ELSEVIER

Discrete Applied Mathematics 52 (1994) 261–273

DISCRETE APPLIED MATHEMATICS

The parallel solution of domination problems on chordal and strongly chordal graphs

Elias Dahlhaus[a,*], Peter Damaschke[1]

[a] *Department of Computer Science, University of Bonn, Römerstrasse 164, W-5300 Bonn 1, Germany*
[b] *Department of Mathematics, Friedrich Schiller University of Jena, O-6900 Jena, Germany*

Received 31 May 1990; revised 21 January 1993

Abstract

We present efficient parallel algorithms for the domination problem on strongly chordal graphs and related problems, such as the set cover problem for α-acyclic hypergraphs and the dominating clique problem for strongly chordal graphs. Moreover, we present an efficient parallel algorithm which checks, for any chordal graph, whether it has a dominating clique.

0. Introduction

Chordal graphs are graphs where each cycle has an edge joining two vertices, that are not neighbors in the cycle. Such an edge is called a *chord*. The DOMINATING SET problem is to find, for any graph $G = (V, E)$, a subset V' of the vertex set of minimum cardinality that *dominates* the whole vertex set, that means each vertex is in V' or is joined by an edge to a vertex of V'.

The DOMINATING SET problem is NP-complete even for chordal graphs [17]. But for so-called strongly chordal graphs, one can find a polynomial solution. An exact definition will be given later.

Since many problems on chordal graphs such as recognizing chordal graphs, finding an elimination ordering characterizing chordal graphs, finding a clique tree, finding a minimum coloring and a maximum clique can be parallelized [27, 24], we are interested in finding a parallel algorithm for the DOMINATING SET problem for strongly chordal graphs. Farber introduced strongly chordal graphs as a natural

* Corresponding author. Present address: Basser Department of Computer Science, University of Sydney, Australia.]

[1] Visiting the department of Computer Science of the University of Bonn in the Complexity Year of the Department of Computer Science of the University of Bonn and the Max Planck Institute of Mathematics. Present address: Department of Computer Science, Fernuniversität Hagen.

0166-218X/94/$07.00
SSDI 0166-218X(93)E0058-7

subclass of chordal graphs which admits an efficient solution of the DOMINATING SET problem. We also consider related problems, such as the existence of a dominating clique and the computation of a minimum dominating clique.

A sequential algorithm for the computation of a minimum dominating clique has been developed in [25].

Section 1 introduces the necessary notation and fundamental definitions.

Section 2 discusses the problem of the existence of a dominating clique in a chordal graph. Damaschke et al. [12] gave a simple criterion for the existence of a dominating clique in a chordal graph (the diameter is three) which also induces an $O(n^3)$-processor $O(\log n)$-time parallel algorithm. Here we give an $O(n + m)$ processor $O(\log^3 n)$ time algorithm that constructs a dominating clique, if one exists.

Section 3 proves the equivalence of the DOMINATING SET problem and the minimum dominating clique problem for strongly chordal graphs. Moreover, it will be proved that both problems are equivalent to the problem of finding a minimum subset of the hyperedge set that covers all vertices of a given β-acyclic hypergraph (the class of β-acyclic hypergraphs corresponds to the class of maximal clique sets of strongly chordal graphs). The last problem is called the COVER problem.

In Section 4 we present an efficient parallel algorithm for the COVER problem for α-acyclic hypergraphs which induces also an efficient parallel algorithm for the domination problems as defined above.

1. Notation and fundamental definitions

1.1. Notation from graph theory

A graph $G = (V, E)$ consists of a *vertex set* V and an *edge set* E. Multiple edges and loops are not allowed. The edge joining x and y is denoted by xy.

We say that x is a *neighbor* of y iff $xy \in E$. The *full neighborhood* of x is the set $\{x\} \cup \{y: xy \in E\}$ consisting of x and all neighbors of x and is denoted by $N(x)$.

A *path* is a sequence $(x_1 \dots x_k)$ of distinct vertices such that $x_i x_{i+1} \in E$.

A *cycle* is a closed path, that means a sequence $(x_0 \dots x_{k-1} x_0)$ such that $x_i x_{i+1 \pmod k} \in E$.

A *subgraph* of (V, E) is a graph (V', E') such that $V' \subset V$, $E' \subset E$.

An *induced subgraph* is an edge-preserving subgraph, that means (V', E') is an induced subgraph of (V, E) iff $V' \subset V$ and $E' = \{xy \in E: x, y \in V'\}$.

A graph (V, E) is *chordal* iff each cycle $(x_0 \dots x_{k-1} x_0)$ of length greater than 3 has an edge $x_i x_j \in E$, $j - i \neq \pm 1 \bmod k$ (which joins vertices which are not neighbors in the cycle). Sometimes they are also called *triangulated* or rigid circuit graphs. We remark that this notion is equivalent to the nonexistence of an induced cycle of length greater than 3.

A subset V' of the vertex set V is *complete* iff all vertices of V' are pairwise joined by an edge.

An inclusion-maximal complete set is called a *maximal clique.*

A theorem of Dirac [14] says:

Theorem 1.1. *A graph is chordal iff each induced subgraph has a vertex whose neighborhood is complete (such a vertex is called* simplicial).

(V, H) is called a *hypergraph* iff H is a family of subsets of V. Each $h \in H$ is also called a *hyperedge.*

The *dual* hypergraph $(V, H)^*$ of (V, H) arises from (V, H), by interchanging hyperedges and vertices and reversing the element relation.

We are especially interested in hypergraphs of maximal cliques of a chordal graph. We call a hypergraph (V, H) α-acyclic, if there is a tree T with vertex set H (*join tree*), such that for any $x \in V$, the set $\{h \in H: x \in h\}$ forms a subtree of T [4].

Independently Gavril [19] and Buneman [7] proved the following theorem.

Theorem 1.2. *A graph is chordal iff it is the intersection graph of vertices of subtrees of a tree.*

Moreover, one can prove the following results.

Theorem 1.3 (Gavril [19], Buneman [1] and Beeri et al. [4]). *A graph is chordal iff its maximal cliques form an α-acyclic hypergraph.*

As a generalization of the notion of a path and a cycle for hypergraphs, we introduce the notion of a β-path and a β-cycle for hypergraphs [15]:

A sequence $(x_0 h_0 x_1 h_1 \ldots x_{k-1} h_{k-1} x_n)$ of vertices x_i and hyperedges h_i is called a *β-path* iff $x_i \in h_i \cap h_{i-1}$ but $x \notin h_j$ for $j \neq i, i-1$.

A sequence $(x_0 h_0 \ldots x_{k-1} h_{k-1} x_0)$ is a *β-cycle* iff $x_i \in h_i \cap h_{i-1 \,(\mathrm{mod}\, k)}$, but for $j \neq i, i-1$, we have $x_i \notin h_j$.

A hypergraph is *β-acyclic* iff it does not have a β-cycle.

Obviously the dual hypergraph of a β-acyclic hypergraph is β-acyclic.

A graph whose maximal cliques form a β-acyclic hypergraph is called *strongly chordal.* This definition of strongly chordal graphs is different from the original definition of Farber [16], but equivalent to it [13] (see also [3]). We remark that every strongly chordal graph is chordal but not vice versa. A simple known counterexample is the so-called 3-*trampoline* (Fig. 1). That is the graph consisting of a complete set of

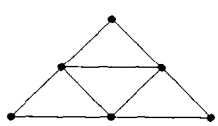

Fig. 1. A trampoline.

size 3 and an independent set of size 3 and an alternating cycle between the independent set and the complete set (of length 6).

Farber [16] defined strongly chordal graphs by *strongly perfect elimination orderings*: A graph $G = (V, E)$ is strongly chordal iff there is an ordering $<$ on the vertices of V such that

(1) for $xy, xz \in E$, such that $x < y$ and $x < z$, also $yz \in E$,

(2) for $x_1y_2, x_2y_1, x_1x_2 \in E$, such that $x_1 < y_1$ and $x_2 < y_2$, we have $y_1y_2 \in E$.

We remark that a graph is chordal iff it has an ordering satisfying 1. In that case $<$ is called a *perfect elimination ordering* [28].

Now we present the optimization problems considered in this paper:

(1) DOMINATING SET: For any graph $G = (V, E)$, find a subset V' of V of minimum cardinality which *dominates* V, that means

$$V = V' \cup \{y : \exists x \in V' \text{ such that } xy \in E\}. \qquad (*)$$

(2) DOMINATING CLIQUE: Find for a graph (V, E) a complete subset V' of minimum cardinality which satisfies (*), if it exists.

(3) COVER: Find for a hypergraph $\mathcal{H} = (V, H)$, a subset H' of H with minimum cardinality which covers V, that means

$$\bigcup_{h \in H'} h = V.$$

For any graph, the number of its vertices is denoted by n and the number of its edges is denoted by m. We represent any hypergraph (V, H) as a bipartite graph consisting of the vertex set $V \cup H$ and the edge set $\{(x, h) | x \in V, h \in H, x \in h\}$. In the case of hypergraphs n is the number of vertices and hyperedges, and m is the number of pairs (x, h) such that $x \in h$.

1.2. Notations from complexity theory

The computation model is the concurrent read exclusive write parallel random access machine (CREW-PRAM) [18].

It consists of possibly infinitely many processors and of an infinite number of memory cells. Only a finite number of processors are active. Each processor can access any memory cell. Two or more processors may read from the same memory cell simultaneously, but two or more processors may not write into the same memory cell simultaneously.

Arrays and pointers are represented in a CREW-PRAM as in a random access machine (RAM) [2].

Graphs are represented as follows: The vertices are numbered by $1, \ldots, n$. The *directed edge set* of $G = (V, E)$ consists of the set $\{(x, y): xy \in E\}$ of ordered pairs. That means each edge appears twice as an ordered pair. We assume that the directed edge set is represented as a lexicographically ordered array as in [29]:

(x, y) appears before (x', y') iff $y < y'$ or $(y = y'$ and $x < x')$.

We assume that each arithmetic operation needs one processor and one time unit.

The following known parallel complexity results are used in this paper:

(1) One processor can check for any x and y, whether $xy \in E$ in $O(\log n)$ time, by binary search.

(2) The connected components of a graph can be computed in $O(\log^{3/2} n)$ time using $O(n + m)$ processors [22] (earlier $O(\log^2 n)$ time algorithms are due to [29, 20]).

(3) For any connected graph, a spanning tree can be computed in $O(\log^2 n)$ time using $O(n + m)$ processors [30].

1.2.1. Parallel tree contraction

For any rooted directed tree T, we define a *chain* of T to be an inclusion-maximal set of consecutive vertices with degree at most two.

The *tree contraction paradigm* consists of the repeated application of the two operations *rake* and *compress* (compare [26]):

(1) *rake*: delete all leaves;

(2) *compress*: halve the chains.

Proposition 1.4 [1]. *After* $O(\log n)$ *rake and contract operations the tree* T *can be contracted to one vertex. The number of processors necessary to execute the tree contraction in parallel in* $O(\log n)$ *time is bounded by* $O(n/\log n)$.

2. The existence problem of a dominating clique

We describe an algorithm that checks for any chordal graph, whether there is a maximal clique that dominates all the vertices. In this section we assume that $G = (V, E)$ is chordal.

We start with the development of an algorithm which is very fast in parallel but not processor efficient.

Theorem 2.1. *A maximal clique dominates all vertices of a chordal graph* G *iff it has a nonempty intersection with all other maximal cliques.*

Proof. $\Rightarrow$: Suppose that the maximal clique c dominates all vertices. Assume that c' is another maximal clique and that $c \cap c' = \emptyset$. Then each $x' \in c'$ has a neighbor $x \in c$. Then for $x', x'' \in c'$, $N(x') \cap c$ and $N(x'') \cap c$ are comparable by inclusion. Otherwise if $xx' \in E$, $yx'' \in E$, $y, x \in c$ but $x'y \notin E$, $xx'' \notin E$, then $xx'x''yx$ form a chordless cycle. This is a contradiction.

But each $x' \in c'$ has a neighbor in c. Let x' be a vertex such that $N(x') \cap c$ is minimal. Then $c' \cup (N(x') \cap c)$ is complete (since $N(x'') \cap c$ contains $N(x') \cap c$, for each $x'' \in c'$). This is a contradiction to the assumption that c' is a maximal clique.

$\Leftarrow$: If each maximal clique c' intersects c, then each vertex of c' is adjacent to a vertex of c. □

Using Theorem 2.1 we can conclude the following.

Theorem 2.2. *The existence of a (not necessarily minimum) dominating clique can be decided, for any chordal graph, in* $O(\log n)$ *time by* $O(n^3)$ *processors where n is the number of vertices. Moreover, a maximal clique which is dominating can be computed in* $O(\log^2 n)$ *time using* $O(n^3)$ *processors.*

Proof. The existence of a dominating clique can be ascertained, by checking whether the diameter of the given chordal graph is at most three [12]. The set of pairs of vertices of distance at most three can be computed in $O(\log n)$ time using $O(nm)$ processors, by multiplying the adjacency matrix three times by itself.

To construct a dominating maximal clique we proceed as follows:

(1) We can compute the set of maximal cliques, by the algorithm of Ho and Lee [21], in $O(\log^2 n)$ time and with $O(n^3)$ processors. Here n is the number of vertices.

(2) Test, for each pair of maximal cliques, whether it has a nonempty intersection. This can be done straightforwardly in $O(\log n)$ time and with $O(n^3)$ processors (remember that the number of maximal cliques is bounded by n).

(3) Test if there is a maximal clique which has a nonempty intersection to each other maximal clique. This can be done straightforwardly in $O(\log n)$ time and $O(n^2)$ processors (remember that each maximal clique is determined by its smallest vertex with respect to the perfect elimination ordering and therefore the number of maximal cliques is bounded by n). □

Remark 2.3. In [8] there is an $O(\log n)$ time algorithm using $O(n^3)$ processors which computes all maximal cliques of a chordal graph. Therefore we can construct a dominating clique in $O(\log n)$ time and $O(n^3)$ processors.

We present a more processor-efficient parallel algorithm.

Using the fact that a perfect elimination ordering can be found by $O(n + m)$ processors in $O(\log^3 n)$ time [24], we can show the following more efficient result:

Theorem 2.4. *The decision problem for the existence of a (not necessarily minimum) dominating clique for chordal graphs and the construction problem for it can be solved by* $O(n + m)$ *processors in* $O(\log^3 n)$ *time.*

Proof. We proceed as follows:

Step 1: Compute a perfect elimination ordering $<$ for the chordal graph G (represented by an enumeration) [24].

Step 2: For each $x \in V$, we select the $<$-greatest neighbor $F(x)$ with respect to the perfect elimination ordering $<$. Let v be the $<$-smallest $F(x)$.

Lemma 2.5. *The complete set $N(v) \cap \{y | y \geqslant v\}$ is dominating if a dominating complete set exists.*

Proof. Suppose C is a dominating complete set and u is the smallest element of C with respect to the perfect elimination ordering $<$. Since the set of greater neighbors of u is complete, C is a subset of $C' = N(u) \cap \{y | u \leqslant y\}$. By the definition of a perfect elimination ordering, C' is also complete, and since $C \subseteq C'$ is a dominating complete set, C' is also a dominating complete set.

We choose a dominating complete set C whose smallest member u is maximal with respect to $<$ among all complete sets C.

We first claim that u is of the form $F(x)$: Suppose this is not the case. Then for all $x \in V$, $F(x) \neq u$. We then construct a dominating clique with larger smallest vertex. Suppose $uy \in E$ and $y > u$. Then $y \in C' \backslash \{u\}$. Suppose $uy \in E$ and $y < u$. Then $F(y) > u$, and since $<$ is a perfect elimination ordering and $yF(y)$ and yu are edges in E, $uF(y) \in E$. Therefore $F(y) \in C' \backslash \{u\}$. Since u is a neighbor of all vertices in $C' \backslash \{u\}$, $C' \backslash \{u\}$ is a dominating complete set and its smallest element is greater that u. This is a contradiction to the assumption that u is as large as possible.

It remains to prove that u is the $<$-minimum vertex of the form $F(x)$. Otherwise there is a vertex $F(x) < u$, and since $F(x)$ is the largest neighbor of x, x is not a neighbor of any vertex in $C' = N(u) \cap \{y | y \geqslant u\}$. □

Proof of Theorem 2.4 (*continued*).

Step 3: Test whether $N(v) \cap \{y | y \geqslant v\}$ is dominating; if this is true, output $N(v) \cap \{y | y \geqslant v\}$ as the dominating complete set. This can be done in $O(\log n)$ time by $O(n + m)$ processors as follows:

(a) For each xy with $x \in N(v) \cap \{y | y \geqslant v\}$, set y to be in DOM. This can be done by $O(m)$ processors in $O(\log n)$ time using standard methods (in constant time by a CRCW-PRAM).

(b) For each $x \in N(v) \cap \{y | y \geqslant v\}$, set x to be in DOM. This can be done in $O(\log n)$ time by $O(n)$ processors using standard methods.

(c) If all $x \in V$ are in DOM, set $N(v) \cap \{y | y \geqslant v\}$ to be dominating. This step is done in $O(\log n)$ time by $O(n)$ processors.

This completes the proof of Theorem 2.4. □

3. Equivalence of domination problems

We shall prove the following theorem.

Theorem 3.1. *The problems*

(1) DOMINATING SET *for strongly chordal graphs and* DOMINATING CLIQUE *for strongly chordal graphs are reducible to* COVER *in* $O(\log n)$ *time using* $O(n + m)$ *processors, provided a strongly perfect elimination ordering is known.*

(2) *COVER for β-acyclic hypergraphs is reducible to DOMINATING CLIQUE and DOMINATING SET for strongly chordal graphs in* $O(\log n)$ *time using* $O(n^2)$ *processors.*

(3) *DOMINATING CLIQUE for strongly chordal graphs is reducible to DOMINATING SET for strongly chordal graphs in* $O(\log n)$ *time using* $O(n+m)$ *processors provided a perfect elimination ordering is known and a dominating complete set exists.*

The proof of Theorem 3.1 is carried out as follows. First we reduce DOMINATING SET to COVER. Then we prove the reducibility of DOMINATING CLIQUE to DOMINATING SET. Lastly we reduce COVER to DOMINATING CLIQUE.

DOMINATING SET for strongly chordal graphs is reducible to the COVER problem for β-acyclic graphs using the following result of Brouwer et al. [5]:

Proposition 3.2. *For any strongly chordal graph* $G = (V, E)$

$$\mathcal{H}'_G := (V, \{N(x): x \in V\}),$$

is a β-acyclic hypergraph.

Proof of Theorem 3.1.

First we reduce DOMINATING SET *to* COVER: To get a minimum dominating set for a strongly chordal graph, we only have to get a minimum cover for $\mathcal{H}'_G := (V, \{N(x): x \in V\})$.

Now we reduce DOMINATING CLIQUE *to* DOMINATING SET: By [12], we know in case that a dominating clique exists that the minimum dominating clique is also a minimum dominating set. To compute a minimum dominating clique, we have to transfer a minimum dominating set D into a minimum dominating clique of the same cardinality.

We first compute a dominating clique DC as in the proof of Theorem 2.4. Note that DC is not necessarily of minimum cardinality and also not necessarily a maximal clique. If we know a strongly perfect elimination ordering we can compute DC by $O(n+m)$ processors in $O(\log n)$ time. If not, we need $O(\log^3 n)$ time using Klein's algorithm for the construction of a perfect elimination ordering [24]. Knowing a perfect elimination ordering, we also can determine the set of maximal cliques in $O(\log n)$ time using $O(n+m)$ processors [24]. For each maximal clique C', we count the number of $x \in C' \cap DC$ in $O(\log n)$ time using $O(n+m)$ processors. Note that there are at most $m+n$ pairs (x, C'), such that $x \in C'$. This follows immediately from the fact that $x \in C'$ iff x is the smallest vertex of C' or x is a neighbor of the smallest vertex of C'. We select a maximal clique C, such that $DC \subseteq C$ (that means we test whether $|C \cap DC| = |DC|$).

Now we assume that a minimum dominating set D is given. We have to transfer D to a minimum dominating complete set, say D', which is a subset of the maximal clique C. We start with two remarks.

Remark 3.3. Let $x \notin C$ and $y \notin C$. Let $xy \in E$. Then $N(x) \cap C$ and $N(y) \cap C$ are comparable with respect to inclusion: Otherwise there would exist $x' \in N(x) \cap C \backslash N(y)$ and $y' \in N(y) \cap C \backslash N(x)$. (x, y, y', x', x) would form a chordless cycle of length four.

Remark 3.4. Let $x, y, z \notin C$ and $xy, xz \in E$. Let $N(y) \cap C \subseteq N(x) \cap C$ and $N(z) \cap C \subseteq N(x) \cap C$. Then $N(y) \cap C$ and $N(z) \cap C$ are comparable with respect to inclusion.

To prove this remark, we make use of the fact that the given graph is strongly chordal. Assume that $N(y) \cap C$ and $N(z) \cap C$ are not comparable with respect to inclusion. Then we find $y' \in N(y) \cap C \backslash N(z)$ and $z' \in N(z) \cap C \backslash N(y)$. Since C is a maximal clique, we find a vertex $w \in C \backslash N(x)$. Note that w is also not in the neighborhood of y and z. But they $\{y, z, w, y', z'\}$ from a trampoline with $T_1 = \{y, z, w\}$ as independent set and $T_2 = \{x, y', z'\}$ as complete set. The alternating cycle between T_1 and T_2 is (y, x, z, z', w, y', y). This is a contradiction to the assumption that the given graph is strongly chordal.

Note that it was essential that C be a maximal clique to prove Remark 3.4.

We continue the description of the algorithm as follows: For each $x \in D \backslash C$, we select a vertex $\hat{x} \in N(x)$ with a minimal number of neighbors in C and select a vertex $x' \in N(\hat{x}) \cap C$. This can be done in $O(\log n)$ time using $O(n + m)$ processors, by counting the neighbors in C, for each $v \notin C$, selecting, for each $x \in D$, the vertex $\hat{x}$ afterwards, and selecting the neighbor x' of $\hat{x}$ in C at the end.

For each vertex in $x \in D \cap C$, we set $x' = x$.

We can replace x by x' and $(D \cup \{x'\}) \backslash \{x\}$ is still dominating: Consider a vertex y such that $xy \in E$. If $y \in C$ then $y = x'$ or $x'y \in E$, because also x' is in the maximal clique C.

We can assume that $y \notin C$. Note that, by Remark 3.3, $N(x) \cap C$ and $N(y) \cap C$ are comparable with respect to inclusion. Also $N(\hat{x}) \cap C \subseteq N(x) \cap C$. If $N(x) \cap C \subseteq N(y) \cap C$ then trivially $N(\hat{x}) \cap C \subseteq N(y) \cap C$. If $N(y) \cap C \subseteq N(x) \cap C$ then, by Remark 3.4, $N(\hat{x}) \cap C \subseteq N(y) \cap C$. Therefore $x' \in N(y)$ and x' is a neighbor of y. This statement is true for any neighbor y of x. Therefore $(D \backslash \{x\}) \cup \{x'\}$ is a dominating set.

Replacing all $x \in D \backslash C$ by x', we get the result that $D' = \{x' | x \in D\}$ is dominating. Since each x' is in C, D' is a dominating clique. Clearly $|D'| \leqslant |D|$. Since D is a minimum dominating set, D' is a minimum dominating clique.

Lastly we have to reduce COVER *to* DOMINATING CLIQUE: That can be done, by identifying the set H of hyperedges with a complete set, identifying the set V of vertices of the β-acyclic hypergraph with an independent set, and joining a hyperedge h and a vertex x by an edge iff $x \in h$. This graph $G_{(V,H)}$ forms a strongly chordal graph, because its maximal clique set consists of a clique, containing all $h \in H$ and the maximal cliques $C_x = \{x\} \cup \{h \in H: x \in h\}$ with $x \in V$. Since the dual hypergraph of a β-acyclic hypergraph is β-acyclic, the maximal cliques C_x together with H form a β-acyclic hypergraph, and therefore $G_{(V,H)}$ is strongly chordal.

A minimum covering set of hyperedges corresponds to a minimum dominating clique.

This completes the proof of Theorems 3.1. □

4. An efficient parallel algorithm for the COVER problem for α-acyclic hypergraphs

Since domination problems above are mutually equivalent, we need only address the parallel complexity of one of them. We choose here the problem COVER. We present an algorithm which works not only for β-acyclic hypergraphs but, moreover, for all α-acyclic hypergraphs. The algorithm is very much in the spirit of the minimum clique cover algorithm for chordal graphs of Klein [24].

Theorem 4.1. *COVER restricted to α-acyclic hypergraphs can be solved in $O(\log^2 n)$ time using $O(n + m)$ processors, provided an underlying join tree is known.*

Remark 4.2. By [8], a perfect elimination ordering and a join tree for an α-acyclic hypergraph can be computed in $O(\log^3 n)$ time using $O(n + m)$ processors.

Therefore COVER can be solved within the same processor and time bounds, if an α-acyclic hypergraph is given.

Proof of Theorem 4.1. We proceed as follows:

Step 1: We compute an enumeration π of the hyperedges of $\mathcal{H} = (V, H)$ which corresponds to an extension of the underlying join tree T to a total ordering, which we call $<$. For example, the postorder enumeration is such an enumeration and can be computed in $O(\log n)$ time using $O(n)$ processors [30].

Step 2: For each vertex $v \in V$, let h_v be the $<$-largest hyperedge that v belongs to. That means h_v is the root of the subtree of T with vertex set $\{h \in H: v \in h\}$.

Step 3: We sort the vertices v of V with respect to the ordering $<$ of the hyperedges h_v obtaining an ordering $\prec$. One who is familiar with the connections to chordal graphs, can easily verify that $\prec$ is a perfect elimination ordering of the corresponding chordal graph.

To compute h_v for all $v \in V$ simultaneously, we have to compute $\max_{\prec}(h: v \in h)$ for all $v \in V$ simultaneously. For each $v \in V$, this can be done in $O(\log n)$ time using $|\{h \in H: v \in h\}|$ processors. Recall that m is the number of pairs (x, h) such that $x \in h$. Then for all v simultaneously, h_v can be computed in $O(\log n)$ time using $O(n + m)$ processors.

Before we continue, we state the following lemma.

Lemma 4.3. *For all hyperedges h containing v, $h \cap \{v' | v \prec v'\} \subset h_v$.*

Proof. Consider any hyperedge h which contains v. Let v' be a vertex in h satisfying $v \prec v'$. If $h_v = h_{v'}$ then $v' \in h_v$ and the Lemma is proved.

Assume now that this is not the case. Then $h_{v'}$ is not a vertex of the maximal subtree T' of T having h_v as its root. Since $v \in h$ and h_v is the root of $\{h': v \in h'\}$, h is in T'. Therefore the unique path from h to $h_{v'}$ in T passes through h_v. Since $T_{v'} = \{h': v' \in h'\}$ forms a subtree of T, all hyperedges on this unique path contain v', and therefore also $v' \in h_v$. □

Corollary 4.4. *$v \prec w$ are in a common hyperedge iff $w \in h_v$.*

Denote by $G_{\mathcal{H}} = G_{(V,H)}$ the graph consisting of the vertex set V and the edge set

$$E = \{xy | \exists h \in H,\ x \in h \text{ and } y \in h\}.$$

By [4], the graph $G_{\mathcal{H}}$ is chordal.

Corollary 4.5. $G_{\mathcal{H}} = (V, \{xy | x \in h_y \text{ or } y \in h_x\})$.

Corollary 4.6. *$\prec$ is a perfect elimination ordering of $G_{\mathcal{H}}$.*

Proof of Theorem 4.1 (*Continued*).

Step 4. To solve COVER we nearly can imitate the algorithm of P. Klein for the minimum clique cover problem. We only have to take care that the number of processors is bounded by $O(n + m)$.

Step 4.1: We compute the following tree $\tilde{T}$ with the parent function P, called the *elimination tree*:

$$P(v) = \min_{\prec} \{w: vw \in E,\ v \prec w\} = \min_{\prec} \{w: w \in h_v,\ v \prec w\}.$$

This can be done in $O(\log n)$ time using $O(n + m)$ processors as follows:

(1) For each $h \in H$, sort the vertices in h with respect to $\prec$.

(2) $P(v)$ is the successor of v with respect to the sorting of h_v.

Step 4.2: We compute, for each $v \in V$, the vertex $m(v)$, which is the $\prec$ minimum vertex w, such that $v \prec w$, $vw \in E$, and $vP(w) \notin E$.

We set $b(v) = P(m(v))$.

To compute $m(v)$ in the claimed time and processor bounds, we first observe that

$$m(v) = \min_{\prec} \{w \in h_v | v \prec w,\ P(w) \notin h_v\}.$$

To compute $m(v)$, we first compute, for each h, the set M_h of vertices $v \in h$ such that $P(v) \notin h$. This can be done in $O(\log n)$ time using $O(n + m)$ processors.

To compute $m(v)$ we compute the smallest $w \in h_v$ which is in M_{h_v} and satisfies the condition $v \prec w$. This can be done in $O(\log n)$ time using $O(n + m)$ processors, for all $v \in V$ simultaneously.

Step 4.3: Using the pointer function b computed in 4.2. and the elimination tree $\tilde{T}$ with parent function P, we compute the lexicographically first maximal independent set, called X, of $G_{\mathcal{H}}$ with respect to the perfect elimination ordering $\prec$. We can proceed as in [24].

- Each leaf of $\tilde{T}$ is set to be in X.
- We call an inclusion-maximal set of consecutive vertices of $\tilde{T}$ of degree $\leqslant 2$ a *chain*. We select chains ending at a leaf, called *leaf chains*.

- We compute the vertices of X in a leaf chain: Starting with the leaves we set $b(v) \in X$. This can be done, by pointer jump techniques, in $O(\log n)$ time using $O(n)$ processors.
- We erase all vertices v which are adjacent to some $x \in X$ in $G_{\mathscr{H}}$. That means, for each $x \in X$, we erase all vertices in h_x. This can be done in $O(\log n)$ time using $O(n + m)$ processors.
- For the remaining vertices, we compute a new elimination tree: For any nonerased vertex x, we set $P(x)$ as the next ancestor of x, which is not erased. This can be done in logarithmic time using $O(n + m)$ processors.

We repeat all these steps in 4.3. until the vertex set is empty. We need at most $O(\log n)$ repetitions of these steps [24]. Therefore for the computation of X, we need $O(n + m)$ processors and $O(\log^2 n)$ time.

Step 4.4. We set $MIN = \{h_x : x \in X\}$ as the minimum cover. One can easily see that it can be computed in $O(\log n)$ time using $O(n + m)$ processors.

We can see that MIN is a minimum cover of $\mathscr{H}$, as follows: First each hyperedge of $\mathscr{H}$ is contained in some maximal clique of $G_{\mathscr{H}}$. Secondly, each maximal clique of $G_{\mathscr{H}}$ is some hyperedge of $\mathscr{H}$. Therefore any (minimum) covering of V by maximal cliques of $G_{\mathscr{H}}$ defines a (minimum) covering by hyperedges. By [27], we have to take, for each $x \in X$, some maximal clique C_x containing all neighbors of x at least as large as x with respect to $\prec$, to get a minimum covering by maximal cliques. By construction, all vertices are covered by some h_x such that $x \in X$. Therefore MIN is a minimum cover of V by hyperedges of $\mathscr{H}$.

This completes the proof of Theorem 4.1. □

Corollary 4.7. *DOMINATING SET and DOMINATING CLIQUE restricted to strongly chordal graphs can be solved by* $O(n + m)$ *processors in* $O(\log^3 n)$ *time.*

5. Conclusion

We have given parallel algorithms for the dominating set problem and the dominating clique problem on strongly chordal graphs, such that the processor-time-product is optimal up to a polylogarithmic factor. For chordal graphs, we gave a parallel algorithm for testing the existence of a dominating clique. It is also of interest to parallelize the problem of minimum Steiner trees for strongly chordal graphs, which also works in linear time [32]. A parallel algorithm to compute a minimum Steiner tree has been developed recently [9].

Acknowledgement

We are grateful to an anonymous referee who indicated the connections between the COVER problem and the minimum clique cover problem. We would like to thank Jeffrey Kingston helping us to improve our English language.

References

[1] K. Abrahamson, N. Dadoun, D.G. Kirkpatrick and T. Przyticka, A simple parallel tree contraction algorithm, J. Algorithms 10 (1989) 287–302.

[2] A. Aho, J. Hopcroft and J. Ullman, Data Structures and Algorithms (Addison-Wesley, Reading, MA, 1983).

[3] H. Bandelt and E. Prisner, Clique graphs and Helly graphs, J. Combin. Theory Ser. B 51 (1991) 34–45.

[4] C. Beeri, R. Fagin, D. Maier and M. Yannakakis, On the desirability of acyclic database schemes, J. ACM 30 (1983) 479–513.

[5] A. Brouwer, P. Duchet and A. Schrijver, Graphs, whose neighborhood has no special cycles, Discrete Math. 47 (1983) 177–182.

[6] A. Brouwer and A. Kolen, A super-balanced hypergraph has a nest-point, Rept. ZW 146/80, Mathematisch Centrum, Amsterdam.

[7] P. Buneman, A characterization of rigid circuit graphs, Discrete Math. 9 (1974) 205–212.

[8] E. Dahlhaus, Chordale Graphen im besonderen Hinblick auf parallele Algorithmen, Habilitation Thesis, University of Bonn (1991).

[9] E. Dahlhaus, A parallel algorithm for computing Steiner trees in strongly chordal graphs, to appear.

[10] E. Dahlhaus and M. Karpinski, Fast parallel computation of perfect and strongly perfect elimination schemes, Forschungsbericht Nr. 8513–CS, Universität Bonn (1987).

[11] E. Dahlhaus and M. Karpinski, The matching problem for strongly chordal graphs is in NC, Forschungsbericht Nr. 855–CS, Universität Bonn (1987).

[12] P. Damaschke, D. Kratsch and A. Lubiv, Dominating cliques in chordal graphs, Manuscript (1989).

[13] A. D'Atri and Moscarini, On hypergraph acyclicity and graph chordality, Rept. IASI-CNR R (1986)

[14] G. Dirac, On rigid circuit graphs, Abh. Math. Sem. Univ. Hamburg 25 (1961) 71–76.

[15] R. Fagin, Degrees of acyclicity and relational database schemes, J. ACM 30 (1983) 514–550.

[16] M. Farber, Characterizations of strongly chordal graphs, Discrete Math. 43 (1983) 173–189.

[17] M. Farber, Domination, independent domination and duality in strongly chordal graphs, Discrete Appl. Math. 7 (1984) 115–130.

[18] S. Fortune and J. Wyllie, Parallelism in random access machines, in: Proc. 10th ACM Symp. on Theory of Computing (1978) 114 118.

[19] F. Gavril, The intersection graphs of subtrees in trees are exactly the chordal graphs, J. Combin. Theory Ser. B 16 (1974) 47–56.

[20] D. Hirschberg, A. Chandra and D. Sarwate, Computing connected components on parallel computers, Comm. ACM 22 (1979) 461–464.

[21] C.W. Ho and R.C.T. Lee, Efficient parallel algorithms for finding maximal cliques, clique trees, and minimum coloring on chordal graphs, Inform. Process. Lett. 28 (1988) 301–309.

[22] D. Johnson and P. Metaxas, Connected components in $O(\log|V|)^{(3/2)}$ parallel time for the CREW-PRAM, in: Proc. 32nd Symp. on Foundations of Computer Science (1991) 688–697.

[23] R. Karp and V. Ramachandran, Parallel algorithms for shared-memory machines, J. van Leeuwen ed., Handbook of Theoretical Computer Science (Elsevier, Amsterdam, 1990) 871–941.

[24] P. Klein, Efficient parallel algorithms for chordal graphs, in: Proc. 29th Symp. on Foundation of Computer Science (1988) 150–161.

[25] D. Kratsch, A linear time algorithm for the minimum dominating clique problem on strongly chordal graphs, Forschungsergebnisse der Friedrich-Schiller-Universität Jena, No 87/29 (1987).

[26] G. Miller and J. Reif, Parallel tree contraction in: Proc. 26th Symp. on Foundation of Computer Science (1985) 478–489.

[27] J. Naor, M. Naor and A. Schäffer, Fast parallel algorithms for chordal graphs, SIAM J. Comput. 18 (1989) 327–349.

[28] D. Rose, Triangulated graphs and the elimination process, J. Math. Anal. Appl. 32 (1970) 597–609.

[29] Y. Shiloach and U. Vishkin, An $O(\log n)$ parallel connectivity algorithm, J. Algorithms 3 (1982) 57–67.

[30] R. Tarjan and U. Vishkin, Finding biconnected components in logarithmic parallel time, SIAM J. Comput. 14 (1985) 862–874.

[31] U. Vishkin, Implementation of simultaneous memory address access in models that forbid it, J. Algorithms 4 (1983) 45–50.

[32] K. White, M. Farber and W. Pulleyblank, Steiner trees, connected domination, and strongly chordal graphs, Networks 15 (1985) 109–124.