third substep, the algorithm computes the desired graph G' as follows: (a) first initialize G' as the graph specified by M'_K; (b) for every two vertices v_1, $v_2 \in V'$, add the edge $\{v_1, v_2\}$ to G' iff $\{v_1, v_2\}$ is an edge in G. This substep can be done in $O(\log n)$ time with $O(n^2)$ processors. Therefore, step 5 can be done in $O(\log n)$ time with $O(n^{2.376})$ processors. Almost the same discussions work for step 8.

Steps 6 and 9 can easily be done in $T_{MIS}(n, n^2)$ time with $P_{MIS}(n, n^2)$ processors.

By the discussions in the above, the theorem follows. ∎

Since finding a MIS in a graph is in NC [12,13,16,18], we have the following two corollaries.

Corollary 3.5 If $l = O(\log^k n)$ for some constant k, then a maximal forest of diameter $\leq l$ in a given n-vertex graph can be found in NC.

Corollary 3.6 If $l = O(n^\alpha)$ for some constant $\alpha < 1$, then a maximal forest of diameter $\leq l$ in a given n-vertex graph can be found in sublinear time with $O(n^{2.376})$ processors.

Since the MIS problem corresponds to the case where $l = 0$, the above two corollaries show that certain generalizations of the MIS problem are still parallelizable. The above two corollaries also show that the problem of finding a maximal forest is parallelizable if the input graph G has a small diameter relative to the number of vertices in G.

3.2 The second algorithm

We here show how to find a maximal forest in a planar graph. The tool used here is that of small separators. A *separator* of an n-vertex connected graph G is a subset S of $V(G)$ such that when all vertices in S are removed from G, the resulting graph has no connected component with more than $\frac{2n}{3}$ vertices. A separator S is *small* if

the size of S is $O(\sqrt{n})$.

Input: A planar graph $G = (V, E)$.
Output: A maximal subset of V whose induced subgraph is a forest.

1. Find a small separator S of G.
2. Remove all vertices of S from G.
3. Set $U = \emptyset$.
4. In parallel, for each connected component C of G, find a maximal forest of C and add it to U.
5. In sequential, for each vertex v in G, check whether adding v to U results in a cycle, and add it to U if not so.
6. Output U and halt.

Theorem 3.7 A maximal forest of an n-vertex planar graph can be found in $O(\sqrt{n}\log^3 n)$ time with $O(n)$ processors.

Proof. The correctness of the algorithm is obvious. We assume that the input graph is given by its adjacency list representation. According to [10], step 1 can be implememted in $O(\sqrt{n}\log^2 n)$ time using $O(\frac{\sqrt{n}}{\log n})$ processors. Other steps can clearly be done in $O(\sqrt{n}\log^2 n)$ time with $O(n)$ processors. Thus, the theorem follows. ∎

4 Finding a maximal subgraph without k-cycles

We here consider how to find a maximal subgraph without k-cycles for some constant $k \geq 3$.

Proposition 4.1 For any constant $k \geq 3$, finding a maximal subgraph without k-cycles is in RNC.

Proof. Consider a hypergraph H defined as follows: $V(H) = V(G)$ and $E(H)$ is the family of those subsets U of $V(G)$ that the vertiecs in U consist of a k-cycle in G. Then, a MIS in H gives us a maximal subgraph without k-cycles in G. Since k is a constant, H can be constructed in poly-logarithmic time with polynomial number of processors. Thus, the problem of finding a maximal subgraph without k-cycles is NC-reducible to the problem of finding a MIS in a hypergraph of dimension k. Since the latter is known to be in RNC [17], the proposition follows. ∎

Here we note that Kelsen's algorithm [17] is very inefficient, because its time complexity heavily depends on k. Even for small k (say $k = 4$), Kelsen's algorithm

only says that finding a MIS in a hypergraph having dimension 4 is in RNC^{91}! So Kelsen's algorithm may not be called "an RNC algorithm". Below, we consider several special cases where a maximal subgraph without k-cycles can be found in NC.

Theorem 4.2 A maximal subgraph without 3-cycles in a graph G with n vertices and m edges can be found in $O(\log^4 n)$ time with $O(mn)$ processors.

Proof. The strategy is the same as that used in the proof of Proposition 4.1. The main difficulty is to construct the graph H efficiently. It is straightforward to construct H in $O(1)$ time using $O(m^3)$ processors. We here describe how to construct H in $O(\log n)$ time using only $O(mn)$ processors. The first step for the construction of H goes as follows: for each edge $\{u, v\} \in E(G)$, construct a list L_{uv} that consists of the vertices adjacent to u or v in G. This step can be done in $O(\log n)$ time with $O(mn)$ processors. The second step goes as follows: for each edge $\{u, v\} \in E(G)$, sort L_{uv} to find out those vertices w that appear in L_{uv} twice (note that $\{u, v, w\}$ must consist of a 3-cycle). Since sorting an n-element list can be done in $O(\log n)$ time with n processors [6], the second step can be done in $O(\log n)$ time with $O(mn)$ processors. Thus, H can be constructed in $O(\log n)$ time with $O(mn)$ processors. Since H has n vertices and at most $O(nm)$ hyperedges, a MIS in H can be found in $O(\log^4 n)$ time with $O(nm)$ processors [8]. Therefore, a maximal subgraph without 3-cycles in G can be found in $O(\log^4 n)$ time with $O(nm)$ processors. ∎

Theorem 4.3 Let $4 \leq k \leq 7$ and $G = (V, E)$ be a d-sparse graph with n vertices and m edges. Then, a maximal subgraph without k-cycles in G can be found in $O(\log^{4+d} n)$ time with $O(m^{k-2} + n^{k'-1}m^2)$ processors, where $k' = \lfloor \frac{k}{2} \rfloor$.

Proof. Consider the following algorithm for finding a maximal subgraph without k-cycles:

1. Set $V' = V$ and $U = \emptyset$.
2. Find an independent set I of size $\frac{\|V'\|^2}{32m'+2\|V'\|}$ in $G[V']$, where m' is the number of edges in $G[V']$.
3. Construct a hypergraph H as follows: $V(H) = I$ and $E(H)$ is the family of all subsets I' of I such that $G[U \cup I']$ has a k-cycle with all vertices of I' on it.
4. Find a MIS in H and add it to U.
5. Remove from V' all vertices in I and all vertices v such that $G[U \cup \{v\}]$ has a k-cycle.

6. If $V' = \emptyset$, then output U and halt; otherwise, goto step 2.

It is obvious that when the above algorithm terminates, $G[U]$ is a maximal subgraph without k-cycles. We next consider how to implement each step above. The implementation of step 1 is trivial. According to [11], step 2 can be done in $O(\log^3 n)$ time with $O(n+m)$ processors. A straightforward implementation of step 3 in poly-logarithmic time needs $\Omega(m^k)$ processors. The next lemma shows how to implement step 3 in $O(\log n)$ time using only $O(m^{k-2} + n^{k'-1}m^2)$ processors. Since I is an independent set in G, each hyperedge of H consists of at most k' vertices. Thus, the dimension of H is at most k' and hence H has at most $O(n^{k'})$ hyperedges. By noting that $k' \leq 3$, we have that step 4 can be done in $O(\log^4 n)$ time with $O(n^{k'})$ processors [8]. The resources needed for step 5 are no more than that for step 3. The implementation of step 6 is trivial. Since G is d-sparse, the size of the independent set I found in step 2 is $\Omega(\frac{||V'||}{\log^d ||V'||})$. By noting that all vertices in I are removed from V' in step 6, we have that steps $2 \sim 6$ are executed at most $O(\log^d n)$ times. Therefore, the above algorithm runs in $O(\log^{d+4} n)$ time using $O(m^{k-2} + n^{k'-1}m^2)$ processors. ∎

Lemma 4.4 Let $G = (V, E)$ be a graph with n vertices and m edges, and let I be an independent set of G such that $G[V - I]$ contains no k-cycle. Then, all subsets U of I such that $G[(V - I) \cup U]$ has a k-cycle with all vertices of U on it can be found in $O(\log n)$ time with $O(m^{k-2} + n^{k'-1}m^2)$ processors, where $k' = \lfloor \frac{k}{2} \rfloor$.

Proof. Consider the following steps for finding all subsets U of I such that $G[(V - I) \cup U]$ has a k-cycle with all vertices of U on it:

1. Construct a list L consisting of all paths P in G such that P is of length $k - 2$ and P's two endpoints are both in $V - I$.
2. In parallel, for each path P in L, remove from P all intermediate vertices $v \in V - I$.
3. Sort L to find all subsets I' of I such that some P in L contains all vertices in I' but contains no vertex in $I - I'$.
4. Sort L to construct, for each subset I' of I found in step 3, a list $L_{I'}$ consisting of all pairs $\langle v_1, v_2 \rangle$ of vertices in $V - I$ such that some P in L consists of v_1, v_2, and the vertices in I'.
5. In parallel, for each subset I' of I found in step 3, sort $L_{I'}$ to remove duplicates from $L_{I'}$ (that is, if a pair appears in the original $L_{I'}$ twice or more, then it appears exactly once in the resulting $L_{I'}$).

6. Construct a list L_G consisting of all triples $\langle v_1, v_2, u\rangle$ such that $u \in I$, $v_1 \neq v_2$, $v_1 \in N_G(u)$, and $v_2 \in N_G(u)$.
7. In parallel, for each subset I' of I found in step 3, first merge $L_{I'}$ with L_G, then sort the resulting list by the vertices in $V - I$ to find all vertices $u \in I$ such that there is some triple $\langle v_1, v_2, u\rangle$ in L_G with $\langle v_1, v_2\rangle \in L_{I'}$, and finally output $I' \cup \{u\}$ for all u just found.

It is easy to see that after the above steps, we get all the subsets U of I such that $G[(V - I) \cup U]$ has a k-cycle with all vertices of U on it. We next consider the resources needed by the above steps. It is easy to implement step 1 in $O(1)$ time with $O(m^{k-2})$ processors. Step 2 runs in $O(\log n)$ time with $O(m^{k-2})$ processors. Using a sorting procedure, step 3 can be done in $O(\log n)$ time with $O(m^{k-2})$ processors [6]. The resources need by steps 4 and 5 are the same as that needed by step 3. Step 6 can be done in $O(1)$ time with $O(m^2)$ processors. Since I is an independent set in G, each subset I' of I found in step 3 has size $k' - 1$. Thus, the number of subsets I' of I found in step 3 is at most $n^{k'-1}$. Hence by using a sorting procedure, step 7 can be done in $O(\log n)$ time with $O(n^{k'-1}m^2)$ processors [6]. Therefore, the theorem holds. ∎

References

[1] S.G. Akl, *The Design and Analysis of Parallel Algorithms* (Prentice-Hall, Englewood Cliffs, NJ, 1989).

[2] R. Anderson, A Parallel Algorithm for the Maximal Path Problem, *Proc. 17th ACM Symposium on Theory of Computing* (1985) 33-37.

[3] Z.-Z. Chen, A Randomized NC Algorithm for the Maximal Tree Cover Problem, *Inform. Process. Lett.* **40** (1991) 241-246.

[4] Z.-Z. Chen, A Simple Parallel Algorithm for Computing the Diameters of All Vertices in a Tree and Its Application, *Inform. Process. Lett.* **42** (1992) 243-248.

[5] Z.-Z. Chen and S. Toda, The Complexity of Selecting Maximal Solutions, *Proc. 8th IEEE Conference on Structure in Complexity Theory* (1993) 313-325.

[6] R. Cole, Parallel Merge Sort, *SIAM J. Comput.* **17** (1988) 770-785.

[7] D. Coppersmith and S. Winograd, Maxtrix Multiplication via Arithmetic Progressions, *Proc. 19th ACM Symposium on Theory of Computing* (1987) 1-6.

[8] E. Dahlhaus, M. Karpinski and P. Kelsen, An Efficient Parallel Algorithm for Computing a Maximal Independnet Set in a Hypergraph of Dimension 3, *Inform. Process. Lett.* **42** (1992) 309-313.

[9] H. Gazit, Optimal EREW Parallel Algorithms for Connectivity, Ear Decomposition and st-Numbering of Planar Graphs, *Proc. 5th IEEE International Parallel Processing Symposium* (1991) 84-91.

[10] H. Gazit and G.L. Miller, A Parallel Algorithm for Finding a Separator in Planar Graphs, *Proc. 28th IEEE Symposium on Foundations of Computer Science* (1987) 238-248.

[11] M. Goldberg, Parallel Algorithms for Three Graph Problems, *Congressus Numerantium* (1986) 111-121.

[12] M. Goldberg and T. Spencer, A New Parallel Algorithm for the Maximal Independent Set Problem, *SIAM J. Comput.* **18** (1989) 419-427.

[13] M. Goldberg and T. Spencer, Constructing a Maximal Independent Set in Parallel, *SIAM J. Disc. Math.* **2** (1989) 322-328.

[14] A. Israeli and Y. Shiloach, An Improved Maximal Matching Parallel Algorithm, *Inform. Process. Lett.* **22** (1986) 57-60.

[15] A. Israeli and A. Itai, A Fast and Simple Randomized Parallel Algorithm for Maximal Matching, *Inform. Process. Lett.* **22** (1986) 77-80.

[16] R.M. Karp and A. Wigderson, A Fast Parallel Algorithm for the Maximal Independent Set Problem, *Journal of ACM* **32** (1985) 762-773.

[17] P. Kelsen, On the Parallel Complexity of Computing a Maximal Independent Set in a Hypergraph, *Proc. 24th ACM Symposium on Theory of Computing* (1992) 339-350.

[18] M. Luby, A Simple Parallel Algorithm for the Maximal Independent Set Problem, *SIAM J. Comput.* **15** (1986) 1036-1053.

[19] S. Miyano, The Lexicographically First Maximal Subgraph Problems: P-Completeness and NC Algorithms, *Math. Systems Theory* **22** (1989) 47-73.

[20] T. Shoudai and S. Miyano, Using Maximal Independent Sets to Solve Problems in Parallel, *Proc. 17th International Workshop on Graph-Theoretic Concepts in Computer Science*, Lecture Notes in Computer Science **570** (1991) 126-134.