ELSEVIER

Discrete Applied Mathematics 78 (1997) 1–16

DISCRETE APPLIED MATHEMATICS

Efficient computation of implicit representations of sparse graphs ☆

Srinivasa R. Arikati [a], Anil Maheshwari [b,1] Christos D. Zaroliagis [a,*]

[a] *Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany*
[b] *School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6*

Received 1 June 1995; revised 26 July 1996

Abstract

The problem of finding an implicit representation for a graph such that vertex adjacency can be tested quickly is fundamental to all graph algorithms. In particular, it is possible to represent sparse graphs on n vertices using $O(n)$ space such that vertex adjacency is tested in $O(1)$ time. We show here how to construct such a representation efficiently by providing simple and optimal algorithms, both in a sequential and a parallel setting. Our sequential algorithm runs in $O(n)$ time. The parallel algorithm runs in $O(\log n)$ time using $O(n/\log n)$ CRCW PRAM processors, or in $O(\log n \log^* n)$ time using $O(n/\log n \log^* n)$ EREW PRAM processors. Previous results for this problem are based on matroid partitioning and thus have a high complexity.

Keywords: Implicit representation; Sparse graphs; Arboricity; Graph algorithms; Parallel computation

1. Introduction

A fundamental data structuring question in the design of efficient graph algorithms is how to represent a graph in memory using as little space as possible, so that given any two vertices we can test their adjacency quickly [12, 21]. Following [12, 21], we say that a class of graphs has an *implicit representation* if there exists a constant β such that for every n-vertex, m-edge graph G in the class, there is a labeling of the vertices with $\beta\lceil\log n\rceil$-bits each that allows us to decide vertex adjacency in $O(1)$ time. Implicit representation eliminates the need for an adjacency matrix. In the adjacency matrix representation of G adjacency can be tested in $O(1)$ time, but n^2 bits are required even for the case where G is *sparse* (i.e. $m = O(n)$). On the other hand, a representation of G using adjacency lists requires $m\lceil\log n\rceil$ bits, but the test for adjacency takes $O(\log n)$ time.

☆ This work was partially supported by the EU ESPRIT Basic Research Action No. 7141 (ALCOM II).
* Corresponding author. E-mail: zaro@mpi-sb.mpg.de.
[1] This research was done while the author was with the Max-Planck-Institut für Informatik.

0166-218X/97/$17.00 © 1997 Elsevier Science B.V. All rights reserved
PII S0166-218X(97)00007-3

An alternative characterization of sparse graphs is given through a graph-parameter called *arboricity*. The arboricity of a graph G is defined as $\max_J\{|E(J)|/(|V(J)|-1)\}$, where J is any subgraph of G with $|V(J)|$ vertices and $|E(J)|$ edges. Graphs of bounded arboricity are called *sparse*. As observed in [12], an implicit representation can be computed by partitioning the edges of G into edge-disjoint forests, or alternatively, by coloring the edges of G with k colors such that there is no monochromatic cycle. If G has this latter property, we say that it is *k-forest colorable*. It follows from a theorem of Nash–Williams [16, 17] that if G has arboricity c, then G is c-forest colorable, and consequently that G has an implicit representation of $cn\lceil\log n\rceil$ bits.[2] In such a case, G is said to have an *optimal* implicit representation.

In this paper, we are concerned with the efficient computation of optimal implicit representations of sparse graphs. The known sequential and parallel algorithms [5, 15] for obtaining an optimal implicit representation are based on involved techniques such as Edmonds' results on matroid partitioning [1]. In [5], an efficient matroid partitioning algorithm results in the computation of a c-forest coloring of a graph with arboricity c. (For sparse graphs, the algorithm runs in $O(n^{1.5}\sqrt{\log n})$ time.) Similarly, the algorithms in [15] for matroid union and intersection result in a randomized parallel algorithm for finding a c-forest coloring of graphs with arboricity c. (The algorithm runs in $O(\log^3 n)$ time using $O(n^{4.5})$ processors on a randomized CREW PRAM.) Planar graphs, an important case of sparse graphs with $c \leqslant 3$, have received a considerable amount of attention [4, 12, 19].

An alternative way to generate the implicit representation of a graph is proposed in [21] (Theorem 1.8). If G has treewidth t, then it has an implicit representation of $tn\lceil\log n\rceil$ bits. Note that this approach is not efficient in general, since there exist sparse graphs of small arboricity but of large treewidth. For example, planar graphs may have treewidth $\Theta(\sqrt{n})$.

The main contribution of this paper is twofold. First, we provide optimal sequential and parallel algorithms for obtaining an almost optimal implicit representation of a sparse graph, when its arboricity c is known. Our results and their comparison with previous work are summarized in Table 1. Note that several important subclasses of sparse graphs are of known arboricity, for example, planar graphs ($c \leqslant 3$), graphs of genus $o(n)$ ($c \leqslant 4$), graphs of bounded degree d ($c \leqslant \lfloor d/2\rfloor+1$) and graphs of bounded treewidth t ($c \leqslant t$).

The second contribution is based on the observation that the results in Table 1 require a priori the knowledge of the arboricity of the input graph. However, the known algorithms for computing the exact value of the arboricity (when nothing else is known about the graph) are based on matroid theory: a sequential algorithm [5] and a randomized parallel algorithm [15]. We also present here simple and optimal algorithms, including a deterministic parallel algorithm, to compute a 2-approximation for arboricity (i.e. an approximation which can be at most twice the exact value). Moreover, this

[2] Note that other authors, see e.g. [12, 21], refer to this number of bits as $(c+1)n\lceil\log n\rceil$ due to a slightly different storage of the implicit representation they use.

Table 1
Our results and their comparison with previous work, for sparse graphs of known arboricity c. The sequential (resp. parallel) previous results for planar graphs are due to [19] (resp. [4]). The sequential (resp. parallel) previous results for sparse graphs are due to [5] (resp. [15])

Implicit representation		Previous results for planar graphs	Previous results for sparse graphs	Results of this paper
Number of bits		$3n\lceil\log n\rceil$	$cn\lceil\log n\rceil$	$(c+2)n\lceil\log n\rceil$‡
Sequential time		$O(n)$	$O(n^{1.5}\sqrt{\log n})$	$O(n)$
CRCW PRAM	Parallel Time	$O(\log n \log\log n)$	$O(\log^3 n)$†	$O(\log n)$
	Number of Processors	$O(n/\log n \log\log n)$	$O(n^{4.5})$†	$O(n/\log n)$
EREW PRAM	Parallel Time	$O(\log^2 n \log\log n)$	$O(\log^4 n)$†	$O(\log n \log^* n)$
	Number of Processors	$O(n/\log n \log\log n)$	$O(n^{4.5})$†	$O(n/\log n \log^* n)$

† These results are for randomized PRAMs.
‡ For planar graphs $c \leqslant 3$.

Table 2
Our results and their comparison with previous work, for sparse graphs of unknown arboricity c. The sequential (resp. parallel) previous results are due to [5] (resp. [15]).

Implicit representation		Sparse graphs of unknown arboricity c	
		Previous results	This paper
Number of bits		$cn\lceil\log n\rceil$	$(c+2)n\lceil\log n\rceil$
Sequential time		$O(n^{1.5}\sqrt{\log n})$	$O(n)$
CRCW PRAM	Parallel time	$O(\log^3 n)$†	$O(\log^2 n/\log\log n)$
	Number of Processors	$O(n^{4.5})$†	$O(n\log\log n/\log^2 n)$
EREW PRAM	Parallel time	$O(\log^4 n)$†	$O(\log^2 n)$
	Number of processors	$O(n^{4.5})$†	$O(n/\log^2 n)$

† These results are for randomized PRAMs.

approximation leads to an implicit representation that needs the same amount of space as required by the implicit representation computed using the exact value for arboricity. Our results and their comparison with previous work are summarized on Table 2.

Our results are achieved by simple and rather intuitive techniques compared with those used in [1, 4, 5, 15, 19] and moreover, our algorithms are easy to implement. Also, our results extend to the k-forest coloring problem which is of independent

interest since it is a fundamental problem in the design of fault-tolerant communication networks [10], analysis of electric networks [9, 18] and the study of rigidity of structures [13].

The paper is organized as follows. In Section 2 we show the reduction of the problem of finding an implicit representation of a sparse graph G to that of finding a forest coloring of G. The latter problem is solved in Section 3, under the assumption that the arboricity of G is known. In Section 4 we show how a 2-approximation for the arboricity of a graph can be found, when the exact value of its arboricity is not known, and from that how a forest coloring can be computed. In Section 5, we present some extensions of our work and conclude with final remarks.

2. Implicit representation and forest coloring

In this section, we show how an implicit representation of a sparse graph with arboricity c is computed and adjacency queries are answered. We will first discuss the case where a c-forest coloring of G is also given (such a forest coloring can be computed from the algorithms in [5, 15]). We will then show how a k-forest coloring, for $k = O(c)$, can be used to compute an almost optimal implicit representation. The computation of such a k-forest coloring will be discussed in Section 3 (if c is known) and in Section 4 (if c is not known). We begin with some preliminaries.

2.1. Preliminaries

Our model of parallel computation is the well-known PRAM [11]. A PRAM employs a number of processors all of which operate synchronously and have access to a common memory. We shall use here two variants: the EREW PRAM (where simultaneous access to the same memory location by more than one processor is not allowed) and the CRCW PRAM (which allows concurrent access to the same memory location by more than one processor; in the case of concurrent writing one such processor succeeds arbitrarily).

Throughout the paper, $G = (V,E)$, $|V| = n$, $|E| = m$, denotes a simple undirected graph. The vertices of G are given distinct labels $1, 2, \ldots, n$, and, unless stated otherwise, v_i refers to the vertex with label i. We assume that G is given in the standard form of doubly-linked adjacency lists. This means that for each neighbor u of a vertex v, there exists one entry for u in the adjacency list of v. (*Remark:* Only for our EREW PRAM algorithms, we will further assume that the adjacency lists are provided with the so-called cross-links: the entry for u in the adjacency list of vertex v is provided, in addition to its identification, with a pointer to v's entry in the adjacency list of u. For details, see e.g. [7, 8].)

Many times throughout the paper, we will need to perform parallel prefix computations on adjacency lists. Note that performing a prefix computation on a list (instead

of an array) does not cause a problem, since a list of size p can be converted into an array in $O(\log p)$ time using $O(p/\log p)$ EREW PRAM processors [11]. Hence, we shall assume from now on that every adjacency list L_v, for $v \in G$, has been converted into its associated array $A(L_v)$ and we shall not make any distinction between L_v and $A(L_v)$ when we refer to the adjacency list of v.

2.2. Computing the implicit representation

We first show how to compute an implicit representation of G, if G is a tree. Choose any vertex r and root G at r. The data structure is an array $P(v)$ for all $v \in V$, where $P(v)$ is the parent of v ($P(r) = \emptyset$). The number of bits needed to store P is $n\lceil \log n \rceil$. Two vertices u and v are adjacent in G iff either $P(v) = u$ or $P(u) = v$. Hence, the adjacency test can be done in constant time. Clearly, the above method works if G is a forest also.

Suppose a c-forest coloring of a sparse graph G is given, where c is the arboricity of G. To compute an implicit representation of G, root all forests. The data structure is an $n \times c$ array P, where $P(v,i)$ is the parent of v in the ith forest. The number of bits needed to store P is $cn\lceil \log n \rceil$. Two vertices u and v are adjacent in G iff, for some i, either $u = P(v,i)$ or $v = P(u,i)$. For a given pair of vertices this test takes $O(c) = O(1)$ time, since c is constant.

Lemma 1. *Given a c-forest coloring of an n-vertex sparse graph G with arboricity c, an (optimal) implicit representation of $cn\lceil \log n \rceil$ bits can be computed either in $O(n)$ sequential time, or in $O(\log n)$ parallel time using $O(n/\log n)$ EREW PRAM processors.*

Proof. As shown above, the array P provides an implicit representation of G. The basic steps for computing P involve: rooting a tree and computing the parent of each vertex. Both these steps can be implemented in $O(n)$ sequential time, or in $O(\log n)$ time with $O(n/\log n)$ EREW PRAM processors using standard techniques (see e.g. [11, Ch. 3]). □

We now discuss the computation of an implicit representation of G when a k-forest coloring of G is given, where k is a constant approximation for c. If we use the previous approach and compute an $n \times k$ array P, we need $kn\lceil \log n \rceil$ bits to store P. However, we can do better than this by following a different approach to reduce the number of bits.

Our data structure consists of two arrays P and Q: P is an array of length m and Q of length n. In the array P we store first the parents of v_1, then the parents of v_2, and so on. $Q(i)$ indicates the position in P where the parents of v_i begin (if $Q(i) = Q(i+1)$ or $Q(i) > m$, then v_i has no parents). The implementation is presented in Algorithm 1.

Algorithm 1. Computation of implicit representation.

Input: A graph $G = (V,E)$, $|V| = n$, with a k-forest coloring.
Output: An implicit representation of G.

```
(1) ℓ := 1;
(2) for i := 1 to n do
    (a) Q(i) := ℓ;
    (b) for j := 1 to k do
        (i) if v_i is not a root in the jth forest then
        (ii)        P(ℓ) := parent of v_i in the jth forest;
        (iii)       ℓ := ℓ + 1;
            fi
        od
    od
```

Observe that each edge of G is represented exactly once in P. Hence, to store P we need $m\lceil \log n \rceil$ bits and to store Q we need $n\lceil \log m \rceil$ bits. Since $m \leqslant c(n-1)$ (because G has arboricity c), the total number of bits required is at most $c(n-1)\lceil \log n \rceil + n\lceil \log n \rceil + n\lceil \log c \rceil \leqslant (c+2)n\lceil \log n \rceil$.

Vertex v_i is a parent of v_j iff $v_i = P(\ell)$ for some $Q(j) \leqslant \ell \leqslant Q(j+1) - 1$. These two vertices are adjacent in G iff one of them is a parent of the other. Since $Q(j+1) - Q(j) \leqslant k$ for all j, the adjacency test takes $O(k) = O(1)$ time.

Lemma 2. *Given a k-forest coloring of an n-vertex sparse graph G with arboricity c, where $k = O(c)$, an implicit representation of $(c+2)n\lceil \log n \rceil$ bits (henceforth* almost optimal implicit representation*) can be computed either in $O(n)$ sequential time, or in $O(\log n)$ parallel time using $O(n/\log n)$* EREW PRAM *processors.*

Proof. As shown above, arrays P and Q provide an implicit representation of G. The sequential time bound follows immediately by Algorithm 1. We derive the parallel complexity bounds as follows. Let $v_1, v_2, \ldots, v_n$ be the vertices of G. From the k-forest coloring, we create (temporarily) an $n \times k$ array P'. In this array we store, for each vertex v_i, its parents in the k forests in the same way as we did in the proof of Lemma 1; i.e. first the parents of v_1, then the parents of v_2, etc. This is done by associating a processor with vertex v_i in forest ℓ, $1 \leqslant \ell \leqslant k$. Then, this processor copies the parent of v_i in the ℓth forest into the array position $P'[(i-1)k + \ell]$. Since every edge of G is represented only once in the k-forest coloring, some of the entries of P' are empty. We remove all the empty entries by performing a parallel prefix computation on P'. The resulting array is the required array P. It is easy to see that having P and by performing another prefix summation on it, we can construct the array Q. Since prefix sums in an array of size p can be computed in $O(\log p)$ time using $O(p/\log p)$ EREW PRAM processors [11], the required implicit representation can be achieved within the stated complexity bounds. □

We have shown that implicit representation can be computed using forest coloring. Thus, for the rest of the paper, we will be concerned with the forest coloring problem.

3. Forest coloring with known arboricity

In this section we present algorithms for computing optimally forest colorings of sparse graphs when the arboricity of the graph is known. It then follows by Lemma 2 that this leads to an optimal computation of an almost optimal implicit representation. We begin with some useful technical lemmas.

Lemma 3. *Suppose the vertices of a graph G can be ordered as $v_1, v_2, \ldots, v_n$ such that each vertex v_i has at most k neighbors before it (i.e., among $v_1, \ldots, v_{i-1}$). Then, G is k-forest colorable.*

Proof. We will use induction on i. The basis, $i = 1$, is trivial. Assume that the subgraph of G induced by $v_1, \ldots, v_{i-1}$ can be colored using k colors, say integers from 1 up to k. Let the neighbors of v_i that come before it, be $u_1, \ldots, u_p$, where $p \leqslant k$. For each $1 \leqslant j \leqslant p$, color the edge (v_i, u_j) with color j. □

We refer to the ordering defined in Lemma 3 as a *k-ordering* of the vertices.

Lemma 4. *Let $G = (V, E)$ be an n-vertex graph with arboricity c. Then, G has a vertex of degree at most $2c - 1$.*

Proof. Since G has arboricity c, $m = |E| \leqslant c(n - 1)$. So the sum of the degrees is at most $2c(n - 1)$ and hence G must have a vertex of degree at most $2c - 1$. □

Lemma 5. *Let $G = (V, E)$ be an n-vertex graph with arboricity c and let U be the set of vertices of degree at most $2c$. Then, $|U| \geqslant (1/(2c + 1))n$.*

Proof. As before, $m = |E| \leqslant c(n - 1)$. There are $n - |U|$ vertices of degree at least $2c + 1$, and summing the degrees of these vertices we get $(n - |U|)(2c + 1) \leq 2m$. The lemma follows by rearranging the terms. □

Lemma 3 implies that in order to find a k-forest coloring (and thus an optimal implicit representation) of a sparse graph G, it suffices to find a k-ordering of G. A sequential algorithm for computing a forest coloring of G is given in Algorithm 2 .

Theorem 1. *Let G be an n-vertex sparse graph with arboricity c. Then, Algorithm 2 finds a $(2c - 1)$-forest coloring of G in $O(n)$ time.*

Proof. By Lemma 4, G has a vertex of degree at most $2c - 1$; call it v_n and delete it from G. The remaining graph has also arboricity $\leqslant c$ and therefore has a vertex,

Algorithm 2. A sequential algorithm to compute forest coloring.

Input: A graph $G = (V,E)$, $|V| = n$, and its arboricity c.
Output: A $(2c-1)$-forest coloring of G.

(1) $G' := G$; $Low := \{v : \text{degree of } v \text{ in } G' \text{ is at most } 2c-1\}$; $i := n$.
(2) **while** $Low \neq \emptyset$ **do**
  (a) Pick a vertex, say u, from the set Low.
  (b) **for** each neighbor $w \notin Low$ of u **do**
    Decrease the degree of w by one and add w to the set Low if its degree becomes $2c-1$.
  (c) $G' := G' - u$; $v_i := u$; $i := i-1$.
(3) Compute a $(2c-1)$-forest coloring of G using the procedure given in the proof of Lemma 3.

Algorithm 3. A parallel algorithm to compute forest coloring.

Input: A graph $G = (V,E)$, $|V| = n$ and its arboricity c.
Output: A $2c$-forest coloring of G.

(1) $G' := G$; $i := 1$; mark all vertices unlabeled.
(2) **while** there is an unlabeled vertex **do**:
  (a) Let U be the set of vertices of G' with degree at most $2c$.
  (b) **for** each $v \in U$ **do**: label$(v) = i$.
  (c) $G' := G' - U$; update the degrees of neighbors of U accordingly.
  (d) $i := i + 1$.
(3) **for** each vertex v in G **do**: delete all the neighbors u from its adjacency list satisfying label$(u) <$ label(v).
(4) **for** each vertex v **do**: let its neighbors be $u_1,\ldots,u_\ell$, where $\ell \leqslant 2c$; color the edge (v,u_i) with color i, $1 \leqslant i \leqslant \ell$.

say v_{n-1}, of degree at most $2c-1$. By repeating this process, we obtain a sequence $v_1,\ldots,v_n$. This procedure is formalized in Algorithm 2. It is easy to verify that the sequence $v_1,\ldots,v_n$, generated in Step 2 of the algorithm, is a $(2c-1)$-ordering of vertices of G and hence, by Lemma 3, a $(2c-1)$-forest coloring of G. We now discuss the complexity of the algorithm. The time needed by each iteration of the while loop is bounded by the degree of the vertex u. So the total time of the while loop is bounded by the sum of degrees, which is $O(m) = O(n)$, since G is sparse. Also Step 3 clearly takes $O(n)$ time. The bound follows. □

A parallel algorithm to compute a forest coloring of sparse graphs is given in Algorithm 3.

Theorem 2. *Let G be an n-vertex sparse graph with arboricity c. Then, Algorithm* 3 *finds a $2c$-forest coloring of G in* $O(\log n)$ *time using* $O(n/\log n)$ CRCW PRAM *processors, or in* $O(\log n \log^* n)$ *time using* $O(n/\log n \log^* n)$ EREW PRAM *processors.*

Proof. The proof of correctness comes easily by Lemma 5. We will now analyze the complexity of the algorithm. Steps 1 and 4 can be implemented in $O(1)$ time using $O(n)$ processors on an EREW PRAM. Step 3 can be implemented in $O(\log n)$ time using $O(n/\log n)$ processors, on an EREW PRAM, by performing a prefix sum computation in the adjacency lists of G' [11]. We now argue about the complexity of Step 2. By Lemma 5, the number of iterations of the while-loop is $O(\log n)$. Note that if in each iteration we update the adjacency list and recompute the degree of each vertex after deleting U, we will spend (roughly) $O(\log n)$ time per iteration and thus $O(\log^2 n)$ time overall. Below, we show how we can do better than this. We begin with the CRCW PRAM implementation.

We will first show how we can implement Step 2 in $O(\log n)$ time with $O(n)$ processors. The implementation is based on the following observation: instead of recomputing the degree of each vertex in G', it is sufficient to mark, during the ith iteration, those vertices that have degree at most $2c$. These are exactly the vertices which will be assigned label i and will not participate in any further iteration. This can be done as follows. For every vertex $v \in G'$, assign one processor P_u to every vertex u in its adjacency list. Call such a processor *active* if u has not been labeled yet. Let $M(v)$ be a specific location in shared memory associated with vertex v. Then, all active processors P_u repeat, in parallel, the following two steps for $2c+1$ times: (a) Every P_u writes its id, $id(P_u)$, into the specified memory location $M(v)$. (b) All P_u read the contents of $M(v)$; if $M(v) = id(P_u)$, then P_u becomes inactive. As a final step, we check if the contents of $M(v)$ after the $(2c+1)$th iteration is the same as that after the $2c$th iteration. (This final step can be easily implemented in the local memory of one processor.) If this is true, then the degree of v is at most $2c$; otherwise, v has degree greater than $2c$. Call the above procedure *mark-U*. It is clear from its description that procedure mark-U takes $O(1)$ time using $O(n)$ processors on a CRCW PRAM. Hence, Step 2 takes overall $O(\log n)$ time and $O(n)$ processors.

We will now show how to reduce the number of processors to $O(n/\log n)$. The analysis is identical to the proof of Lemma 1 in [7] and originates from the method given in Section 4 of [3]. (We only describe it here for the sake of completeness.) We implement Step 2 in two phases. The first phase consists of $O(\log\log n)$ iterations. During the ith iteration we update the adjacency lists and recompute the degrees of vertices in G', using the $O(\log n/\log\log n)$-time, $O(n\log\log n/\log n)$-processor CRCW PRAM algorithm of [3] for computing prefix sums. By Lemma 5, the size of G' reduces by a constant factor ε after each iteration, where $\varepsilon \geqslant 1/(2c+1)$. Using $O(n/\log n)$ (i.e. fewer) processors, the ith iteration can be implemented in time $O(((1-\varepsilon)^i n)/(n/\log n) + \log n/\log\log n) = O((1-\varepsilon)^i \log n + \log n/\log\log n)$. As a consequence, the first phase can be implemented in $O(\log n)$ time using $O(n/\log n)$ processors. At the end of the first phase, the size of G' has been reduced to $O(n/\log n)$.

Then, in the second phase, we simply apply to G' our non-optimal implementation described above. Hence, Step 2 can be implemented in $O(\log n)$ time using $O(n/\log n)$ processors on a CRCW PRAM.

Let us now discuss the EREW PRAM implementation. As before, we will first show how to implement Step 2 in $O(\log n \log^* n)$ time using $O(n)$ processors and then we will discuss the optimal implementation. Our approach (for the non-optimal implementation) is inspired by a method used in [8]. Since now concurrent read and/or write is not allowed, we have to modify the procedure mark-U. The goal is again to mark the vertices with degree $\leqslant 2c$.

For every vertex $v \in G'$, we allocate, as before, a processor P_u to every vertex u in the adjacency list of v. Call a vertex u, as well as its associated processor, *marked* if u is of degree $\leqslant 2c$. If we delete, in one step, all marked vertices in the adjacency list of a vertex v (with degree $> 2c$), then large "gaps" may be created. But now we do not have the concurrent access capability to overcome this problem. Instead of deleting all marked vertices, we delete a (large enough) subset of them in such a way that adjacency lists in G' are correctly updated (i.e. without gaps). This allows us to check easily if the degree of a vertex v is $\leqslant 2c$. (Simply assign a processor to the adjacency list of v and let it follow the successor pointers for at most $2c$ steps. If after $2c$ steps, or earlier, the processor reaches the end of the list, then v has degree $\leqslant 2c$.)

To find the desired subset of marked vertices, we do the following. In the ith iteration of the while-loop we construct an auxiliary graph $H = (V_H, E_H)$, where $V_H = \{x : x \text{ is marked in } G'\}$ and $E_H = \{(x, y) : x, y \text{ are consecutive marked vertices in some adjacency list of } G'\}$. Let $h = |V_H|$. Note that H has maximum degree bounded by $2c$ and can be constructed in $O(1)$ time using $O(h)$ processors: to every marked vertex x in G', assign a processor P_x to its adjacency list. By following successor pointers and cross-links, processor P_x marks (in at most $2c$ steps) all occurrences of x in other adjacency lists. Then P_x, during a second pass on the adjacency list of x, checks if the successor vertex of x, $\mathrm{succ}(x)$, in the adjacency list of a vertex v is also marked. If yes, edge $(x, \mathrm{succ}(x))$ is added to E_H. (Note that duplicate edges are not created, since P_x can easily keep track of the edges that it had already added to E_H.) An edge (x, y) in H means that marked vertices x and y should not be simultaneously deleted in the adjacency list of v. This implies that an independent set in H denotes a set of marked vertices which can be deleted such that neither large gaps are created nor concurrent memory accesses occur. In [6, 8] it is shown how to compute, in such a bounded-degree graph H, an independent set I of size $|I| \geqslant \delta h$, for some constant $0 < \delta < 1$, in $O(\log^* h)$ time using $O(h)$ EREW PRAM processors.

Now, the implementation of procedure mark-U is completed as follows. If P_u is marked and $u \in I$, then u is deleted from the adjacency list of v. Since $u \in I$, this operation is not performed by the predecessor and the successor vertices of u and thus there are no memory conflicts. Hence, at the end of every iteration the adjacency list of v, for every $v \in G'$, has been correctly updated.

At the end of the ith iteration, the size of G' has been reduced by a (constant) factor of at least $\delta/(2c+1)$. This implies that the total number of iterations is bounded

Algorithm 4. A sequential algorithm to approximate arboricity.

Input: A graph $G = (V,E)$.
Output: A 2-approximation of the arboricity of G.
(1) $G' := G$; $k := 0$.
(2) **for** $i := n$ **downto** 1 **do**
(a) Let u be a vertex of smallest degree δ in G'. Define $G' = G' - u$ and update the degrees of neighbors of u accordingly.
(b) $v_i := u$; $k := \max\{k, \delta\}$.
od
(3) return k.

by $O(\log n)$. Since each iteration can be implemented in $O(\log^* n)$ time with $O(n)$ processors, Step 2 takes $O(\log n \log^* n)$ time using $O(n)$ processors. To achieve an optimal number of processors $O(n/\log n \log^* n)$, we just apply the method given in Section 4 of [8], or in the proof of Theorem 5.1 in [2]. (We do not give the analysis here, since it is rather tedious and the interested reader is referred to [2, 8] for the details.) □

By Theorems 1, 2, and Lemma 2, the following result is now immediate.

Theorem 3. *Let G be an n-vertex sparse graph of known arboricity. Then, an almost optimal implicit representation of G can be computed in:* (i) $O(n)$ *sequential time;* (ii) $O(\log n)$ *time using* $O(n/\log n)$ CRCW PRAM *processors;* (iii) $O(\log n \log^* n)$ *time using* $O(n/\log n \log^* n)$ EREW PRAM *processors.*

4. Approximating arboricity

As it is mentioned in Introduction, all previous algorithms as well as those presented in the previous section require a priori the knowledge of the arboricity of the input graph in order to obtain its implicit representation. However, the known algorithms for computing the exact value of the arboricity are based on matroid theory (either in sequential [5] or in parallel randomized computation [15]) and therefore are of high complexity.

In this section we present simple and efficient algorithms to compute a 2-approximation to the arboricity of a graph. It follows by Lemma 2 that this approximate value can be used to give an almost optimal implicit representation. In the following, $G = (V,E)$, $|V| = n$, $|E| = m$, denotes a graph of unknown arboricity c. Algorithm 4 finds a sequential 2-approximation for c.

Lemma 6. *A 2-approximation for the arboricity of a graph G can be computed in* $O(m + n)$ *time.*

Algorithm 5. A parallel algorithm to approximate arboricity.

Input: A graph $G = (V, E)$.
Output: A 2-approximation of the arboricity of G.
Comment: Procedure Par-Test-Ord is described in Algorithm 6.

(1) $\beta := \lceil m/n \rceil$; $i := 0$;
(2) **while** Par-Test-Ord$(G, \beta) =$ *false* **do**
$i := i + 1$; $\beta := 2^i\beta$;
od
(3) $L := \beta/2 + 1$; $R := \beta$; stop $:=$ *false*;
(4) **while** stop $=$ *false* **do**
(a) $\gamma := \lfloor (L + R)/2 \rfloor$;
(b) **If** Par-Test-Ord$(G, \gamma) =$ *true* **then**
If Par-Test-Ord$(G, \gamma - 1) =$ *false* **then** stop $:=$ *true* **else** $R := \gamma$
else $L := \gamma$
od
(5) Return γ.

Proof. Let k be the value returned by Algorithm 4. It is clear that $v_1, v_2, \ldots, v_n$ is a k-ordering and hence G is k-forest colorable by Lemma 3. Moreover, G contains an induced subgraph H such that $|E(H)| \geqslant (k/2)|V(H)|$, since $|E(G')| \geqslant (\delta/2)|V(G')|$ at the beginning of each iteration of the algorithm. Hence $c \geqslant (k/2)$, implying that k is a 2-approximation for c. It is routine to implement Algorithm 4 in $O(m + n)$ time.

Our parallel algorithm to find a 2-approximation for the arboricity c of a graph G consists of two phases. In the first phase we use a repeated-doubling scheme to find a range for c, as follows. Assume we have a procedure Π that, given a graph G and an integer α, returns *true* if it can find an α-ordering of G. Now, observe that $\beta = \lceil m/n \rceil$ is a lower bound for c. We set $\alpha = \beta$ and call Π. If it returns *false*, then we double β, set $\alpha = \beta$, and call Π again. After $O(\log c)$ calls to procedure Π we will obtain a β' such that $\beta'/2 \leqslant c \leqslant \beta'$ and Π returns *false* for $\alpha = \beta'/2$ and *true* for $\alpha = \beta'$. In the second phase we do a binary search in the range $(\beta'/2, \beta']$ to find a γ such that procedure Π returns *true* for $\alpha = \gamma$ and *false* for $\alpha = \gamma - 1$. The entire algorithm is given in Algorithm 5.

Lemma 7. *A 2-approximation for the arboricity c of a graph G can be found in $O(\log^2 n \log c / \log\log n)$ time using $O(m \log\log n / \log^2 n)$ CRCW PRAM processors, or in $O(\log^2 n \log c)$ time using $O(m/\log^2 n)$ EREW PRAM processors.*

Proof. First note that $\lceil m/n \rceil$ is a lower bound on the arboricity of G. Let γ^* be the value of γ for which the algorithm stops. Observe that γ^* is the smallest such value for γ. Then, $\gamma^* \leqslant 2c$, since the algorithm stops for $\gamma = 2c$ by Theorem 2. A γ^*-ordering

Algorithm 6. The procedure called by Algorithm 5.

PROCEDURE Par-Test-Ord(H, α)
Input: A graph $H = (V,E)$, $|V| = n$, and an integer α.
Output: A boolean variable **ans**. The variable **ans** is set to *true* if and only if the procedure is able to find an α-ordering of G.
(1) $H' := H$; $n' := n$; **ans** := *true*; mark all vertices unlabeled.
(2) **while** there is an unlabeled vertex **and** (**ans** = *true*) **do**
Let U be the set of vertices of H' with degree at most α.
If $|U| < (1/(\alpha+1))n'$ **then** **ans** := *false*
else
(a) mark all vertices of U as labeled.
(b) $H' := H' - U$; $n' := n' - |U|$; update the degrees of neighbors of U accordingly.
fi
od

of G results in a γ^*-forest coloring of G by Lemma 3. Hence, γ^* is a 2-approximation for arboricity of G. Let us now discuss the complexity of Algorithm 5. Step 2 is executed at most $\lceil \log c \rceil$ times. The number of iterations of the while-loop in Step 4 is $O(\log c)$. In each iteration of the while-loops in Steps 2 and 4, we call the procedure Par-Test-Ord. The complexity of this procedure dominates the resource bounds of the algorithm, since all other steps can be trivially done in O(1) time. Therefore, in the following we argue only for the complexity of the procedure. Although procedure Par-Test-Ord is very similar to Algorithm 3, unfortunately we cannot implement it in the bounds stated in Theorem 2. The reason is that during every iteration of the while-loop in Step 2 of the procedure, we have to count the cardinality of U. Hence, in every iteration we need $O(\log n/\log\log n)$ time using $O(m\log\log n/\log n)$ CRCW PRAM processors [3], or $O(\log n)$ time and $O(m/\log n)$ EREW PRAM processors [11]. Note that if the procedure does not return *false*, then the size of H' is reduced by a constant factor after every iteration. This means that the total execution time of the procedure is $O(\log^2 n/\log\log n)$ and it can be implemented using an optimal number of $O(m\log\log n/\log^2 n)$ CRCW PRAM processors. Similarly, on the EREW PRAM model, we can implement procedure Par-Test-Ord in $O(\log^2 n)$ time with $O(m/\log^2 n)$ processors. Hence, the complexity bounds stated in the lemma follow. □

It should be clear (by the proofs of Lemmas 6 and 7) that Algorithms 4 and 5 do not only compute a 2-approximation of the arboricity of G, i.e., a value k such that $k \leqslant 2c$, but also they compute a k-ordering of the vertices of G. Consequently, by Lemmas 2 and 3, these algorithms can be used to compute an almost optimal implicit representation of a sparse graph, even without knowing the exact value of arboricity. We summarize the result below.

Theorem 4. *Let G be an n-vertex sparse graph of unknown arboricity. Then, an almost optimal implicit representation of G can be computed in:* (i) $O(n)$ *sequential time;* (ii) $O(\log^2 n/\log\log n)$ *parallel time using* $O(n\log\log n/\log^2 n)$ CRCW PRAM *processors;* (iii) $O(\log^2 n)$ *parallel time using* $O(n/\log^2 n)$ EREW PRAM *processors.*

5. Extensions and final remarks

5.1. Extensions

Our work on implicit representation has interesting consequences in the framework of nested forest partitions. These partitions are introduced in the work of Nagamochi and Ibaraki [14] to find minimum cuts in graphs; see also [20] for a detailed discussion and generalizations to matroids.

Let $P = \langle F_1, F_2, \ldots, F_k\rangle$ be a partition of a graph G into forests; k is called the length of the partition P. The partition P is *nested* if each forest spans the next forest; i.e., $V(F_j) \supseteq V(F_{j+1})$ for each $j \geqslant 1$.

We have shown in Lemma 3 that given a k-ordering of G, we can find a (not necessarily nested) forest partition of G of length k. We now briefly show how this lemma can be extended to yield a nested forest partition of G of length k.

Let $v_1, \ldots, v_n$ be the given k-ordering of the vertices of G. Let us assume (inductively) that the subgraph of G induced by $v_1, \ldots, v_{n-1}$ is partitioned into k nested forests $F_1, \ldots, F_k$. Process as follows the set of edges L_{v_n} incident on v_n. Add to F_1 a maximal subset of edges from L_{v_n} such that no cycle is created in F_1, and delete these edges from L_{v_n}. Next, add to F_2 a maximal subset of edges from L_{v_n} such that no cycle is created in F_2, and delete these edges from L_{v_n}. Repeat this process until L_{v_n} becomes empty. Since v_n has degree at most k, L_{v_n} becomes empty when we add edges either to forest F_k or to some forest F_j, for $j < k$. One can easily verify that the resulting forest partition is nested.

5.2. Final remarks

We have presented simple and optimal algorithms to compute implicit representations of sparse graphs. It is known that many intersection graphs also have implicit representations [12]. The problem of characterizing the classes of graphs having implicit representation is open.

Note that Lemmas 6 and 7 compute a 2-approximation of the arboricity of any graph G (i.e., not necessarily sparse). Our bounds compare favorably with both the sequential results in [5] (whose time varies between $O(n^{1.5}\sqrt{\log n})$ and $O(n^3 \log n)$) and the parallel ones in [15] (presented in Introduction) which find the exact value of the arboricity. It will be interesting to find better approximations for the arboricity of a graph, without substantially increasing the resource bounds.

Although with our approximation we can compute an almost optimal implicit representation, our algorithms compute a number of forests which is at most twice the optimal. The known algorithms for computing an optimal forest coloring use matroid partitioning and thus have a high complexity. It is of independent interest to come up with efficient algorithms for computing an optimal forest coloring.

Acknowledgements

We are grateful to the anonymous referee for carefully reading the manuscript and making several valuable comments and suggestions. We are also grateful to Shiva Chaudhuri, Torben Hagerup, Glenn Manacher, Kurt Mehlhorn, Jaikumar Radhakrishnan, Raimund Seidel and K.V. Subrahmanyam for many helpful discussions and much useful criticism.

References

[1] J. Edmonds, Minimum partition of a matroid into independent sets, Res. NBS 69B (1965) 67–72.
[2] R. Cole and U. Vishkin, Deterministic coin tossing with applications to optimal parallel list ranking, Inform. and Control 70 (1986) 32–53.
[3] R. Cole and U. Vishkin, Faster optimal prefix sums and list ranking, Inform. and Comput. 81 (1989) 334–352.
[4] M. Fürer, X. He, M. Kao and B. Raghavachari, Parallel algorithms for straight-line grid embeddings of planar graphs, Proceedings of the 4th ACM Symposium on Parallel Algorithms and Architectures (SPAA'92) (1992) 410–419.
[5] H.N. Gabow and H.H. Westermann, Forests, frames, and games: algorithms for matroid sums and applications, Algorithmica 7 (1992) 465–497.
[6] A. Goldberg, S. Plotkin and G. Shannon, Parallel symmetry-breaking in sparse graphs, SIAM J. Discrete Math. 1 (1988) 434–446.
[7] T. Hagerup, Optimal parallel algorithms on planar graphs, Inform. Comput. 84 (1990) 71–96.
[8] T. Hagerup, M. Chrobak and K. Diks, Optimal parallel 5-colouring of planar graphs, SIAM J. Comput. 18 (1989) 288–300.
[9] M. Iri and S. Fujishige, Use of matroid theory in operating research, circuits and systems theory, Int. J. Systems Sci. 12 (1981) 27–54.
[10] A. Itai and M. Rodeh, The multi-tree approach to reliability in distributed networks, Proceedings of the 25th IEEE Symposium on FOCS (1984) 137–147.
[11] J.JáJá, An Introduction to Parallel Algorithms, (Addison-Wesley, New York, 1992).
[12] S. Kannan, M. Naor, and S. Rudich, Implicit representation of graphs, Proceedings of the 20th ACM Symposium on Theory of Computing (STOC'88) (1988) 334–343.
[13] L. Lovasz and Y. Yemini, On generic rigidity in the plane, SIAM J. Algebr. Discrete Methods 3 (1982) 91–98.
[14] H. Nagamochi and T. Ibaraki, Computing edge-connectivity in multigraphs and capacitated graphs, SIAM J. Discrete Math. 5 (1992) 54–68.
[15] H. Narayanan, H. Saran and V.V. Vazirani, Randomized parallel algorithms for matroid union and intersection, with applications to arborescences, and edge-disjoint spanning trees, Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms (SODA'92) (1992) 357–366.
[16] C.St. J.A. Nash-Williams, Edge-disjoint spanning trees of finite graphs, J. London Math. Soc. 36 (1961) 445–450.
[17] C.St. J.A. Nash-Williams, Decomposition of finite graphs into forests, J. London Math. Soc. 39 (1964) 12.

[18] T. Ohtsuki, Y. Ishizaki and H. Watanabe, Topological degrees of freedom and mixed analysis of electrical networks, IEEE Trans. Circuit Theory CT-17 (1970) 491–499.
[19] W. Schnyder, Embedding planar graphs on the grid, Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms (SODA'90) (1990) 138–148.
[20] A. Subramanian, Two recent algorithms for the global minimum cut problem, SIGACT News 26 (1995) 78–87.
[21] J. van Leeuwen, Graph algorithms, in: J. van Leeuwen, Ed., Handbook of Theoretical Computer Science, Vol. A (Elsevier, Amsterdam, 1990) 525–631.