An Optimal Parallel Matching Algorithm for Cographs

R. Lin
Department of Computer Science
SUNY at Geneseo
Geneseo, NY 14454

S. Olariu
Department of Computer Science
Old Dominion University
Norfolk, VA 23529-0162

Abstract

The class of cographs, or complement-reducible graphs, arises naturally in many different areas of applied mathematics and computer science. We show that the problem of finding a maximum matching in a cograph can be solved optimally in parallel by reducing it to parenthesis matching. With an n-vertex cograph G represented by its parse tree as input, our algorithm finds a maximum matching in G in O($\log~n$) time using O($n0$) processors in the EREW-PRAM model.

Key Words: list ranking, tree contraction, matching, parenthesis matching, scheduling, operating systems, cographs, parallel algorithms, EREW-PRAM.

1. Introduction

A well-known class of graphs arising in a wide spectrum of practical applications [1,2,7] is the class of *cographs*, or complement-reducible graphs. The cographs are defined recursively as follows:

- a single-vertex graph is a cograph;
- if G is a cograph, then its complement $\bar{G}$ is also a cograph;
- if G and H are cographs, then their union is also a cograph.

As it turns out [7], the cographs admit a unique tree representation up to

† Work supported, in part, by the National Science Foundation under grant CCR-9407180.

isomorphism. Specifically, one can associate with every cograph G a unique rooted tree $T(G)$ called the *cotree* of G, featuring the following properties:

(c1) every internal node of $T(G)$ has at least two children;

(c2) the internal nodes of $T(G)$ are labeled by either 0 (0-nodes) or 1 (1-nodes) in such a way that labels alternate along every path starting at the root;

(c3) the leaves of $T(G)$ are precisely the vertices of G; vertices x and y are adjacent in G if, and only if, the lowest common ancestor of x and y in $T(G)$ is a 1-node.

Figure 1 features a cograph and the corresponding cotree.

A *matching* in a graph is a set of edges with the property that no two of them share a common endpoint. The *size* of a matching is the number of edges it contains. A matching is *maximum* if its size is as large as possible. The *matching problem* is to find a maximum matching in a given graph G. The literature on matching is extensive. Matching problems are related to flow problems, covering problems, and scheduling, to name just a few (the interested reader is referred to [4,9,19,23,25] where many other applications are summarized).

We assume the Parallel Random Access Machine model (PRAM, for short) which consists of autonomous processors, each having access to a common memory. At each step, every processor performs the same instruction, with a number of processors masked out. In a Concurrent Read Concurrent Write PRAM (CRCW-PRAM) model, several processors may simultaneously access the same memory location in reading or writing; in the Concurrent Read Exclusive Write PRAM (CREW-PRAM) several processors may simultaneously read the same memory location, but exclusive access is used for writing. In the Exclusive Read Exclusive Write PRAM (EREW-PRAM) model, a memory location cannot be simultaneously accessed by more than one processor. The interested reader is referred to [16,28] for a detailed discussion of the PRAM family. It is easy to see that the more restrictive EREW-PRAM is the weakest member of the PRAM family. Additionally, several authors argue that EREW-PRAM is the only model that is reasonably close to real machines, making it of a particular practical interest. With this observation in mind, we shall adopt the EREW-PRAM as our model of computation.

Parallel algorithms to compute a maximum matching in general graphs have been studied but only with moderate success. In [22] a randomized parallel algorithm running in O(${log} sup 2$) expected time determines a maximum matching for general n-vertex graphs. The fastest deterministic algorithm [24] for general graphs runs in O($n log~n$) time and uses O($n sup 2$) processors. The first sublinear maximum matching algorithms for bipartite graphs runs in O($n sup {2/3} {log} sup 3$) time [12]. Given the status of the maximum matching problem for general graphs, it makes sense to study parallel algorithms to compute a maximum matching in particular classes of graphs.

In [2] a suboptimal parallel algorithm to compute a maximum matching in cographs is presented: their algorithm runs in O($log sup 2$n) time and using O($n sup 2$) processors in the CRCW model. After this paper was submitted, we learned of an algorithm of Xe [14] that computes a maximum matching in an n-vertex cograph in O($log~n$) time using O(n) processors in the CRCW-PRAM. The purpose of this work is to show that the problem of finding a maximum matching in a cograph can be solved optimally in parallel by reducing it to tree contraction and to parenthesis matching. Specifically, with an n-vertex cograph represented by its cotree, our algorithm runs in O($log~n$) EREW-PRAM time using O($n0$) processors being, therefore, cost-optimal.

The paper is organized as follows: Section 2 describes the basic tools that are used throughout the paper, namely list ranking, Euler-tour on trees, tree contraction, along with a new result about matching in graphs which lays the theoretical basis for our subsequent algorithm; Section 3 gives a detailed description of the algorithm; Section 4 summarizes the results and poses some open problems.

2. The Tools

The problem of *list ranking* is to determine in parallel the rank of every element in a given linked list, that is, the number of elements following it in the list. List ranking has turned out to be one of the fundamental techniques in parallel processing, playing a crucial role in a vast array of important parallel algorithms [1,3,5,6,17,28]. In particular, Cole and Vishkin [5] and Anderson and Miller [3] have showed that list ranking can be done optimally in O($\log n$) time using O($n0$) processors in the EREW-PRAM model.

The well-known *Euler-tour technique* developed in [26] allows one to compute a number of tree functions by reducing them to list ranking. To make our presentation self-contained, we shall present now the details of a variant of this technique. To begin, we replace every node u in T with $d(u)$ children by $d(u)+1$ copies of u, namely u^1, u^2, ..., $u^{d(u)+1}$. Next, letting w_1, w_2, ..., $w_{d(u)}$ stand for the children of u in T, with w_i $(1 \le i \le d(u))$ having d_i children, set for all $i=1,2,...,d(u)$:

• link(u^i) ← ${w_i}^1$,

• link(${w_i}^{d_i+1}$) ← u^{i+1}.

Assuming that the root of T has t children, what results is a linked list starting at ${root(T)}^1$ and ending at ${root(T)}^{t+1}$ with every edge of T traversed exactly once in each direction. Therefore, the total length of the resulting linked list is O(n). Figure 2 features a tree along with the corresponding Euler Tour.

It is worth noting that the linked list can be obtained in O($\log n$) time using roughly $\frac{d(u)}{\log n}$ processors associated with every node u in T. Clearly, this translates into a total number of O($\sum_{u \in T} \frac{d(u)}{\log n}$)=O($n0$) processors.

The problem of *tree contraction* involves reducing in parallel a given tree to its root by a sequence of vertex removals. Along with list ranking, tree contraction is already recognized as one of the fundamental techniques in parallel processing. It has found important applications in dynamic expression evaluation [1,6,10,11,15], isomorphism testing [20,21], among many others. Miller and Reif's tree contraction technique [20] uses two basic operations, RAKE - which removes all leaves in the tree, and COMPRESS - which uses a variant of list ranking to reduce the size of paths involving nodes of degree two. These two operations, together, will cause a tree with n nodes to be reduced to a single node in O($log˜n$) steps. Their algorithm runs in O($log˜n$) time using O(n) processors in the CRCW-PRAM model.

Later, Cole and Vishkin [6] proposed an optimal tree contraction algorithm running in O($log˜n$) time using O($n0$) processors in the EREW-PRAM model, by reducing the problem to list ranking. In spite of its optimality, however, their algorithm is rather involved. In a first stage, using the Euler tour technique they compute for every node in the tree the size of the subtree rooted at that node. In the second stage, based on this information, the tree is partitioned into a number of so-called centroid paths. The third stage involves scheduling an order for the vertex removal based on the centroid information. Finally, in the last stage the vertex removal takes place according to the schedule computed in the previous stage.

Quite recently, Abrahamson *et al* [1] and Gibbons and Rytter[11] proposed optimal tree contraction algorithms that are very similar in spirit to that of Cole and Vishkin, except for the way the vertices are scheduled for removal. This order is based on the initial numbering of the leaves in the original tree.

Abrahamson *et al* [1] argued that the tree contraction technique can be used to devise optimal parallel algorithms for the following problems on cographs: computing the size of the largest clique, number of cliques, number of maximal independent sets, number of cliques of the largest size, number of independent sets of largest size, and the problem of identifying a clique of maximum size. All these problems can be solved in O($log˜n$) EREW-PRAM time using O($n0$) processors by solving a corresponding tree contraction problem.

All graphs in this paper are finite with no loops or multiple edges. We use standard graph theoretical terminology compatible with [4]. Consider a graph G along with a matching M in G. It is customary to call the edges in M *matched* and the edges outside M *free*. Vertices not incident with matched edges are referred to as *exposed.* A path P: x_1, x_2, ..., x_p is called *alternating* if the edge $x_i x_{i+1}$ is free for all odd values of i and matched for all even values of i. If both x_1 and x_p are exposed vertices, then the path P is termed *augmenting*. The following fundamental result of Berge (see [23]) characterizes maximum matchings in terms of augmenting paths.

Theorem 0. A matching M is maximum if and only if there is no augmenting path with respect to M. □

Let $G=(V,E)$ be an arbitrary graph whose vertex set V partitions into disjoint sets A and B with $|A| \le |B|$, such that every vertex in A is adjacent to all the vertices in B. Let $M(B)$ be a maximum matching in the subgraph of G induced by B and let X stand for the set of all exposed vertices in B. Consider the set M of edges constructed by the following *greedy* procedure.

Procedure Greedy;
begin
    if $|A| \le |X|$ **then**
        find a bijection f from A to a subset of X;
        $M \leftarrow M(B) \cup \{\{a,f(a)\} \mid a \in A\}$
    else
        let C be a subset of $M(B)$ of size $\lceil \frac{|A|-|X|}{2} \rceil$;
        find a bijection f from A to a subset of
         $X \cup \{u,v \mid$ the edge uv belongs to $C\}$;
        $M \leftarrow (M(B)-C) \cup \{\{a,f(a)\} \mid a \in A\}$
    return(M)
end; {Greedy}

As it turns out, the following result is at the heart of our optimal parallel algorithm to compute a maximum matching in a cograph.

Theorem 1. The set M of edges returned by procedure Greedy is a maximum matching in G.

Proof. To begin note that, by construction, M is a matching in G. Let Y stand for the set of exposed vertices with respect to M. In case $|A| > |X|$, lines 5-7 guarantee that Y has at most one element and so there can be no augmenting path with respect to M. Now Theorem 0 guarantees that M is a maximum matching.

Therefore, we only need consider the case $|A| \leq |X|$. In this case, lines 2-3 in Greedy guarantee that $|M| = |M(B)| + |A|$ and that $Y \subset X$. If the statement of the theorem is false, then we find an augmenting path P: x_1, x_2, ..., x_{2p} in G. Let D stand for the set of all the vertices in P that belong to A. We note that D cannot be empty for otherwise P is contained in B only, contradicting the maximality of the matching $M(B)$.

Let x_i be an arbitrary vertex in D. Since P is alternating, exactly one of the edges $x_i x_{i-1}$ and $x_i x_{i+1}$ is matched. Now consider the matching M' of G obtained from M in the obvious way: every edge of M outside P remains unchanged while the edges in P are flipped (i.e. matched edges become free and free edges become matched). Trivially, M' has one edge more than M. Clearly, at most $|A|$ of them involve vertices from A. But now, removing from M' all the edges involving vertices from A leaves a matching of B of size $|M(B)| + 1$, contradicting the maximality of $M(B)$. The conclusion follows. □

3. The Algorithm

We shall adopt the terminology of [1]. To make our exposition self-contained we present a number of definitions. First, we assume the trees represented by an unordered array with every node in the tree featuring a parent pointer along with a doubly linked list of children. Note that property (c1) of cotrees motivates us to restrict our attention to ordered trees in which every internal node has at least two children. We now demonstrate how such a tree T can be transformed into a full binary tree BT (refer to Figure 3 for an illustration): if a node x has d ($d \geq 2$) children in T then, in BT, we add $d-2$ identical copies of x, namely x_1, x_2, ..., x_{d-2} in such a way that, with x_0 standing for x,

• the parent of x_i is x_{i-1} whenever $i \geq 1$;

• the left child of x_i is the $(i+1)$st child of x in T;

• the right child of x_i is x_{i+1} in case $i \leq d-3$, and the d-th child of x in T otherwise.

As pointed out in [1], there is no cost associated with the construction of BT, since all we need do is to reinterpret the existing pointers in T.

Given a binary tree T, a *tree contraction sequence* is a sequence of trees $T = T_1$, T_2, ..., T_m such that T_i is obtained for T_{i-1} by one of the following basic operations:

• prune(v) − leaf v of T_{i-1} is removed;

• bypass(v) − a node v having exactly one child is removed from T_{i-1}, with the unique child of v replacing v.

Abrahamson *et al* [1] show that every full binary tree has an optimal contraction sequence of length O($\log n$), and that this sequence can be obtained in O($\log n$) time using O($n0$) processors in the EREW-PRAM model.

We are now in a position to show how the tree contraction technique can be used to obtain an optimal parallel algorithm to compute a maximum matching for cographs. For this purpose, consider an n-vertex cograph G represented by its cotree $T(G)$. We binarize $T(G)$ as described above and let $BT(G)$ stand for the resulting full binary tree. For each node x of $BT(G)$ we let $BT(x)$ stand for the subtree of $BT(G)$ rooted at x; $L(x)$ stands for the set of all the leaves in $BT(x)$, and $G(x)$ represents the subgraph of G induced by $L(x)$. Throughout this paper we shall deal with a generic node x of $BT(G)$; whenever this happens, it will be assumed that x has left and right children y and z, respectively.

Our algorithm proceeds in the four stages that we briefly outline next.

Stage 1. This stage is essentially a preprocessing stage;

Stage 2. In this stage we compute for every node v of $BT(G)$ the *size* of the maximum matching in $G(v)$;

Stage 3. In this stage, using the partial result computed in the previous stage we essentially traverse $BT(G)$ from the root down to the leaves allocating $alpha$ and $beta$ tokens to various nodes of $BT(G)$. As we shall prove, every leaf of $BT(G)$ receives at most one such token;

Stage 4. The purpose of this stage is to match leaves that have received an $alpha$ token with leaves that have received a $beta$ token in Stage 3. This is accomplished essentially by calling an optimal parenthesis matching algorithm. What results is a maximum matching in G.

We now describe each and every stage in full detail.

Stage 1.

Perform an Euler-tour of $BT(G)$ and use the information returned to compute for every internal node x of $BT(G)$ the cardinality of $L(x)$. For every 1-node x of $BT(G)$ do the following:

if $|L(y)|$ ≤ $|L(z)|$ **then**
    mark y
else
    swap$(T(y),T(z))$;
    mark z
endif

In other words, we ensure that for every 1-node, its left subtree contains at most as many leaves as the right subtree. Furthermore, every left child of a 1-node in $BT(G)$ is marked. (For convenience, we continue to refer to the new tree as $BT(G)$.) Call a node v of $BT(G)$ *special* if some ancestor of v has been marked in the operation described above. Nodes of $BT(G)$ which are not special are referred to as *regular*.

Next, performing an Euler-tour again mark all the special nodes in $BT(G)$, construct a linked list of all the leaves in $BT(G)$, number the leaves in preorder as 1, 2, ..., n, and recompute the set $L(v)$ for every node v (note that the cardinality of $L(v)$ remains the same and is not recomputed).

Stage 2. In the second stage, we invoke the tree contraction algorithm in [1] to construct an optimal contraction sequence $T sub 1$, $T sub 2$, ..., $T sub {log˜n}$ for $T(G)$. This sequence will be used explicitly in the next stage. In addition, we compute a certain algebraic expression associated by $BT(G)$; specifically, we associate with every node x of $BT(G)$ a non-negative integer $m(x)$ as follows.

(1) if x is a regular leaf then $m(x)$ ← 0;

(2) if x is a regular 0-node then $m(x)$ ← $m(y)+m(z)$;

(3) if x is a regular 1-node {y special} then

(4) if x is special then $m(x)$ ← $|L(x)|$;

The interpretation of $m(x)$ is given by the following result.

Theorem 2. For every regular node x of $BT(G)$, $m(x)$ represents the size of a maximum matching in $G(x)$.

Proof. Let x be a regular node in $BT(G)$. We proceed by induction on the height of x, that is, the number of edges in the longest path from x to a leaf of $BT(G)$. If the height of x is 0, then the conclusion follows from (1).

Assume the statement true for all regular nodes in $BT(G)$ with height less than the height of x. Let x have left and right children y and z, respectively. If x is a 0-node, then by (c3), $G(x)$ is disconnected and so the size of a maximum matching in $G(x)$ equals the size of a maximum matching in $G(y)$ plus the size of a maximum matching in $G(z)$. By the induction hypothesis, this is just $m(y)+m(z)$. Now x is a 1-node; by the induction hypothesis, $m(z)$ is the size of a maximum matching in $G(z)$. Clearly, $|L(z)|−2m(z)$ represents the number of exposed vertices in the corresponding maximum matching of $G(z)$. By virtue of (c3), every vertex in $G(y)$ is adjacent to all the vertices in $G(z)$. Now let the procedure Greedy described in Section

2 compute a matching in $G(x)$. By Theorem 1, this is a maximum matching of size either $m(z)+m(y)=m(z)+|L(y)|$, in case $|L(y)| \le |L(z)|-2m(z)$, or $m(z)+m(y)-\lceil \frac{m(y)-|L(z)|+2m(z)}{2} \rceil = m(y)-\lceil \frac{m(y)-|L(z)|}{2} \rceil = m(y)+\lfloor \frac{|L(z)|-m(y)}{2} \rfloor = \lfloor \frac{|L(z)|+m(y)}{2} \rfloor = \lfloor \frac{|L(z)|+|L(y)|}{2} \rfloor = \lfloor \frac{|L(x)|}{2} \rfloor$, otherwise.

This completes the proof of Theorem 2. □

Note that Theorem 2 guarantees that the value $m(root)$ computed at the root of $BT(G)$ gives the size of a maximum matching in G. In addition, for later reference we take note of the following result.

Corollary 2.1. In case x is a regular 1-node, the size $m(x)$ of a maximum matching in $G(x)$ is given by

$$m(x) = \min\{\lfloor \frac{|L(x)|}{2} \rfloor, m(z)+|L(y)|\}.$$

Proof. To see that this is the case, note that in case $|L(y)| \le |L(z)|-2m(z)$, $m(x) = |L(y)|+m(z) \le |L(y)| + \frac{|L(z)|-|L(y)|}{2} = \frac{|L(y)|+|L(z)|}{2}$.

Since $m(x)$ is an integer, it follows that $m(x) \le \lfloor \frac{|L(y)|+|L(z)|}{2} \rfloor = \lfloor \frac{|L(x)|}{2} \rfloor$.

Similarly, in case $|L(y)| > |L(z)|-2m(z)$, we have $m(x) = \lfloor \frac{|L(y)|+|L(z)|}{2} \rfloor < \lfloor \frac{|L(y)|+|L(y)|+2m(z)}{2} \rfloor = m(z)+|L(y)|$. The conclusion follows. □

Stage 3. In the third stage, using the contraction sequence obtained in Stage 2, we traverse the tree $BT(G)$ from the root down to the leaves and distribute to various nodes in $BT(G)$ three types of objects:

- (α,β)-units, that is pairs of α-tokens and β-tokens that are treated as a unit;
- α-tokens;
- β-tokens.

To anticipate, it will turn out that for every node v of $BT(G)$, $U(v)$ is the number of matched edges in $G(v)$ in the maximum matching to be constructed; $A(v)$ is the number of vertices that will be matched with vertices with higher preorder numbers;

$B(v)$ is the number of vertices that will be matched with vertices of a lower preoder number.

We note that every (α,β)-unit contains exactly one α-token and one β-token. During the execution of the third stage of the algorithm, the (α,β)-units will be broken: when this happens, the α and β-tokens contained in the broken units are released. Every regular node x of $BT(G)$ will receive $U(x)$ (α,β)-units, $A(x)$ α-tokens and $B(x)$ β-tokens. Eventually, every leaf of $BT(G)$ will receive *at most* one such token (this will be proved later). To distribute these objects, we traverse the sequence T_1, ..., $T_{\log n}$ *backwards* allocating tokens to the nodes of $BT(G)$ in the following recursive way:

(5.1) $U(root) \leftarrow m(root)$ (α,β)-units;

(5.2) $A(root) \leftarrow 0$ α-tokens;

(5.3) $B(root) \leftarrow 0$ β-tokens.

Let x be a regular node of $BT(G)$. Then x transmits to its children:

if x is a 1-node **then**

(6.1) $U(y) \leftarrow 0$ (α,β)-units;

(6.2) $A(y) \leftarrow \min\{|L(y)|,U(x)\}$ α-tokens;

(6.3) $B(y) \leftarrow \min\{B(x),|L(y)|-A(y)\}$ β-tokens;

(6.4) $U(z) \leftarrow U(x)-A(y)$ (α,β)-units;

(6.5) $A(z) \leftarrow 0$ α-tokens;

(6.6) $B(z) \leftarrow B(x)-B(y)+A(y)$ β-tokens.

(Note that (6.1)-(6.6) guarantee that when an (α,β)-unit is broken, the released α-token is assigned to the left child and the released β-token is assigned to the right child.)

if x is a 0-node **then** {no (α,β)-unit is broken up}

(7.1) $U(y) \leftarrow \min\{m(y),U(x)\}$ (α,β)-units;

(7.2) $A(y) \leftarrow 0$ α-tokens;

(7.3) $B(y) \leftarrow \min\{B(x),|L(y)|-2U(y)\}$ β-tokens;

(7.4) $U(z) \leftarrow U(x)-U(y)$ (α,β)-units;

(7.5) $A(z) \leftarrow 0$ α-tokens;

(7.6) $B(z) \leftarrow B(x)-B(y)$ β-tokens.

The following result shows that no regular node v of $BT(G)$ receives more tokens then the number of leaves in $L(v)$. More precisely we have

Theorem 3. For a regular node v of $BT(G)$ $U(v) \le m(v)$ and $2U(v)+A(v)+B(v) \le |L(v)|$.

Proof. Let v be an arbitrary regular node in $BT(G)$. We proceed by induction on the length d of the path from v to the root. If $d=0$, then v coincides with the root and the conclusion follows by (5.1)-(5.3).

Assume the statement true for all regular nodes of $BT(G)$ whose path to the root is shorter than d. Let x stand for the parent of v. Since v is regular, it must be that x is also regular. By the induction hypothesis,

(8)

As usual, let y and z stand for the left and right children of x. Clearly, v is either y or z. Consequently, we only need show that z and y (in case y is regular) satisfy the conclusion of Theorem 3. To show that this is the case, we distinguish between the following two cases.

Case 1. x is a 0-node.

In this case, both y and z are regular. By (7.1), $U(y)=\min\{m(y),U(x)\} \le m(y)$; by (7.3), $B(y) \le |L(y)|-2U(y)$; by (7.2), $A(y)=0$. Therefore,

$$2U(y)+A(y)+B(y) \le 2U(y)-2U(y)+|L(y)|=|L(y)|,$$

as claimed.

Similarly, by (7.4), $U(z)=U(x)-U(y)$. If $U(y)=U(x)$ then $U(z)=0 \le m(z)$; otherwise, by (7.1), $U(y)=m(y)$. By (8), $U(x) \le m(x)$; by (2), $m(x)=m(y)+m(z)$. We have

$$U(z) \le m(x)-m(y)=m(y)+m(z)-m(y)=m(z).$$

To see that $2U(z)+A(z)+B(z) \le |L(z)|$ observe that if

$$2U(y)+A(y)+B(y) < |L(y)|$$

then (7.1)-(7.3), combined guarantee that $B(y)=B(x)$ and so $B(z)=0$. Therefore,

$$2U(z)+A(z)+B(z)=2U(z) \le 2m(z) \le |L(z)|.$$

Now we may assume that

$$2U(y)+A(y)+B(y) = |L(y)|.$$

By (7.4)-(7.6), together with (8), we write

$$2U(z)+A(z)+B(z)=2U(x)-2U(y)+B(x)-B(y) \le |L(x)|-2U(y)-B(y)$$

$$=|L(y)|+|L(z)|-2U(y)-B(y)=|L(z)|.$$

This settles Case 1.

Case 2. x is a 1-node.

In this case, y is a special node and so we only need show that z satisfies the conclusion of Theorem 3. First, note that by (6.2) and (6.4),

$$U(z)=U(x)-\min\{|L(y)|,U(x)\}.$$

If $U(x) \le |L(y)|$ then $U(z)=0 \le m(z)$; now we may assume that $U(x)>|L(y)|$ and so, by (8),

(9)

In case $|L(y)| \le |L(z)|-2m(z)$, (3) guarantees that $m(x)=m(z)+|L(y)|$ and so, by (9), $U(z) \le m(z)$.

In case $|L(y)| > |L(z)|-2m(z)$, we have $m(z) > \frac{|L(z)|-|L(y)|}{2}$. Now (3) implies

$$m(x)=\left\lfloor \frac{|L(x)|}{2} \right\rfloor=\left\lfloor \frac{|L(y)|+|L(z)|}{2} \right\rfloor.$$

Consequently, by (9), we can write

$$U(z) \le \left\lfloor \frac{|L(y)|+|L(z)|}{2} \right\rfloor - |L(y)| = \left\lfloor \frac{|L(z)|-|L(y)|}{2} \right\rfloor \le \frac{|L(z)|-|L(y)|}{2} < m(z).$$

To show that $2U(z)+A(z)+B(z) \le |L(z)|$, note that by (6.4), (6.5), (6.6), and (8) combined, we have

(10)

Therefore, if $A(y)+B(y)=|L(y)|$ then

$$2U(z)+A(z)+B(z)\le |L(z)|.$$

Thus, we may assume that $A(y)+B(y) < |L(y)|$. Note that by (6.2) and (6.3), $B(y)=B(x)$ (otherwise $A(y)+B(y)=A(y)+|L(y)|-A(y)=|L(y)|$, contradicting our assumption), and $A(y)=U(x)$ (otherwise, $A(y)=|L(y)|$ implying that $A(y)+B(y)\ge |L(y)|$, a contradiction).

It follows that $2U(z)+A(z)+B(z)=A(y)=U(x)\le m(x)$.

By Corollary 2.1, $m(x)\le \left\lfloor \frac{|L(x)|}{2} \right\rfloor$. Now since y is the left child of a 1-node, we must have $|L(y)|\le |L(z)|$. It follows that

$$2U(z)+A(z)+B(z)\le \left\lfloor \frac{|L(x)|}{2} \right\rfloor = \left\lfloor \frac{|L(y)|+|L(z)|}{2} \right\rfloor \le \left\lfloor \frac{|L(z)|+|L(z)|}{2} \right\rfloor = |L(z)|,$$

as claimed. This completes the proof of Theorem 3. □

Corollary 3.1. Every regular leaf of $BT(G)$ receives at most one β-token.

Proof. For every regular leaf u of $BT(G)$, (1) guarantees that $m(u)=0$; by Theorem 3, $U(u)\le m(u)=0$ and so u receives no (α,β)-units. Finally, (6.5), (7.2), (7.5) combined guarantee that u receives no α-token. The conclusion follows by Theorem 3. □

Let x be a *special* node in $BT(G)$ with left and right children y and z, respectively. By (6.1), x receives no (α,β)-units in case its parent is regular. It is immediate that none of its children will receive (α,β)-units either. Assume that x has received $A(x)$ α-tokens and $B(x)$ β-tokens. Now x will distribute tokens to its children according to the following scheme.

(11.1) $B(y) \leftarrow \min\{B(x),|L(y)|\}$ β-tokens;

(11.2) $A(y) \leftarrow \min\{A(x),|L(y)|-B(y)\}$ α-tokens.

(Note that by (11.1)-(11.2), every special node assigns as many of its received β-tokens to the left subtree as possible.)

Similarly, z receives

(12.1) $A(z)$ ← $A(x)–A(y)$ α-tokens;

(12.2) $B(z)$ ← $B(x)–B(y)$ β-tokens.

The following result is similar in spirit to Theorem 3. It asserts that the distribution of tokens to special nodes is such that no special node v receives more than $|L(v)|$ tokens. Specifically, we have

Theorem 4. Every special node v receives $A(v)$ α-tokens and $B(v)$ β-tokens such that $A(v)+B(v)≤|L(v)|$.

Proof. Let v be a special node in $BT(G)$. Since the root is always regular, v is distinct from the root. Let x stand for the parent of v in $BT(G)$. We proceed by induction on the length d of the path from v to its *lowest* regular ancestor in $BT(G)$.

Assume the statement true for all special nodes whose path to their lowest regular ancestor is shorter than d. If x is regular, than v must be the left child of x and the conclusion follows by (6.3). Therefore, we may assume that x itself is special. By the induction hypothesis applied to x, we have

(13)

Let y and z stand for the left and right child of x, respectively. First, if $v=y$ then, by (11.2), $A(y)≤|L(y)|–B(y)$, and so

$A(v)+B(v)=A(y)+B(y)≤|L(y)|–B(y)+B(y)=|L(y)|$.

Now we may assume that $v=z$. By (12.1) and (12.2),

(14)

Note that (13) and (14) guarantee that

(15)

If $A(y)+B(y)≥|L(y)|$, then (15) guarantees that $A(z)+B(z)≤|L(z)|$. We may, therefore, assume that

$A(y)+B(y)$ < $|L(y)|$.

Note that with this assumption, (11.1) and (11.2) imply that $B(y)=B(x)$ and that

$A(y)=A(x)$. But now, by (14), $A(z)+B(z)=0\leq|L(z)|$, and the proof of Theorem 4 is complete. □

Corollary 4.1. Every special leaf of $BT(G)$ receives at most one α-token or one β-token.

Proof. Follows directly from Theorem 4. □

Corollary 4.2. For a special node x of $BT(G)$ the tokens received by $L(x)$ occur in the order ββ...βαα...α.

Proof. By Corollary 4.1 every vertex in $L(x)$ receives at most one token. Let, as usual, y and z stand for the left and right children of x. By Theorem 4, the number $A(x)$ and $B(x)$ of α-tokens and β-tokens received by x satisfy $A(x)+B(x)\leq|L(x)|=|L(y)|+|L(z)|$.

We note that

[To see that this is the case, observe that by (11.1)-(11.2), if y receives α-tokens then $|L(y)|-B(y)>0$ and so $B(y)=B(x)$, in which case (12.1)-(12.2) guarantee that z will not receive β-tokens. Similarly, if z receives β-tokens then by (11.1)-(11.2), $A(y)$ must be 0, as claimed.]

Stage 4. Consider the set S of all the leaves of $BT(G)$ after Stage 3 of our algorithm is completed. By Corollaries 3.1 and 4.1 every leaf in S receives at most one α-token or β-token. Using list ranking we eliminate from S all leaves that have received no token. The following surprising results assert that S has now a very special structure.

Theorem 5. S contains $m(root)$ α-tokens and $m(root)$ β-tokens. Furthermore, every prefix of S contains at least as many α-tokens as β-tokens.

Proof. Since we started with $m(root)$ (α,β)-units and no token was lost, it is clear that S must contain exactly $m(root)$ α-tokens and $m(root)$ β-tokens. By (6.2) and (6.6) whenever a (α,β)-unit is broken, the α-token is assigned to the left subtree and the β-token is assigned to the right subtree. The conclusion follows. □

Theorem 5 guarantees that S is a well-formed set of α and β-tokens. In fact, if we replace every α-token by a left parenthesis and every β-token by a right parenthesis we get a well-formed sequence of parenthesis. Now applying the parenthesis matching of [27] we identify a set of $m(root)$ matched α-tokens and β-tokens that have been assigned to say, vertices u and v of G. The following result is at the basis of the correctness of our algorithm.

Theorem 6. Let x be an arbitrary regular 1-node in $BT(G)$ with left and right children y and z, respectively. Then every leaf in $T(y)$ that has received an α-token is matched with a leaf in $T(z)$ that has received a β-token.

Proof Consider an arbitrary regular 1-node x in $BT(G)$, and let y and z stand for the left and right children of x, respectively. By (6.1)-(6.6), when x is processed in Stage 3 of our algorithm, y receives $A(y)$ α-tokens and $B(y)$ β-tokens; similarly, z receives $U(z)$ (α,β)-units and $B(z)$ β-tokens.

Note that by Corollary 4.2, when the token distribution stage is finished, all the leaves of $T(y)$ that receive α-tokens occur to the right (in a left to right ordering) of the leaves that have received β-tokens. Therefore, no leaf in $T(y)$ that has received an α-token is matched with some other leaf in $T(y)$.

Consequently, the theorem can only fail if the number of β-tokens in $T(z)$ is smaller than $A(y)+U(z)$ (which happens to be the total number of α-tokens in $T(x)$). However, by (6.6), z receives $U(z)+B(x)-B(y)+A(y)$ β-tokens. By (6.3), $B(x)-B(y)\geq 0$ implying that $U(z)+B(x)-B(y)+A(y)\geq A(y)+U(z)$.

Since x was an arbitrary, the conclusion follows. □

We are now in a position to prove the main result of this work, namely that a maximum matching in a cograph can be computed optimally in parallel.

Theorem 7. Given an arbitrary n-vertex cograph represented by its cotree, a maximum size matching in G can be computed in O($log~n$) time using O($n0$) processors in the EREW-PRAM model of computation.

Proof. The correctness of our approach follows from Theorem 6, the initial assignment to the root expressed in (5.1)−(5.3), and Corollary 2.1.

To argue for the complexity, note that Stage 1 can be trivially performed in O($log˜n$) time using O($n0$) processors. Next, it is easy to confirm that Stage 3 can be performed in the same time and processor bounds since at every node the tokens can be distributed in O(1) time. Further, by using list ranking and the optimal parallel parenthesis matching algorithm in [27], Stage 4 has the same complexity.

Thus we only need show that Stage 2 requires O($log˜n$) time and O($n0$) processors. Specifically, we have to show that computing $m(root)$ can be done in the time of tree contraction. To show that this is the case, at each node u of $BT(G)$ we store an expression of the form $f sub u$(x) where x is an indeterminate representing the value of the unevaluated $m(u)$. Referring to Figure 4.a, suppose that u has two children v and w where w is a leaf (i.e. the value of $m(w)$ is a known constant c), and let $f sub u$(x) and $f sub v$(x) be the expressions stored by u and v respectively. Let u contain an operation #. After prune(w) and bypass(u) have been performed, the reconstructed tree is featured in Figure 4.b. Note that the expression stored at v is now $f sub u$($f sub v$(x)#c).

The crucial observation is that for the purpose of computing $m(u)$ at every node u of $BT(G)$, the expression stored at u is of the form

(16)

where a and b are constants. Initially, every node u of $BT(G)$ stores $f sub u$=min{ ⌊$|L(u)| over 2$⌋, ⌊$|L(u)| over 2$⌋+0}=⌊$|L(u)| over 2$⌋. Note, in particular, that in this scheme every leaf of $BT(G)$ stores 0 as it should.

Let u, v, w be as before. To justify (16) we only need show that if prior to prune(w) and bypass(u) u stores min{$a,x+b$} and v stores min{$a’,x+b’$}, then after these operations, v stores min{$a’’,x+b’’$} for some constants $a’’$ and

b''. We distinguish between the following two cases.

Case 1. u is a 1-node.

By Corollary 1.1, after prune(w) and bypass(u) the expression stored at v is

$$\min\{a,\min\{\lfloor \frac{|L(u)|}{2} \rfloor,\min\{a',x+b'\}+c+b\}=\min\{a'',x+b''\}$$

with $a''=\min\{a,\lfloor \frac{|L(u)|}{2} \rfloor,a'+b+c\}$ and $b''=b'+b+c$.

Case 2. u is a 0-node.

Now (2) implies that after prune(w) and bypass(u) the expression stored at v is

$$\min\{a,\min\{a',x+b'\}+c+b\}=\min\{a'',x+b''\}$$

with $a''=\min\{a,a'+b+c\}$ and $b''=b'+b+c$.

Consequently, the reconstruction of the tree after a prune and bypass operation can be performed in O(1) time. Thus, computing $m(root)$ can be done in the time of tree contraction as claimed. Since no read/write conflicts occur, the computation can be performed in the EREW-PRAM model. This completes the proof of Theorem 7. □

4. Conclusions and open problems

We have presented a cost-optimal parallel algorithm to compute a maximum matching in a complement-reducible graph (cograph, for short) thus improving over the algorithm in [2]. However, a number of questions remain open. The most important one is how to construct the cotree representation of a given cograph efficiently. Typically, a cograph recognition algorithm also creates the cotree representation. Recently, the authors have presented an algorithm that solves this task in O($\log n$) time using O($\frac{n^2+mn}{\log n}$) processors in the EREW-PRAM model [18]. More recently, the authors as well as Dahlhaus [8] have independently proposed very similar algorithms running in O($\log^2$) time using O($n+m$) processors in the CREW-PRAM model of computation. Yet another such algorithm is contained in [14]. However, the algorithm in [14] runs in O($\log^2$) time using O($n+m$) processors in the CRCW model of computation. None of the algorithm mentioned is optimal in view of a linear time sequential algorithm developed in [7]. It is a long standing open problem whether the cographs can be recognized optimally in parallel. We conjecture that this is possible and pose this as an open problem.

Further, it would be interesting to know if the technique developed in this paper, namely to reduce a matching problem on graphs to parenthesis matching can be applied to graphs other than the cographs. This promises to be an exciting topic for further research.

Acknowledgment: The authors would like to thank four anonymous referees for an exceptionally thorough reviewing job and for many constructive comments that have resulted in a significantly improved presentation. We are indebted to two of the referees for pointing out reference [14] to us.

References

1. K. Abrahamson, N. Dadoun, D. G. Kirkpatrick, and T. Przytycka, A simple parallel tree contraction algorithm, *Journal of Algorithms* 10 (1989) 287−302.

2. G. S. Adhar and S. Peng, A Parallel Algorithm for Maximum Matching in Cographs, *Proc. 21-st Intern. Conference on Combinatorics, Graph Theory, and Computing*, Boca Raton, Florida, 1990.

3. R. J. Anderson, G. L. Miller, Deterministic parallel list ranking, *Algorithmica*, 6, (1991), 859−868.

4. J. A. Bondy, U. S. R. Murty, Graph Theory with Applications, North-Holland, Amsterdam, 1976.

5. R. Cole and U. Vishkin, Approximate parallel scheduling. Part I: The basic technique with applications to optimal parallel list ranking in logarithmic time, *SIAM Journal on Computing*, 17, (1988) 128−142.

6. R. Cole and U. Vishkin, The accelerated centroid decomposition technique for optimal tree evaluation in logarithmic time, *Algorithmica*, 3, (1988), 329-346.

7. D. G. Corneil, Y. Perl, and L. K. Stewart, A linear recognition algorithm for cographs, *SIAM J. on Computing*, 14 (1985), 926−934.

8. E. Dahlhaus, Efficient parallel recognition algorithms for cographs and distance hereditary graphs, a manuscript.

9. Z. Galil, Efficient algorithms for finding maximum matchings in graphs, *Computing Surveys*, 18 (1986), 23−38.

10. H. Gazit, G. L. Miller, and S. H. Teng, Optimal tree contraction in the EREW model, *Princeton Workshop Book*, Plenum Press, New York, 1989.

11. A. Gibbons and W. Rytter, Optimal parallel algorithms for dynamic expression evaluation and context-free recognition, *Information and Computation*, 81 (1989), 32−45.

12. A. V. Goldberg, S. A. Plotkin, and P. M. Vaidya, Sublinear-time parallel algorithms for matching and related problems, *Proc. 29th IEEE Annual Symposium on Foundations*

of Computer Science, 1988, pp. 174−185.

13. X. He, Efficient parallel algorithms for solving some tree problems, *Proc. 24-th Allerton Conf. on Communication, Control, and Computing*, 1986, 777−786.

14. X. He, Parallel algorithm for cograph recognition with applications, Technical Report 91-26, Department of Computer Science, SUNY Buffalo, December 1991.

15. X. He and Y. Yesha, Binary tree algebraic computation and Parallel algorithms for simple graphs, *Journal of Algorithms* 9 (1988) 92−113.

16. R. M. Karp and V. Ramachandran, A survey of parallel algorithms for shared memory machines, in *Handbook of Theoretical Computer Science*, J. van Leeuwen, ed. North Holland, 1990, 869-941.

17. C. P. Kruskal, L. Rudolph, and M. Snir, Efficient parallel algorithms for graph problems, *Algorithmica*, 5 (1990) 43−64.

18. R. Lin and S. Olariu, A fast recognition algorithm for cographs, *Journal of Parallel and Distributed Processing*, 13, (1991), 76-91.

19. L. Lovasz, M. D. Plummer, Matching Theory, *Annals of Discrete Mathematics* 29, North-Holland, Amsterdam, 1986.

20. G. L. Miller and J. Reif, Parallel tree contraction and its applications, *Proc. 17th Annual Symposium on the Theory of Computing*, 1985, pp. 478-489.

21. G. L. Miller and J. Reif, Parallel tree contraction, Part 2: Further applications, *SIAM Journal on Computing*, 20, (1991), 1128-1147.

22. K. Mulmuley, U. Vazirani, and V. Vazirani, Matching is as easy as matrix inversion, *Combinatorica*, 7 (1987), 105−113.

23. C. H. Papadimitriou and K. Steiglitz, Combinatorial Optimization, Algorithms and Complexity, Prentice-Hall, 1982.

24. B. Schieber and S. Moran, Slowing sequential algorithms for obtaining fast distributed and parallel algorithms: maximum matchings, *Proc. 15th Annual Symposium on Principles of Distributed Computing*, 1986, 282-292.

25. M. M. Syslo, N. Deo, and J. S. Kowalik, Discrete Optimization Algorithms, Prentice-Hall, 1983.

26. R. E. Tarjan and U. Vishkin, An efficient parallel biconnectivity algorithm, *SIAM Journal on Computing*, 14 (1985) 862−874.

27. W. W. Tsang, T. W. Lam and F. Y. L. Chin, An optimal EREW parallel algorithm for parenthesis matching, *1989 International Conference on Parallel Processing*, III, 185−192.

28. U. Vishkin, Structural parallel algorithmics, Technical Report, UMIACS-TR-91-53, April, 1991.