An optimal parallel solution for the path cover problem on P_4-sparse graphs

Katerina Asdre[1] Stavros D. Nikolopoulos[1] Charis Papadopoulos[2]

[1]*Department of Computer Science, University of Ioannina*
P.O.Box 1186, GR-45110 Ioannina, Greece
[2]*Department of Informatics, University of Bergen, N-5020 Bergen, Norway*

katerina@cs.uoi.gr stavros@cs.uoi.gr charis@ii.uib.no

Abstract: Nakano et al. in [20] presented a time- and work-optimal algorithm for finding the smallest number of vertex-disjoint paths that cover the vertices of a cograph and left open the problem of applying their technique into other classes of graphs. Motivated by this issue we generalize their technique and apply it to the class of P_4-sparse graphs, which forms a proper superclass of cographs. We show that the path cover problem on P_4-sparse graphs can also be optimally solved. More precisely, given a P_4-sparse graph G on n vertices and its modular decomposition tree, we describe an optimal parallel algorithm which returns a minimum path cover of G in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM model. Our results generalize previous results and extend the family of perfect graphs admitting optimal solutions for the path cover problem.

Keywords: P_4-sparse graphs, cographs, modular decomposition, parallel algorithms, path cover.

1 Introduction

A well studied problem with numerous practical applications in graph theory is to find a minimum number of vertex-disjoint paths of a graph G that cover the vertices of G. This problem, also known as the path cover problem, finds application in the fields of database design, networks, code optimization among many others (see [2, 8]); it is well-known that the path cover and many of its variants are NP-complete in general graphs [8]. A graph that admits a path cover of size one is referred to as Hamiltonian. Thus, the path cover problem is at least as hard as the problem of deciding whether a graph has a Hamiltonian path.

The study of graphs with few P_4's (chordless paths on four vertices) has practical applications related to examination scheduling and semantic clustering of index terms [5, 15]. These applications have motivated both the theoretical and algorithmic study of the class of cographs, which contain no induced P_4's. By extending the notion of a P_4-free graph many classes have been obtained by relaxing in various ways the absence of P_4's.

The class of *P_4-sparse* graphs is defined as the class which contains the graphs for which every set of five vertices induces at most one chordless path on four vertices [11]. This class has been extensively studied and several sequential and/or parallel algorithms for the recognition and classical optimization problems have been proposed. Giakoumakis et al. in [9] solved the recognition problem and also the problems of finding the clique number, the stability number and the chromatic number on P_4-sparse graphs in linear sequential time, i.e., in $O(n+m)$ time, where n and m are the number of vertices and edges of the input graph, respectively. Hochstättler and Tinhofer [12] presented a sequential algorithm for the path cover problem on this class of graphs, which runs in $f(n)+O(n)$ time, where $f(n)$ is the time complexity for the construction of a tree representation of a P_4-sparse graph. Recently, Asdre et al. [3] proposed a sequential algorithm for the same problem; their algorithm takes as input a P_4-sparse graph G and its modular decomposition tree, and produces a minimum path cover of G in $O(n+m)$ time. Sequential algorithms for optimization problems on other related classes of graphs (proper subclasses or superclasses of P_4-sparse graphs) have been also proposed: Lin et al. in [17] proposed an optimal algorithm for the path cover problem on cographs (a proper subclass of P_4-sparse graphs), while Giakoumakis et al. in [9] studied hamiltonicity properties for the class of P_4-tidy graphs (a proper superclass of P_4-sparse graphs); see also [4].

In a parallel environment, the recognition problem on the class of P_4-sparse graphs was studied in [16] and presented a recognition algorithm running in $O(\log^2 n)$ time with $O(\frac{n^2+nm}{\log n})$ processors on the EREW PRAM model. The problem of finding maximum matching on P_4-tidy graphs was examined in [22] and proposed an optimal parallel algorithm for the problem; the algorithm optimally computes a maximum matching of a P_4-tidy graph given its modular decomposition tree. For the class of quasi-threshold graphs (a proper subclass of cographs), the problem of recognizing whether such a graph is a Hamiltonian graph and finding a Hamiltonian path (cycle) was solved in $O(\log n)$ time with $O(n+m)$ processors on the CREW PRAM model [21]; in the same work, the coloring and other optimization problems was also solved in $O(\log n)$ time using a linear number of processors.

Recently, Nakano et al. in [20] offered a time- and work-optimal parallel solution for the path cover problem on the class of cographs. In particular, they first proved that any algorithm that solves the path cover problem on a cograph of n vertices represented by its modular decomposition tree must take $\Omega(\log n)$ time on the CREW PRAM model, and then showed that this time lower bound is tight for the class of cographs by presenting an EREW algorithm that, given an n-vertex cograph G represented by its cotree, finds and reports a minimum path cover of G in $O(\log n)$ time using $O(n/\log n)$ processors. It is worth noting that it was open for more than 10 years to find a time- and work-optimal parallel solution for this important problem.

Nakano et al. in [20] use novel techniques that combine in a clever way tree structures, called *path trees*, and sequences of square and round brackets; their algorithm produces a minimum path cover of a cograph by finding matchings of brackets in these sequences, constructing path trees, and converting the path trees to a minimum path cover using the inorder traversal. In [20] they left open the problem of applying their technique into other classes of graphs. Motivated by this issue we generalize their technique and apply it to the class of P_4-sparse graphs. We investigate the structure of the paths that occur in a minimum path cover of a P_4-sparse graph and the structure of the corresponding path trees, and present a time- and work-optimal algorithm that runs in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM model. We also show that our results can be extended to a proper superclass of P_4-sparse graphs, namely the P_4-tidy graphs.

Our work is organized as follows. In Section 2 we establish the notation and related terminology and we present background results. In Section 3 we investigate the paths that occur in a minimum path cover of a P_4-sparse graph, while in Section 4 we describe the path trees that efficiently produce such paths in a parallel process environment. Section 5 describes the construction of the bracket sequences [20] and in Section 6 we describe our optimal parallel path cover algorithm. In Section 7 we extend our results to a proper superclass of P_4-sparse graphs, namely the P_4-tidy graphs. Finally, in Section 8 we conclude the paper and discuss possible future extensions.

2 Preliminaries

We consider finite undirected graphs with no loops or multiple edges. For a graph G, we denote its vertex and edge set by $V(G)$ and $E(G)$, respectively. Let S be a subset of the vertex set of a graph G. Then, the subgraph of G induced by S is denoted by $G[S]$. Moreover, we denote by $G - S$ the graph $G[V(G) - S]$ and by $G - v$ the graph $G[V(G) - \{v\}]$.

The *neighborhood* $N(x)$ of a vertex x of the graph G is the set of all the vertices of G which are adjacent to x. The *closed neighborhood* of x is defined as $N[x] := N(x) \cup \{x\}$. The set of vertices in $V(G) - \{x\}$ that are not neighbors of x is its *non-neighbors* and is denoted by $\overline{N}(x)$. The *degree* of a vertex x in a graph G, denoted by degree(x), is the number of edges incident on x; thus, degree$(x) = |N(x)|$. A *clique* is a set of pairwise adjacent vertices while a *stable set* is a set of pairwise non-adjacent vertices.

2.1 Modular Decomposition

A subset M of vertices of a graph G is said to be a *module* of G, if every vertex outside M is either adjacent to all vertices in M or to none of them. The emptyset, the singletons, and the vertex set $V(G)$ are *trivial* modules and whenever G has only trivial modules it is called a *prime* (or *indecomposable*) *graph*. A non-trivial module is also called *homogeneous set*. A module M of the graph G is called a *strong module*, if for any module M' of G, either $M' \cap M = \emptyset$ or one module is included into the other. Furthermore, a module in G is also a module in $\overline{G}$.

The *modular decomposition* of a graph G is a linear-space representation of all the partitions of $V(G)$ where each partition class is a module. The *modular decomposition tree* $T(G)$ of the graph G (or *md-tree* for short) is a unique labelled tree associated with the modular decomposition of G in which the leaves of $T(G)$ are the vertices of G and the set of leaves associated with the subtree rooted at an internal node induces a strong module of G. Thus, the md-tree $T(G)$ represents all the strong modules of G. An internal node is labelled by either P (for *parallel* module), S (for *series* module), or N (for *neighborhood* module). It is shown that for every graph G the md-tree $T(G)$ is unique up to isomorphism and it can be constructed sequentially in linear time [7, 19]. Dahlhaus in [6] suggests a parallel algorithm for the modular decomposition of graphs that runs in $O(\log^2 n)$ on a CRCW PRAM with a linear number of processors. The approach of Dahlhaus (1995) recursively develops the modular decomposition of two induced subgraphs, which are then spliced together to produce the modular decomposition of the whole graph. Moreover, using $O(n+m)$ EREW processors the modular decomposition tree can be retrieved in $O(\log^3 n)$ as shown in [7].

Let t be an internal node of the md-tree $T(G)$ of a graph G. We denote by $M(t)$ the module corresponding to t which consists of the set of vertices of G associated with the subtree of $T(G)$ rooted at node t; note that $M(t)$ is a strong module for every (internal or leaf) node t of $T(G)$. Let $u_1, u_2, \ldots, u_p$ be the children of the node t of md-tree $T(G)$. We denote by $G(t)$ the *representative graph* of the module $M(t)$ defined as follows: $V(G(t)) = \{u_1, u_2, \ldots, u_p\}$ and $u_i u_j \in E(G(t))$ if there exists an edge $v_k v_\ell \in E(G)$ such that $v_k \in M(u_i)$ and $v_\ell \in M(u_j)$.

By the definition of a module, if a vertex of $M(t_i)$ is adjacent to a vertex of $M(t_j)$ then every vertex of $M(t_i)$ is adjacent to every vertex of $M(t_j)$. Thus $G(t)$ is isomorphic to the graph induced by a subset of $M(t)$ consisting of a single vertex from each maximal strong submodule of $M(t)$ in the modular decomposition of G. It is easy to show that the following lemma holds (see also [10]):

Lemma 2.1. *Let G be a graph, $T(G)$ its modular decomposition tree, and t an internal node of $T(G)$. Then, $G(t)$ is an edgeless graph if t is a P-node, $G(t)$ is a complete graph if t is an S-node, and $G(t)$ is a prime graph if t is an N-node.*

2.2 P_4-sparse Graphs

Below, we review characterizations and properties of P_4-sparse graphs and prove results which we use in our algorithm for the solution of the problem of the path cover of a P_4-sparse graph.

A graph G is called a *spider* if the vertex set $V(G)$ of the graph G admits a partition into sets S, K, and R such that:

P1: $|S| = |K| \geq 2$, the set S is a stable set, and the set K is a clique;

P2: all the vertices in R are adjacent to all the vertices in K and to no vertex in S;

P3: there exists a bijection $f : S \longrightarrow K$ such that exactly one of the following statements holds:

- (i) for each vertex $v \in S$, $N(v) \cap K = \{f(v)\}$;
- (ii) for each vertex $v \in S$, $N(v) \cap K = K - \{f(v)\}$.

The triple (S, K, R) is called the *spider-partition.* A graph G is a *prime spider* if G is a spider with $|R| \leq 1$. If the condition of case P3(i) holds then the spider G is called a *thin spider*, whereas if the condition of case P3(ii) holds then G is a *thick spider*; note that the complement of a thin spider is a thick spider and vice versa. A prime spider with $|S| = |K| = 2$ is simultaneously thin and thick.

It turns out that a P_4-sparse graph contains prime spiders with special properties which will be detailed later. Thus, we need to identify efficiently the spider partition of a prime spider graph G on n vertices and m edges. Based on the structural properties of a prime spider graph G, we can determine its degree sequence and partition its vertices according to the maximum and minimum degree of the sequence. This approach, though, requires $O((n+m)/\log n)$ processors and, thus, it is not optimal. Nevertheless, we prove the following lemma.

Lemma 2.2. *Let G be a prime spider on n vertices. We can recognize whether G is a thin or a thick spider in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors.*

Proof. We denote by (S, K, R) the spider-partition of G. Note that if n is an even number we know that $R = \emptyset$; otherwise $R = \{r\}$. Let $n = 2\ell + 1$ (resp. $n = 2\ell$), where $\ell \geq 3$. We choose an arbitrary vertex $x \in V(G)$ and compute the sets of its neighbors $N(x)$ and non-neighbors $\overline{N}(x)$ in G. Based on the fact that G is a prime spider we have the following cases:

(i.1) $|N(x)| = 1$: G is a thin spider, since $x \in S$.

(i.2) $|N(x)| = \ell + 1$ (resp. $|N(x)| = \ell$): G is a thin spider, since $x \in K$.

(i.3) $|N(x)| = \ell - 1$ (resp. $|N(x)| = \ell - 1$): G is a thick spider ($x \in S$).

(i.4) $|N(x)| = 2\ell - 1$ (resp. $|N(x)| = 2\ell - 2$): G is a thick spider ($x \in K$).

(i.5) $|N(x)| = \ell$: In this case, we can not immediately detect whether G is a thin or a thick spider, since $x \in R$, but we have that $K = N(x)$ and $S = \overline{N}(x)$. Thus, we choose a vertex y of the set $\overline{N}(x)$ and check the number $|N(y)|$ by the cases (i.1) and (i.3).

All the cases are verified by the definitions of a thin and a thick spider. Since the sets $N(x)$ and $\overline{N}(x)$ of an n-vertex graph G are computed in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors, the five cases can be checked within the same time and processor complexity on the same model of computation. ∎

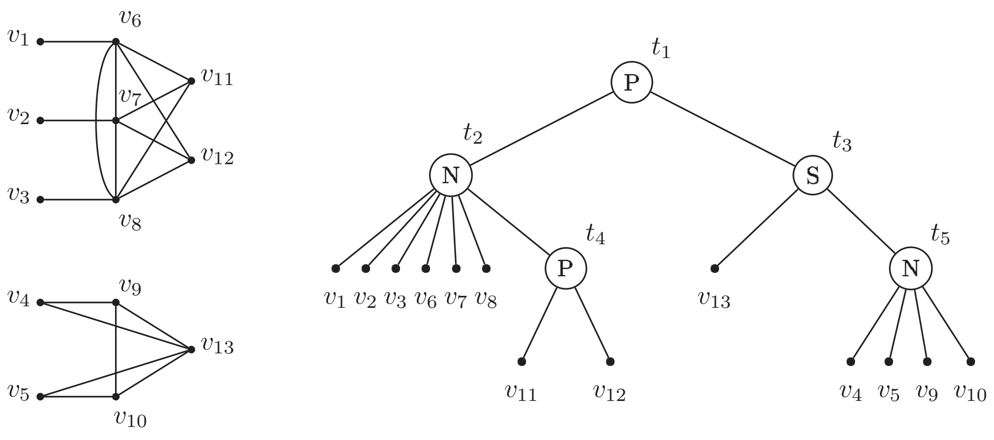

Figure 1: A disconnected P_4-sparse graph G on 13 vertices and the corresponding modular decomposition tree $T(G)$.

Let us now return to general P_4-sparse graphs. Let G be a graph and t be an N-node of the md-tree $T(G)$; recall that the vertices of $G(t)$ are the children of node t in $T(G)$. Giakoumakis and Vanherpe [10] showed the following result:

Lemma 2.3. *Let G be a graph and let $T(G)$ be its modular decomposition tree. The graph G is P_4-sparse iff for every N-node t of $T(G)$, $G(t)$ is a prime spider with a spider-partition (S, K, R) and no vertex of $S \cup K$ is an internal node in $T(G)$.*

The above lemma implies that every N-node t of the md-tree $T(G)$ of a P_4-sparse graph G has either $2k$ or $2k+1$ children, where $|S| = |K| = k \geq 2$ and $|R| \leq 1$; the sets S, K, and R form the spider-partition of the graph $G(t)$. More precisely, the N-node t has k children which correspond to S, k children which correspond to K, and either no other child if $R = \emptyset$ or one more child if $R \neq \emptyset$ (in this case, $|R| = 1$ and this child is the root of a subtree of $T(G)$). The children which correspond to S and K are leaves in $T(G)$ and, thus, they are vertices of G, while the child which corresponds to R, if $R \neq \emptyset$, is either a leaf (i.e., a vertex of G) or an internal node labelled by either P, S, or N.

The md-tree $T(G)$ depicted in Fig. 1 contains two N-nodes, that is, the nodes t_2 and t_5. The graph $G(t_2)$ is a prime spider on seven vertices with spider-partition $S = \{v_1, v_2, v_3\}$, $K = \{v_6, v_7, v_8\}$, and $R = \{t_4\}$, while $G(t_5)$ is also a prime spider on four vertices with spider-partition $S = \{v_4, v_5\}$, $K = \{v_9, v_{10}\}$, and $R = \emptyset$. The graph $G[M(t_2)]$ is a spider (non prime spider) with $R = \{v_{11}, v_{12}\}$, and the graph $G[M(t_5)]$ is also a spider (prime spider) with $R = \emptyset$.

2.3 Number of Paths

Based on the techniques described in [18, 20], we modify the tree $T(G)$: We binarize the tree $T(G)$ in such a way that each of its internal nodes labelled by either P or S has exactly two children; we denote by $T_b(G)$ the resulting tree. The left and right child of an internal P-node or S-node t of $T_b(G)$ will be denoted by t_l and t_r, respectively. Note that if $T(G)$ has only N-nodes then $T_b(G)$ coincides with $T(G)$.

Let $G[M(t)]$ denote the subgraph induced by the leaf descendants of t in $T_b(G)$, and let $L(t)$ denote the number of vertices of $G[M(t)]$. We say that $T_b(G)$ is *leftist*, denoted by $T_{bl}(G)$, if for every internal node t labelled by either P or S, the condition $L(t_l) \geq L(t_r)$ is satisfied, where t_l and t_r are the left and right child of t, respectively. For every S-node t of $T_{bl}(G)$, we replace the subtree rooted at node t_r with the $L(t_r)$ leaves and call the resulting tree the *reduced* leftist binary tree of $T_{bl}(G)$; we denote it by $T_{blr}(G)$.

Let $\lambda(t)$ denote the number of paths in the minimum path cover of the graph $G[M(t)]$. It is easy to see that, in order to construct the path cover using the tree $T_{blr}(G)$, we need to know the number of paths $\lambda(t)$ of each internal node $t \in T_{blr}(G)$. Recall that, if t is a P-node or S-node then it has a left child t_l and a right child t_r; otherwise, t is an N-node and it has at least 4 children which induce a prime spider $G(t) = (S, K, R)$ with either $R = \emptyset$ or $R = \{r\}$, $r \in T_{blr}(G)$. Note that in the case where $R = \emptyset$ we set $\lambda(r) = 0$, otherwise $\lambda(r) \geq 1$. Based on the results of [12, 18], we obtain the following formula for the number of paths in a minimum path cover of a P_4-sparse graph.

$$\lambda(t) = \begin{cases} \lambda(t_l) + \lambda(t_r) & \text{if } t \text{ is a P-node,} \\ \max\{1,\ \lambda(t_l) - L(t_r)\} & \text{if } t \text{ is an S-node,} \\ \lambda(r) + \left\lceil \max\left\{0,\ \frac{|K| - 2\lambda(r)}{2}\right\}\right\rceil & \text{if } G(t) \text{ is a thin spider,} \\ \max\{1,\ \lambda(r)\} & \text{if } G(t) \text{ is a thick spider.} \end{cases} \tag{1}$$

We conclude with the following results.

Lemma 2.4. *Let G be a P_4-sparse graph on n vertices and let $T(G)$ be its modular decomposition tree. The leftist binary tree $T_{bl}(G)$ and the reduced leftist binary tree $T_{blr}(G)$ can be computed in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors.*

Proof. Since we only binarize each subtree of $T(G)$ rooted at a P-node or an S-node and since $T(G)$ contains $O(n)$ nodes, we can construct both the leftist binary tree $T_{bl}(G)$ and the reduced leftist binary tree $T_{blr}(G)$ in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors; see [1] and Lemma 5.2 of [20]. ∎

Lemma 2.5. *Let G be a P_4-sparse graph on n vertices and $T_{blr}(G)$ be its reduced leftist binary tree. For every internal node t of $T_{blr}(G)$, the number of paths $\lambda(t)$ in a minimum path cover of $G[M(t)]$ can be computed in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors.*

Proof. By Eq. (1) and Lemma 2.4 of [20] it suffices to detect whether $G(t)$ is a thin spider or a thick spider for every internal N-node t of $T_{blr}(G)$. Let q be the number of the N-nodes of $T_{blr}(G)$ and let t_i be an internal N-node of $T_{blr}(G)$, $1 \leq i \leq q$. By Lemma 2.2 it follows that each graph $G(t_i)$ can be checked in $O(\log n_i)$ time using $O(n_i/\log n_i)$ EREW PRAM processors, where $n_i = V(G(t_i))$. Recall that for every graph $G(t_i)$ at least $n_i - 1$ vertices are leaves in $T_{blr}(G)$; see Lemma 2.3. Thus the overall time and processor complexity are $O(\log n)$ time and $O(n/\log n)$ EREW PRAM processors, respectively. ∎

2.4 Path Trees and Bracket Matching

In this section we review the *path trees* [20], which are the key ingredients of our algorithm and we show that using bracket matching we can make the construction of path trees more efficient.

Let G be a graph and let $P = [p, \ldots, p']$ be a path of a minimum path cover $\mathcal{P}$ of G. Note that, if G is a hamiltonian graph then $\mathcal{P} = \{P\}$. A path tree, denoted by $T(p, p')$, is a rooted binary tree whose nodes are exactly the vertices of a path P of the minimum path cover $\mathcal{P}$ of G and p, p' are the endpoints of P. The vertices of the path tree $T(p, p')$ are placed in $T(p, p')$ in such a way that the inorder traversal of $T(p, p')$ returns the path P. It follows that the path tree of a given path P, is not unique. Note that, a path P can be constructed from its corresponding path tree $T(p, p')$, optimally in parallel, by applying the Euler tour technique [20].

Let $T_{blr}(G)$ be the reduced leftist binary tree obtained from the md-tree $T(G)$ of the graph G. In order to construct the path trees efficiently in a parallel environment, we generate a sequence of square/round brackets for each node of $T_{blr}(G)$. We use two types of brackets: Square brackets "["

and "]" and round brackets "(" and ")". The path trees are constructed by finding matching pairs of square brackets and matching pairs of round brackets independently. Note that, given a bracket sequence corresponding to the vertices of a graph G, the path trees and consequently the path cover of G can be constructed optimally. These matchings correspond to the edges of a path tree. Specifically, for vertices a and b, we establish an edge as follows:

- $\overset{a^p}{[}\ \overset{b^l}{]}$ an edge connecting the vertex a to its parent b as a left child;
- $\overset{a^p}{[}\ \overset{b^r}{]}$ an edge connecting the vertex a to its parent b as a right child;
- $\overset{a^l}{(}\ \overset{b^p}{)}$ an edge connecting the vertex b to its parent a as a left child;
- $\overset{a^r}{(}\ \overset{b^p}{)}$ an edge connecting the vertex b to its parent a as a right child.

2.5 Time and Work Optimality

Let G be a cograph on n vertices and let $T(G)$ be its representative cotree. For the minimum path cover problem, Nakano et al. [20] have proved the following result:

Theorem 2.1. (Nakano et al. [20]): *Every algorithm that determines the number of paths in a minimum path cover or reports the minimum path cover of an n-vertex cograph represented by its cotree must take* $\Omega(\log n)$ *CREW time even if an infinite number of processors is available.*

To verify the time- and work-optimality of our algorithm note that the class of P_4-sparse graphs is a proper superclass of cographs and that the md-tree $T(G)$ of a P_4-sparse graph G contains P-nodes, S-nodes and N-nodes; if $T(G)$ has only P- or S-nodes then $T(G)$ coincides with the definition of a cotree (see [4]). Thus, due to the following result, our algorithm is time- and work-optimal.

Corollary 2.1. *Every algorithm that determines the number of paths in a minimum path cover or reports the minimum path cover of an n-vertex* P_4*-sparse graph* G *represented by its md-tree must take* $\Omega(\log n)$ *CREW time even if an infinite number of processors is available.*

3 Path Cover in P_4-sparse Graphs

In this section we review some ideas for finding a minimum path cover of a P_4-sparse graph G. We suppose that the reduced leftist binarized tree $T_{blr}(G)$ of the input graph G is given. We focus on the internal N-nodes since the cases of the P-nodes and S-nodes have already been studied in [17].

Let t be an internal N-node of $T_{blr}(G)$. Let $\mathcal{P}$ be the minimum path cover of the graph $G[M(t)]$ and let $\lambda(t)$ be the number of paths in $\mathcal{P}$, i.e., $\lambda(t) = |\mathcal{P}|$; recall that, $M(t)$ is the module which corresponds to node t and consists of all the vertices of G associated with the subtree of $T(G)$ rooted at t.

Let $G(t) = (S, K, R)$ be a prime spider and let $S = \{s_1, s_2, \ldots, s_\ell\}$ and $K = \{k_1, k_2, \ldots, k_\ell\}$, where $|S| = |K| = \ell$; by definition, there exists a bijection f such that $f(s_i) = k_i, 1 \le i \le \ell$. If $R = \{r\}$ then let $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_d\}$ be a minimum path cover of $G[M(r)]$, where $d = \lambda(r)$, and let q_i and q'_i be the endpoints of the path Q_i, $1 \le i \le d$. Then, for the computation of the minimum path cover $\mathcal{P}$ of $G[M(t)]$ we distinguish the following two cases.

Case 1. $G(t) = (S, K, R)$ is a thin spider.

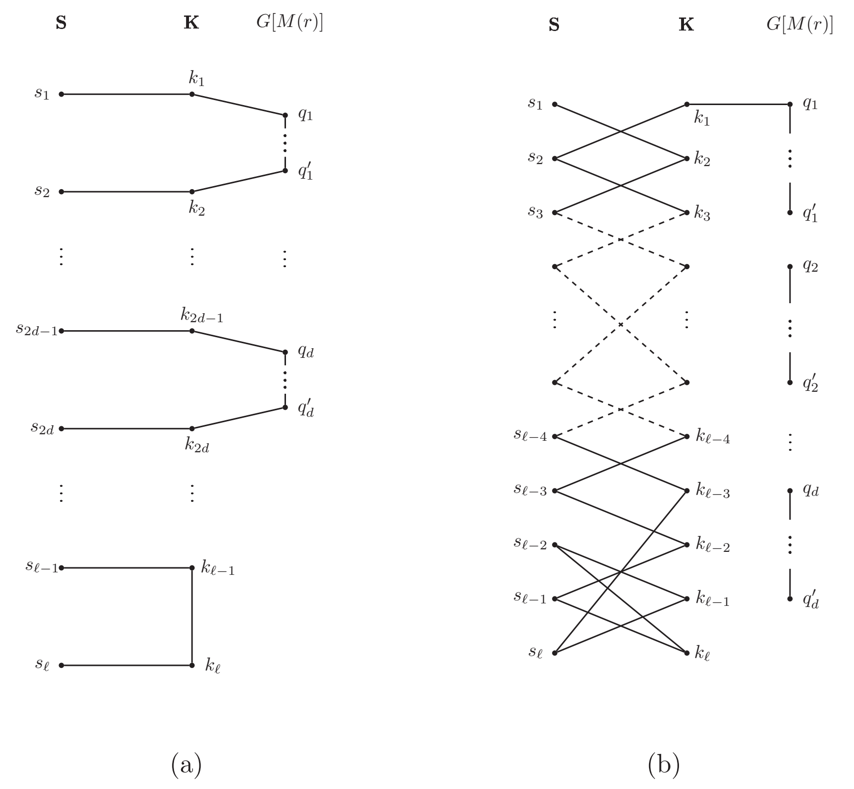

Figure 2: Illustrating the path cover of (a) a thin spider graph and (b) a thick spider.

1.1 $R = \emptyset$. The graph $G[M(t)]$ contains $k = \lceil \frac{\ell}{2} \rceil$ paths. Thus, if ℓ is even then the $\ell/2$ paths in a minimum path cover of $G[M(t)]$ are the following:

$$\mathcal{P} = \{[s_1k_1k_2s_2], [s_3k_3k_4s_4], \ldots, [s_{\ell-1}k_{\ell-1}k_\ell s_\ell]\}. \tag{2}$$

Note that the form of a path P of $\mathcal{P}$ can be arbitrary in terms of the order of the pair of adjacent vertices s_ik_i; that is, any path P of $\mathcal{P}$ can have the following form $P = [s_ik_ik_js_j]$, for $i \neq j$ and $1 \leq i, j \leq \ell$. For simplicity we adopt the order of the vertices as shown in Eq. (2).

In the case where ℓ is odd, the $(\ell+1)/2$ paths in a minimum path cover of $G[M(t)]$ are:

$$\mathcal{P} = \{[s_1k_1k_2s_2], [s_3k_3k_4s_4], \ldots, [s_{\ell-2}k_{\ell-2}k_{\ell-1}s_{\ell-1}], [s_\ell k_\ell]\}. \tag{3}$$

1.2 $R = \{r\}$. The paths of $G[M(t)]$ are obtained by joining the endpoints of some paths Q_i of $G[M(r)]$ with the vertices of the k paths of $G[S \cup K]$. Thus, if $k \leq d$ (see Fig. 2(a)) and ℓ is even we have:

$$\mathcal{P} = \{[s_1k_1q_1 \ldots q_1'k_2s_2], \ldots, [s_{\ell-1}k_{\ell-1}q_k \ldots q_k'k_\ell s_\ell], Q_{k+1}, Q_{k+2}, \ldots, Q_d\}, \tag{4}$$

while if $k \leq d$ and ℓ is odd we have:

$$\mathcal{P} = \{[s_1k_1q_1 \ldots q_1'k_2s_2], \ldots, [s_{\ell-2}k_{\ell-2}q_{k-1} \ldots q_{k-1}'k_{\ell-1}s_{\ell-1}], [s_\ell k_\ell q_k \ldots q_k'], Q_{k+1}, Q_{k+2}, \ldots, Q_d\}. \tag{5}$$

If $k > d$ and ℓ is even then the paths that occur in a minimum path cover of $G[M(t)]$ are obtained by joining d paths of $G[S \cup K]$ with the d paths of $G[M(r)]$. Thus, we have:

$$\mathcal{P} = \{[s_1k_1q_1 \ldots q_1'k_2s_2], \ldots, [s_{2d-1}k_{2d-1}q_d \ldots q_d'k_{2d}s_{2d}], \ldots, [s_{\ell-1}k_{\ell-1}k_\ell s_\ell]\}. \tag{6}$$

If $k > d$ and ℓ is odd, then the paths in a minimum path cover of $G[M(t)]$ are:

$$\mathcal{P} = \{[s_1k_1q_1 \ldots q_1'k_2s_2], \ldots, [s_{2d-1}k_{2d-1}q_d \ldots q_d'k_{2d}s_{2d}], \ldots, [s_{\ell-2}k_{\ell-2}k_{\ell-1}s_{\ell-1}], [s_\ell k_\ell]\}. \tag{7}$$

Case 2. $G(t) = (S, K, R)$ is a thick spider.

2.1 $R = \emptyset$. The graph $G[M(t)]$ is a hamiltonian graph and every edge in the Hamilton path has one endpoint in S and the other endpoint in K (i.e., there exists no Hamilton path which contains an edge with both endpoints in K; for example, see Fig. 2(b)). Thus, if ℓ is an odd number and $\ell > 3$ we have:

$$\mathcal{P} = \{[s_1 k_2 s_3 \dots k_{\ell-3} s_{\ell-2} k_{\ell-1} s_\ell k_{\ell-2} s_{\ell-1} k_\ell s_{\ell-3} k_{\ell-4} \dots k_1]\}. \tag{8}$$

Note that, if $\ell = 3$ the path is $\mathcal{P} = \{[s_1 k_3 s_2 k_1 s_3 k_2]\}$. In the case where ℓ is even and $\ell = 2$ the graph $G[M(t)]$ is also a thin spider and thus the path that occurs is $\mathcal{P} = \{[s_1 k_2 k_1 s_2]\}$. Thus, if ℓ is even and $\ell > 2$ the paths that occur in a path cover of $G[M(t)]$ are:

$$\mathcal{P} = \{[s_1 k_2 s_3 \dots k_{\ell-2} s_{\ell-1} k_\ell s_{\ell-2} k_{\ell-1} s_\ell k_{\ell-3} s_{\ell-4} k_{\ell-5} \dots k_1]\}. \tag{9}$$

We note that in Eqs. (8)–(9) the order of the vertices can be obtained in many other ways. In fact, if we assume that the set K is an independent set then the Hamilton path of $S \cup K$ can be constructed by any DFS traversal starting from an arbitrary vertex of S. More specifically in Eq. (8) (resp. Eq. (9)) the order between the vertices s_1 and s_ℓ (resp. k_ℓ) can have the form $k_i s_{i+1} k_j s_{j+1}$, for $i \neq j$, $j \neq i+1$, and $1 < i, j < \ell$. Similarly, between the vertices k_ℓ (resp. s_ℓ) and k_1 we can have an arbitrary order of the vertices of the form $s_i k_{i-1} s_j k_{j-1}$ (resp. $k_i s_{i-1} k_j s_{j-1}$), for $i \neq j$, $j \neq i-1$, and $1 < i, j \leq \ell - 3$. In our study, for convenience we adopt the order shown in Eqs. (8)–(9).

2.2 $R = \{r\}$. The Hamilton path of $G[S \cup K]$ is connected to the path Q_1 of $G[M(r)]$. Thus, the paths that occur in a path cover of $G[M(t)]$ are:

$$\mathcal{P} = \{[s_1 k_2 s_3 \dots k_3 s_2 k_1 q_1 \dots q_1'], Q_2, Q_3, \dots, Q_d\}. \tag{10}$$

Again, if $\ell = 2$ the paths that occur are $\mathcal{P} = \{[s_1 k_2 q_1 \dots q_1' k_1 s_2], Q_2, Q_3, \dots, Q_d\}$, while if $\ell = 3$ we have $\mathcal{P} = \{[s_1 k_3 s_2 k_1 s_3 k_2 q_1 \dots q_1'], Q_2, Q_3, \dots, Q_d\}$.

In both cases, the paths in $\mathcal{P}$ form a minimum path cover of $G[M(t)]$. Note that in a sequential environment we can compute a minimum path cover in a P-node or an S-node t of $T_{blr}(G)$ by using appropriate functions described in [17]. Similar results have appeared in [12].

4 Path Trees of P_4-sparse Graphs

Although the sequential algorithm is quite simple, a naive parallelization of this algorithm needs time proportional to the height of the tree $T_{blr}(G)$, which in the worst case is $O(n)$. In order to obtain an efficient parallel algorithm, we make use of the path tree structures and a bracket matching technique introduced in [20] (see Section 2).

According to the way that a path tree is constructed, a vertex can be characterized as *insert* or *bridge* vertex. A detailed description of a path tree construction, corresponding to a subtree of $T_{blr}(G)$ rooted at a P-node or an S-node t is presented in [20]. The construction of a path tree corresponding to a subtree of $T_{blr}(G)$ rooted at a P-node or an S-node t is performed by taking the union of two path trees (in the case where t is a P-node) or by inserting a vertex v into a path tree or by using a vertex v to bridge two path trees (in the cases where t is an S-node).

Let $P_1 = [p_1, \dots, p_1']$ and $P_2 = [p_2, \dots, p_2']$ be two paths of a graph G and let $T(p_1, p_1')$ and $T(p_2, p_2')$ be their path trees rooted at nodes $p_1'' \in P_1$ and $p_2'' \in P_2$.

Suppose that a vertex $v \notin P_1$ has to be inserted in the path P_1. In this case, we seek for an appropriate modification of the path tree $T(p_1, p_1')$. Let u be a node of $T(p_1, p_1')$, i.e., $u \in P_1$, having at most one child (either left or right) in $T(p_1, p_1')$. Then, the vertex v is inserted in the path tree $T(p_1, p_1')$ either (i) as a left or right child of the node u of $T(p_1, p_1')$ or (ii) as the root of the path tree $T(p_1, p_1')$ (in this case, p_1'' becomes a left or right child of vertex v).

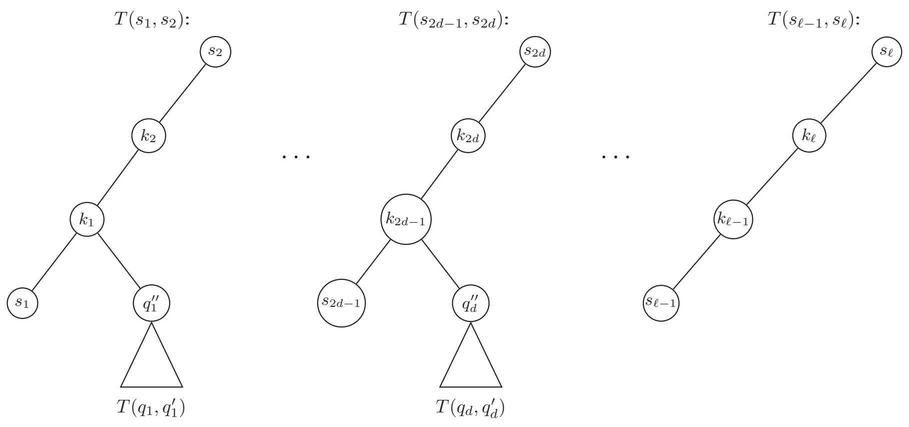

Figure 3: The corresponding path trees of Figure 2(a).

Suppose now that a vertex $v \notin P_1 \cup P_2$ has to bridge the two paths P_1 and P_2, i.e., v is connected to an endpoint, say, p_1', of P_1 and to an endpoint, say, p_2', of P_2. In this case, the vertex v merges the two path trees $T(p_1, p_1')$ and $T(p_2, p_2')$ into a new path tree having root v with children the roots p_1'' and p_2'' of the path trees $T(p_1, p_1')$ and $T(p_2, p_2')$, respectively.

Next we describe a procedure which generates a path tree corresponding to a subtree of $T_{blr}(G)$ rooted at an N-node. In general, the paths $P_i = [p_i, \dots, p_i']$ of a path cover $\mathcal{P}$ of a prime spider are considered to be path trees rooted at either p_i or p_i', $1 \le i \le |\mathcal{P}|$.

Let G be a P_4-sparse graph and let $T_{blr}(G)$ be its reduced leftist binary tree. Let t be an N-node and let $G(t) = (S, K, R)$ be a prime spider with $S = \{s_1, s_2, \dots, s_\ell\}$ and $K = \{k_1, k_2, \dots, k_\ell\}$, where $|S| = |K| = \ell$. In Section 3 we have described the form of the paths of a minimum path cover of $G(t)$. Every such path P_i is considered as a path tree rooted at a vertex $x \in S \cup K$, where x is the rightmost vertex of the path P_i, $1 \le i \le \lambda(t)$. More specifically, we distinguish two cases:

Case 1. $G(t) = (S, K, R)$ is a thin spider.

1.1 $R = \emptyset$. If ℓ is even then every path P_i, $1 \le i \le k$, of the path cover of $G[S \cup K]$ has the following form:

$$P_i = [s_{2i-1} k_{2i-1} k_{2i} s_{2i}], \qquad \text{for } 1 \le i \le k,$$

where $k = \lceil \frac{\ell}{2} \rceil$. Then each of the k path trees $T(s_{2i-1}, s_{2i})$ is rooted at vertex s_{2i} and each internal node has only a left child. If ℓ is odd then only the k-th path differs from the previous case; it has the form: $P_k = [s_\ell k_\ell]$. In this case the vertex s_ℓ is the root of the corresponding path tree $T(s_\ell, k_\ell)$ and the root s_ℓ has left child the vertex k_ℓ.

1.2 $R = \{r\}$. Let $q_1'', q_2'', \dots, q_d''$ be the roots of the path trees $T(q_1, q_1'), T(q_2, q_2'), \dots, T(q_d, q_d')$ of the graph $G[M(r)]$, where $d = \lambda(r)$. As described above, every vertex q_i'' becomes the right child of vertex k_{2i-1}, $1 \le i \le d$, in the path trees (see Fig. 3). Note that if ℓ is odd and $d \ge k$, then the root of the corresponding path tree of $G[M(r)]$ becomes the right child of vertex k_ℓ.

Case 2. $G(t) = (S, K, R)$ is a thick spider.

2.1 $R = \emptyset$. The paths described in Eqs. (8)–(9) are path trees $T(s_1, k_1)$ rooted at vertex k_1, which is the rightmost vertex of the Hamilton path of $G[S \cup K]$. Each internal node u of $T(s_1, k_1)$ has only a left child which is the previous node of u in the sequences of Eqs. (8)–(9); that is, vertex s_1 is the leftmost leaf of $T(s_1, k_1)$.

2.2 $R = \{r\}$. In this case, one path of a path cover of $G[M(r)]$ is connected to vertex k_1; see Eq. (10). Let $q_1'', q_2'', \dots, q_d''$ be the roots of the path trees $T(q_1, q_1'), T(q_2, q_2'), \dots, T(q_d, q_d')$ of $G[M(r)]$,

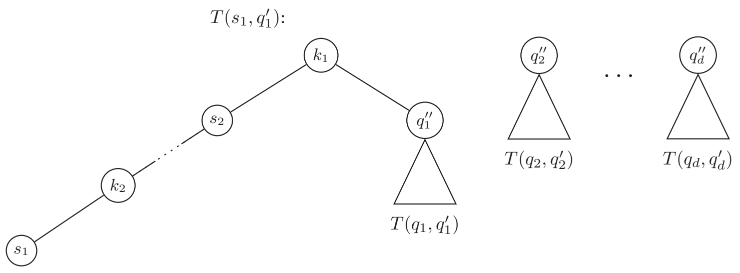

Figure 4: The corresponding path trees of Figure 2(b).

respectively, where $d = \lambda(r)$. Then the resulting path tree $T(s_1, q'_1)$ is similar to $T(s_1, k_1)$ of the previous case; the only difference is that vertex q''_1 becomes the right child of the root k_1 of $T(s_1, q'_1)$ (see Fig. 4).

In all the cases, the structure of the corresponding path trees is verified from the fact that the inorder traversal of the path trees returns the paths described in Eqs. (2)–(10).

5 Bracket Sequence on N-node

Let t be a node of $T_{blr}(G)$ and let $T(q_1, q'_1), T(q_2, q'_2), \dots, T(q_{\lambda(t)}, q'_{\lambda(t)})$ be the path trees of $G[M(t)]$. We denote by $B(t)$ the bracket sequence of node t (see Section 2.4). Note that a bracket matching of $B(t)$ corresponds to an edge of a path tree $T(q_i, q'_i)$, $1 \le i \le \lambda(t)$. Thus, all the bracket matchings of $B(t)$ generate the path trees of $G[M(t)]$.

Let $T(p_i, p'_i)$ be a path tree of $G[M(t)]$ and let a be a vertex of $T(p_i, p'_i)$. Then, we have: If a is the root of $T(p_i, p'_i)$ then there is an unmatched bracket $\overset{a^p}{[}$ in $B(t)$. If a has only a right child in $T(p_i, p'_i)$ then there is an unmatched bracket $\overset{a^l}{(}$ in $B(t)$, while if a has no right child in $T(p_i, p'_i)$ then there is an unmatched bracket $\overset{a^r}{(}$ in $B(t)$.

The cases where t is a P-node or an S-node of $T_{blr}(G)$ have been established in [20]. Here we study the case where t is an N-node of $T_{blr}(G)$ and we show the construction of an appropriate bracket sequence $B(t)$ using the above results.

Let t be an N-node of $T_{blr}(G)$ and let $G(t) = (S, K, R)$ be a prime spider with $S = \{s_1, s_2, \dots, s_\ell\}$ and $K = \{k_1, k_2, \dots, k_\ell\}$, where $|S| = |K| = \ell$. For simplicity, we associate a dummy node $\hat{t}$ to each N-node t and define $B(\hat{t})$ to be the bracket sequence of the vertices of the set $S \cup K$. In the case where $R = \{r\}$, let $B(r)$ be the bracket sequence of node r. For the bracket sequence $B(t)$ of the N-node t, we have:

$$B(t) = \begin{cases} B(\hat{t}) & \text{if } R = \emptyset, \\ B(r) \cdot B(\hat{t}) & \text{if } R = \{r\}, \end{cases} \tag{11}$$

where $B(r) \cdot B(\hat{t})$ denotes the concatenation of $B(r)$ and $B(\hat{t})$. Thus, given the bracket sequence $B(r)$, we need to construct the bracket sequence $B(\hat{t})$.

To simplify our description, for each path tree $T(p_i, p'_i)$ of $G[S \cup K]$, we denote by $\Lambda(i)$ the bracket sequence which has the property that its bracket matching generates the path tree $T(p_i, p'_i)$. We also denote by $\Pi(i)$ the bracket sequence of the unmatched brackets which correspond to the root of $T(p_i, p'_i)$ or to the vertices of $T(p_i, p'_i)$ that do not have a child (left or right) in $T(p_i, p'_i)$. For example, if $T(p_i, p'_i)$ contains only a vertex x, then $\Lambda(i) = \emptyset$ since there is no edge in the path tree, and $\Pi(i) = \overset{x^p}{[}\ \overset{x^l}{(}\ \overset{x^r}{(}$ since x is the root of the path tree and has neither left nor right child. For the special case where $G[M(t)]$ is a prime spider with $R = \{r\}$, i.e., r is a leaf in $T_{blr}(G)$, we associate the bracket sequence $B(r) = \overset{r^p}{[}\ \overset{r^l}{(}\ \overset{r^r}{(}$. As before, we distinguish two cases for the prime spider $G(t)$.

Case 1. $G(t) = (S, K, R)$ is a thin spider.

1.1 $R = \emptyset$. The graph $G[S \cup K]$ contains $k = \lceil \frac{\ell}{2} \rceil$ paths. Thus, if ℓ is even we associate to node t an appropriate bracket sequence $B(t)$ that generates $k = \frac{\ell}{2}$ path trees $T(q_1, q_1'), T(q_2, q_2'), \ldots, T(q_k, q_k')$. Each path tree $T(q_i, q_i')$ should have the following structure: (i) it consists of four vertices, (ii) it has root a vertex of S, and (iii) its internal vertices have only left children. Recall, that the inorder traversal of $T(q_i, q_i')$ produces the path $P_i = [s_{2i-1}k_{2i-1}k_{2i}s_{2i}]$, which is the i-th path of $\mathcal{P}$ in Eq. (2) (see also Section 4). The following two bracket sequences generate the path tree $T(q_i, q_i')$:

$$
\begin{aligned}
\Lambda(i) &= \overset{s^p_{2i-1}}{[}\ \overset{k^l_{2i-1}}{]}\ \overset{k^p_{2i-1}}{[}\ \overset{k^l_{2i}}{]}\ \overset{k^p_{2i}}{[}\ \overset{s^l_{2i}}{]}, \quad 1 \le i \le \frac{\ell}{2}, \quad \text{and} \\
\Pi(i) &= \overset{s^l_{2i-1}}{(}\ \overset{s^r_{2i-1}}{(}\ \overset{k^r_{2i-1}}{(}\ \overset{k^r_{2i}}{(}\ \overset{s^p_{2i}}{[}\ \overset{s^r_{2i}}{(}, \quad 1 \le i \le \frac{\ell}{2},
\end{aligned}
$$

By definition, the matching pair $\overset{s^p_{2i-1}}{[}\ \overset{k^l_{2i-1}}{]}$ in $\Lambda(i)$ makes vertex s_{2i-1} to be the left child of vertex k_{2i-1}, the matching pair $\overset{k^p_{2i-1}}{[}\ \overset{k^l_{2i}}{]}$ makes vertex k_{2i-1} to be the left child of vertex k_{2i}, while the matching pair $\overset{k^p_{2i}}{[}\ \overset{s^l_{2i}}{]}$ makes vertex k_{2i} the left child of vertex s_{2i}. The bracket sequence $\Pi(i)$ consists only of left brackets (round or square), because each vertex of the path tree $T(q_i, q_i')$ should be able to have two children and a parent. In detail, the brackets $\overset{s^l_{2i-1}}{(}$ and $\overset{s^r_{2i-1}}{(}$ mean that the vertex s_{2i-1} can have a left and a right child, respectively, since it is the leftmost leaf of the path tree $T(q_i, q_i')$. The vertices k_{2i-1} and k_{2i} already have only left children in $T(q_i, q_i')$ and thus we add brackets $\overset{k^r_{2i-1}}{(}$ and $\overset{k^r_{2i}}{(}$. As the root s_{2i} of the tree $T(q_i, q_i')$ can have a right child and a parent, we add brackets $\overset{s^p_{2i}}{[}$ and $\overset{s^r_{2i}}{(}$ in $\Pi(i)$.

If ℓ is odd, there are $k = \frac{\ell+1}{2}$ paths in the graph $G[M(t)]$. Thus, $\frac{\ell+1}{2}$ path trees are generated by $B(t)$, which produce the paths in Eq. (3). As in the previous case, the $\frac{\ell-1}{2}$ path trees $T(q_i, q_i')$ consist of four vertices and the root of the path tree $T(q_k, q_k')$ is the vertex s_ℓ which has vertex k_ℓ as a left child. Thus, we now need to distinguish the bracket sequence that generates path tree $T(q_k, q_k')$ from the rest $\frac{(\ell-1)}{2}$ path trees $T(q_i, q_i')$. Therefore, the following bracket sequences generate the corresponding path trees:

$$
\begin{aligned}
\Lambda(i) &= \overset{s^p_{2i-1}}{[}\ \overset{k^l_{2i-1}}{]}\ \overset{k^p_{2i-1}}{[}\ \overset{k^l_{2i}}{]}\ \overset{k^p_{2i}}{[}\ \overset{s^l_{2i}}{]}, \quad 1 \le i \le \frac{\ell-1}{2}, \\
\Lambda(\frac{\ell+1}{2}) &= \overset{k^p_\ell}{[}\ \overset{s^l_\ell}{]}, \\
\Pi(i) &= \overset{s^l_{2i-1}}{(}\ \overset{s^r_{2i-1}}{(}\ \overset{k^r_{2i-1}}{(}\ \overset{k^r_{2i}}{(}\ \overset{s^p_{2i}}{[}\ \overset{s^r_{2i}}{(}, \quad 1 \le i \le \frac{\ell-1}{2}, \quad \text{and} \\
\Pi(\frac{\ell+1}{2}) &= \overset{k^l_\ell}{(}\ \overset{k^r_\ell}{(}\ \overset{s^p_\ell}{[}\ \overset{s^r_\ell}{(},
\end{aligned}
$$

1.2 $R = \{r\}$. Since the graph $G[S \cup K]$ contains k paths, the paths of $G[M(t)]$ are obtained by joining the endpoints of the paths $Q_1, Q_2, \ldots, Q_k$ of $G[M(r)]$ with the endpoints of the k paths of $G[S \cup K]$. Thus, we need to connect the path trees $T(q_i, q_i')$ corresponding to the paths Q_i of $G[M(r)]$ with the path trees corresponding to the paths of $G[S \cup K]$ in such a way that the inorder traversal of the resulting path trees provides the correct paths. Thus, if $k \le d$ and ℓ is even we have the following

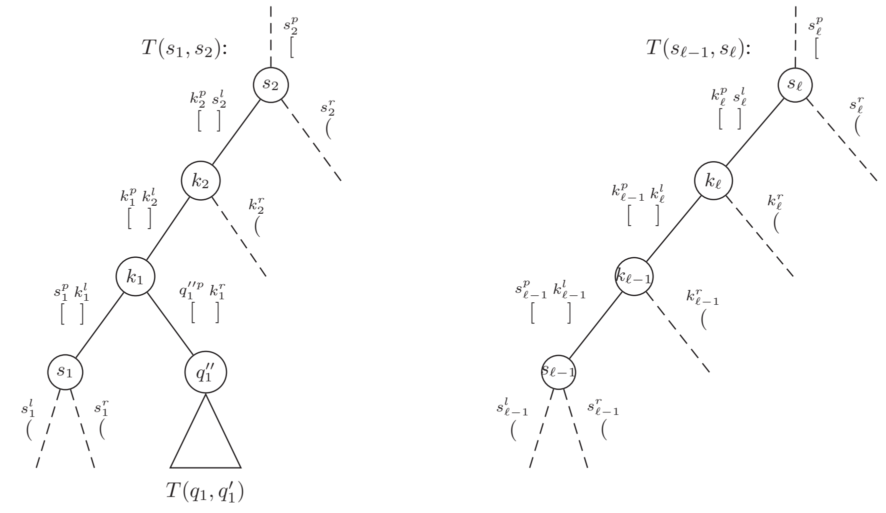

Figure 5: The brackets for the path trees $T(s_1, s_2)$ and $T(s_{\ell-1}, s_\ell)$ of Figure 3.

bracket sequences:

$$\Lambda(i) = \overset{s^p_{2i-1}}{[}\ \overset{k^l_{2i-1}}{]}\ \overset{k^p_{2i-1}}{[}\ \overset{k^l_{2i}}{]}\ \overset{k^p_{2i}}{[}\ \overset{s^l_{2i}}{]}\ \overset{k^r_{2i-1}}{]}\,, \quad 1 \le i \le \frac{\ell}{2}, \quad \text{and}$$

$$\Pi(i) = \overset{s^l_{2i-1}}{(}\ \overset{s^r_{2i-1}}{(}\ \overset{k^r_{2i}}{(}\ \overset{s^p_{2i}}{[}\ \overset{s^r_{2i}}{(}\,, \quad 1 \le i \le \frac{\ell}{2}.$$

Comparing $\Lambda(i)$ and $\Pi(i)$ with the corresponding sequences of the previous case where $R = \emptyset$, one can observe that the round bracket $\overset{k^r_{2i-1}}{(}$ does not appear in $\Pi(i)$ but it is appended to $\Lambda(i)$ as square bracket $\overset{k^r_{2i-1}}{]}$. This is because we need to connect the root of a path tree of $G[M(r)]$ as a right child of vertex k_{2i-1} of the i-th path tree.

Recall that, in the case where ℓ is odd there exists a path tree T_{q_k} consisting of two vertices: The root s_ℓ and its left child k_ℓ. Consequently, if $k \le d$, a path tree of $G[M(r)]$ is connected as a left child of vertex k_ℓ of T_{q_k} and, thus, we have the following bracket sequences:

$$\Lambda(i) = \overset{s^p_{2i-1}}{[}\ \overset{k^l_{2i-1}}{]}\ \overset{k^p_{2i-1}}{[}\ \overset{k^l_{2i}}{]}\ \overset{k^p_{2i}}{[}\ \overset{s^l_{2i}}{]}\ \overset{k^r_{2i-1}}{]}\,, \quad 1 \le i \le \frac{\ell-1}{2},$$

$$\Lambda\left(\frac{\ell+1}{2}\right) = \overset{k^p_\ell}{[}\ \overset{s^l_\ell}{]}\ \overset{k^l_\ell}{]}\,,$$

$$\Pi(i) = \overset{s^l_{2i-1}}{(}\ \overset{s^r_{2i-1}}{(}\ \overset{k^r_{2i}}{(}\ \overset{s^p_{2i}}{[}\ \overset{s^r_{2i}}{(}\,, \quad 1 \le i \le \frac{\ell-1}{2} \quad \text{and}$$

$$\Pi\left(\frac{\ell+1}{2}\right) = \overset{k^r_\ell}{(}\ \overset{s^p_\ell}{[}\ \overset{s^r_\ell}{(}\,.$$

If $k > d$ then the paths that occur in a minimum path cover of $G[M(t)]$ are obtained by joining d paths of $G[S \cup K]$ with the d paths of $G[M(r)]$. The rest $k - d$ paths of $G[S \cup K]$ remain the same. As a result, if $k > d$ and ℓ is even, the bracket $\overset{k^r_{2i-1}}{]}$ appears only in $\Lambda(i)$ where $1 \le i \le d$. Recall that, the bracket $\overset{k^r_{2i-1}}{]}$ means that vertex k_{2i-1} can have a right child which is the root of a path tree corresponding to a path of $G[M(r)]$; for example, see Fig. 5. Therefore, we have the following bracket sequences:

$$\begin{aligned}
\Lambda(i) &= \overset{s^p_{2i-1}}{[}\ \overset{k^l_{2i-1}}{]}\ \overset{k^p_{2i-1}}{[}\ \overset{k^l_{2i}}{]}\ \overset{k^p_{2i}}{[}\ \overset{s^l_{2i}}{]}\ \overset{k^r_{2i-1}}{]}\ ,\quad 1 \le i \le d,\\
\Lambda(i) &= \overset{s^p_{2i-1}}{[}\ \overset{k^l_{2i-1}}{]}\ \overset{k^p_{2i-1}}{[}\ \overset{k^l_{2i}}{]}\ \overset{k^p_{2i}}{[}\ \overset{s^l_{2i}}{]},\quad d < i \le \frac{\ell}{2},\\
\Pi(i) &= \overset{s^l_{2i-1}}{(}\ \overset{s^r_{2i-1}}{(}\ \overset{k^r_{2i}}{(}\ \overset{s^p_{2i}}{[}\ \overset{s^r_{2i}}{(},\quad 1 \le i \le d \text{ and}\\
\Pi(i) &= \overset{s^l_{2i-1}}{(}\ \overset{s^r_{2i-1}}{(}\ \overset{k^r_{2i-1}}{(}\ \overset{k^r_{2i}}{(}\ \overset{s^p_{2i}}{[}\ \overset{s^r_{2i}}{(},\quad d < i \le \frac{\ell}{2}.
\end{aligned}$$

In the case where $k > d$ and ℓ is odd, the sequence of brackets can be generated in a similar way; that is, d paths trees corresponding to paths of $G[S \cup K]$ are joined with the d path trees corresponding to paths of $G[M(r)]$. The rest $k - d$ path trees corresponding to paths of $G[S \cup K]$ remain the same. Recall that, when ℓ is odd there is a path tree generating a path of $G[S \cup K]$ which consists of only two vertices. In order to obtain the paths of $G[M(t)]$ we have the following bracket sequences:

$$\begin{aligned}
\Lambda(i) &= \overset{s^p_{2i-1}}{[}\ \overset{k^l_{2i-1}}{]}\ \overset{k^p_{2i-1}}{[}\ \overset{k^l_{2i}}{]}\ \overset{k^p_{2i}}{[}\ \overset{s^l_{2i}}{]}\ \overset{k^r_{2i-1}}{]}\ ,\quad 1 \le i \le d,\\
\Lambda(i) &= \overset{s^p_{2i-1}}{[}\ \overset{k^l_{2i-1}}{]}\ \overset{k^p_{2i-1}}{[}\ \overset{k^l_{2i}}{]}\ \overset{k^p_{2i}}{[}\ \overset{s^l_{2i}}{]},\quad d < i \le \frac{\ell-1}{2},\\
\Lambda(\frac{\ell+1}{2}) &= \overset{k^p_{\ell}}{[}\ \overset{s^l_{\ell}}{]},\\
\Pi(i) &= \overset{s^l_{2i-1}}{(}\ \overset{s^r_{2i-1}}{(}\ \overset{k^r_{2i}}{(}\ \overset{s^p_{2i}}{[}\ \overset{s^r_{2i}}{(},\quad 1 \le i \le d,\\
\Pi(i) &= \overset{s^l_{2i-1}}{(}\ \overset{s^r_{2i-1}}{(}\ \overset{k^r_{2i-1}}{(}\ \overset{k^r_{2i}}{(}\ \overset{s^p_{2i}}{[}\ \overset{s^r_{2i}}{(},\quad d < i \le \frac{\ell-1}{2}, \quad \text{and}\\
\Pi(\frac{\ell+1}{2}) &= \overset{k^l_{\ell}}{(}\ \overset{k^r_{\ell}}{(}\ \overset{s^p_{\ell}}{[}\ \overset{s^r_{\ell}}{(}\ .
\end{aligned}$$

Collecting all the previous results in both Case 1.1 and Case 1.2, we have that for ℓ even the bracket sequence that generates the path trees of $G[S \cup K]$ is the following:

$$B(\hat{t}) = \Lambda(1) \cdot \Lambda(2) \cdots \Lambda(\frac{\ell}{2} - 1) \cdot \Lambda(\frac{\ell}{2}) \cdot \Pi(1) \cdot \Pi(2) \cdots \Pi(\frac{\ell}{2} - 1) \cdot \Pi(\frac{\ell}{2}), \tag{12}$$

and for ℓ odd the bracket sequence is the following:

$$B(\hat{t}) = \Lambda(1) \cdot \Lambda(2) \cdots \Lambda(\frac{\ell-1}{2}) \cdot \Lambda(\frac{\ell+1}{2}) \cdot \Pi(1) \cdot \Pi(2) \cdots \Pi(\frac{\ell-1}{2}) \cdot \Pi(\frac{\ell+1}{2}). \tag{13}$$

We note that the order between any $\Lambda(i)$ and $\Lambda(j)$ in Eqs. (12)–(13) does not affect the correctness of the construction of the corresponding path trees. The same holds for any bracket sequences $\Pi(i)$ and $\Pi(j)$. Furthermore the number of brackets of each $\Lambda(i)$ and $\Pi(i)$ is at most seven and six, respectively.

Case 2. $G(t) = (S, K, R)$ is a thick spider.

2.1 $R = \emptyset$. The graph $G[S \cup K]$ is Hamiltonian and a Hamilton path can be constructed by edges which have one endpoint in S and the other endpoint in K. Therefore, the sequence of brackets described below result to a path tree. The first edge of the associated path is the one connecting the first vertex of S, say, s_1, with the second vertex of K, say, k_2. The second edge is that connecting the vertex k_2 with the vertex s_3, while the third edge connects the vertex s_3 with the vertex k_4, and so on. Recall that, by definition there is no edge from a vertex s_j of S to a vertex k_j of K. Also, note that the graph $G[S \cup K]$ does not have a unique Hamilton path; in fact every edge of a Hamilton path can have its endpoints in two arbitrary vertices s_i and k_j as long as $i \neq j$ and $1 < i, j \le \ell - 3$.

As described in Section 3, we need to distinguish two cases depending on the value of ℓ. If ℓ is an odd number, then we have the following bracket sequences:

$$\begin{array}{rcl}
\Lambda(1) & = & \overset{s_1^p}{\lbrack}, \quad \Lambda(\ell) \; = \; \overset{k_1^l}{\rbrack}, \\
\Lambda(i) & = & \overset{k_i^l}{\rbrack}\;\overset{k_i^p}{\lbrack}\;\overset{s_{i+1}^l}{\rbrack}\;\overset{s_{i+1}^p}{\lbrack}, \quad i = 2, 4, \ldots \ell-5, \ell-3, \\
\Lambda(i) & = & \overset{k_i^l}{\rbrack}\;\overset{k_i^p}{\lbrack}\;\overset{s_{i-1}^l}{\rbrack}\;\overset{s_{i-1}^p}{\lbrack}, \quad i = 3, 5, \ldots \ell-6, \ell-4, \\
\Lambda_{\text{mid}} & = & \overset{k_{\ell-1}^l}{\rbrack}\;\overset{k_{\ell-1}^p}{\lbrack}\;\overset{s_\ell^l}{\rbrack}\;\overset{s_\ell^p}{\lbrack}\;\overset{k_{\ell-2}^l}{\rbrack}\;\overset{k_{\ell-2}^p}{\lbrack}\;\overset{s_{\ell-1}^l}{\rbrack}\;\overset{s_{\ell-1}^p}{\lbrack}\;\overset{k_\ell^l}{\rbrack}\;\overset{k_\ell^p}{\lbrack}\;\overset{s_{\ell-3}^l}{\rbrack}\;\overset{s_{\ell-3}^p}{\lbrack}, \\
\Lambda_{\text{even}} & = & \Lambda(2) \cdot \Lambda(4) \cdots \Lambda(\ell-5) \cdot \Lambda(\ell-3), \\
\Lambda_{\text{odd}} & = & \Lambda(\ell-4) \cdot \Lambda(\ell-6) \cdots \Lambda(5) \cdot \Lambda(3),
\end{array}
\qquad
\begin{array}{rcl}
\Pi(1) & = & \overset{s_1^l}{(}\;\overset{s_1^r}{(}, \\
\Pi(i) & = & \overset{s_i^r}{(}\;\overset{k_i^r}{(}, \quad 2 \le i \le \ell-1, \\
\Pi(\ell) & = & \overset{k_1^p}{\lbrack}\;\overset{k_1^r}{(}.
\end{array}$$

The sequences $\Lambda(1)$ and $\Lambda(\ell)$ are used to identify the endpoints s_1 and k_1 of the Hamilton path. Each $\Lambda(i)$ consists of one matching pair of vertices k_i and $s_{i'}$, $i' \neq i$, such that the vertex k_i will become the left child of the vertex $s_{i'}$ in the corresponding path tree as described in Section 4. By concatenating two sequences $\Lambda(i) \cdot \Lambda(j)$ in the sequence Λ_{even}, where i and j are even numbers, the vertex $s_{i'}$ becomes the left child of the vertex k_j in the corresponding path tree. The same holds for the sequence Λ_{odd}, where, in this case, i and j are odd numbers. In this way we eventually connect the two edges $k_i s_{i'}$ and $k_j s_{j'}$ by adding the edge $s_{i'} k_j$.

Thus, the sequence $\Lambda(1) \cdot \Lambda_{\text{even}}$ constructs a path consisting of the odd-labelled vertices s_j and the even-labelled vertices k_i, $1 \le i, j \le \ell - 2$ (where i is even and j is odd). Let s_j be the endpoint of the path produced by the sequences $\Lambda(1) \cdot \Lambda_{\text{even}}$ having an odd label, $1 < j \le \ell - 2$. Then, we know that we can append a closing square bracket] corresponding to the even-labelled vertex $k_{\ell-1}$ since there is no bracket corresponding to the vertex $s_{\ell-1}$ in the sequence Λ_{even}. Now there is a matching between the bracket of the vertex $k_{\ell-1}$ and the bracket of the vertex s_j. The five matching pairs of brackets in Λ_{mid} construct the sequence of the corresponding vertices described in Eq. (8). In a similar manner the sequence $\Lambda_{\text{odd}} \cdot \Lambda(\ell)$ constructs a path of the even-labelled vertices s_i and the odd-labelled vertices k_j, $1 \le i, j < \ell - 3$ (where i is even and j is odd).

Note that the above sequences of brackets $\Lambda(i)$ will eventually produce a path tree which has nodes that have only a left child. In order to make each internal node able to connect with a right child we append $\Pi(i)$ which contains right brackets (round or square). In addition, vertex s_1 which is the leftmost leaf of the path tree can obtain both a left and a right child and, therefore, we use the brackets $\overset{s_1^l}{(}\;\overset{s_1^r}{(}$ in $\Pi(1)$. In addition, we use the square bracket $\overset{k_1^p}{\lbrack}$ in $\Pi(\ell)$ because vertex k_1 is the root of the path tree and it can become a child itself. Thus, the bracket sequence that generates the path trees of $G[M(t)]$ is the following:

$$B(\hat{t}) = \Lambda(1) \cdot \; \Lambda_{\text{even}} \cdot \Lambda_{\text{mid}} \cdot \Lambda_{\text{odd}} \; \cdot \Lambda(\ell) \cdot \Pi(1) \cdot \Pi(2) \cdots \Pi(\ell). \tag{14}$$

In the case where ℓ is even, we have the following bracket sequences:

$$\begin{array}{rcl}
\Lambda(1) & = & \overset{s_1^p}{\lbrack}, \quad \Lambda(\ell) \; = \; \overset{k_1^l}{\rbrack}, \\
\Lambda(i) & = & \overset{k_i^l}{\rbrack}\;\overset{k_i^p}{\lbrack}\;\overset{s_{i+1}^l}{\rbrack}\;\overset{s_{i+1}^p}{\lbrack}, \quad i = 2, 4, \ldots \ell-4, \ell-2, \\
\Lambda(i) & = & \overset{k_i^l}{\rbrack}\;\overset{k_i^p}{\lbrack}\;\overset{s_{i-1}^l}{\rbrack}\;\overset{s_{i-1}^p}{\lbrack}, \quad i = 3, 5, \ldots \ell-5, \ell-3, \\
\Lambda_{\text{mid}} & = & \overset{k_\ell^l}{\rbrack}\;\overset{k_\ell^p}{\lbrack}\;\overset{s_{\ell-2}^l}{\rbrack}\;\overset{s_{\ell-2}^p}{\lbrack}\;\overset{k_{\ell-1}^l}{\rbrack}\;\overset{k_{\ell-1}^p}{\lbrack}\;\overset{s_\ell^l}{\rbrack}\;\overset{s_\ell^p}{\lbrack}, \\
\Lambda_{\text{even}} & = & \Lambda(2) \cdot \Lambda(4) \cdots \Lambda(\ell-4) \cdot \Lambda(\ell-2), \\
\Lambda_{\text{odd}} & = & \Lambda(\ell-3) \cdot \Lambda(\ell-5) \cdots \Lambda(5) \cdot \Lambda(3),
\end{array}
\qquad
\begin{array}{rcl}
\Pi(1) & = & \overset{s_1^l}{(}\;\overset{s_1^r}{(}, \\
\Pi(i) & = & \overset{s_i^r}{(}\;\overset{k_i^r}{(}, \quad 2 \le i \le \ell-1, \\
\Pi(\ell) & = & \overset{k_1^p}{\lbrack}\;\overset{k_1^r}{(}.
\end{array}$$

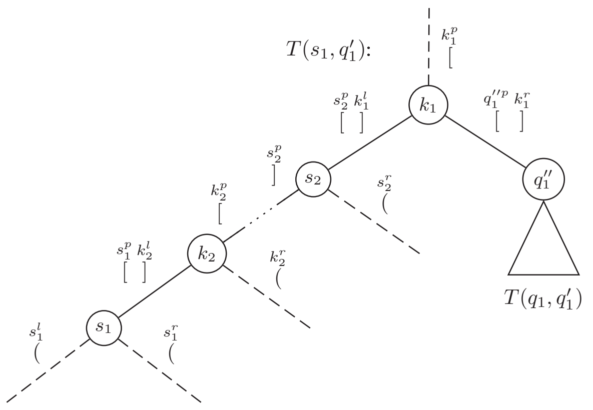

Figure 6: The brackets for the path tree $T(s_1, q_1')$ of Figure 4.

The above sequences are similar to those of the case where ℓ is an odd number. The only difference is in sequence Λ_{mid} in which the three matching pairs can easily be verified by Eq. (9). As described above, we need to make each internal node able to connect with a right child, and therefore we append to the sequence $\Lambda(1) \cdot \Lambda_{\text{even}} \cdot \Lambda_{\text{mid}} \cdot \Lambda_{\text{odd}}$ the bracket sequences $\Pi(i)$, $1 \leq i \leq \ell$. Thus, the bracket sequence $B(\hat{t})$ of Eq. (14) generates the path trees of $G[S \cup K]$.

2.2 $R = \{r\}$. The Hamilton path of $G[S \cup K]$ is connected to the path Q_1 of $G[M(r)]$. Consequently, the bracket sequences $\Lambda(i)$ and $\Pi(i)$ are similar to those described above in the case where R is empty (for example, see Fig. 6). The only difference is that we need bracket $\overset{k_1^r}{]}$ in order to make the path tree corresponding to the path Q_1 of $G[M(r)]$ be the right child of the root vertex, that is vertex k_1, of the path tree corresponding to the Hamilton path of $G[S \cup K]$. Hence, only the two following brackets sequences are different from the previous cases:

$$\Lambda(\ell) \;=\; \overset{k_1^l}{]}\,\overset{k_1^r}{]}\,, \qquad\qquad \text{and} \qquad\qquad \Pi(\ell) \;=\; \overset{k_1^p}{[}\;.$$

Note that in the sequence of brackets $\Pi(\ell)$ we now have one less bracket, compared with the case where R is empty. The missing bracket is $\overset{k_1^r}{(}$ because vertex k_1 already has a right child. As a result, we have that the bracket sequence $B(\hat{t})$ that generates the path trees of $G[S \cup K]$ is described in Eq. (14).

We note that the sequences $\Lambda(2)$, $\Lambda(4)$, $\ldots$, $\Lambda(\ell-4)$, $\Lambda(\ell-2)$ can appear in any order in the bracket sequence Λ_{even}; similarly, the sequences $\Lambda(\ell-3)$, $\Lambda(\ell-5)$, $\ldots$, $\Lambda(5)$, $\Lambda(3)$ can appear in any order in Λ_{odd}. The same holds for the sequences $\Pi(i)$, $1 \leq i \leq \ell$; that is, they can appear in any order in $B(\hat{t})$ after the sequence $\Lambda(\ell)$. Furthermore, the sequences $\Lambda(1)$, Λ_{mid}, $\Lambda(\ell)$, $\Pi(1)$ and $\Pi(\ell)$ contain a constant number of brackets.

6 The Algorithm

In this section we present an optimal parallel algorithm for the minimum path cover problem on P_4-sparse graphs. Our algorithm takes as input a P_4-sparse graph G on n vertices and its modular decomposition tree $T(G)$, and finds the paths of a minimum path cover in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM model.

Let us first sketch the workings of our algorithm. Initially, we compute the binarized md-tree $T_b(G)$; recall that we only binarize each subtree of $T(G)$ rooted at a P-node or an S-node. In order to make the binarized tree leftist, we compute the number $L(t)$ for each internal node t of $T_b(G)$. We next compute the number $\lambda(t)$ of paths in the minimum path cover of $G(M[t])$ for each internal node t of $T_{bl}(G)$. Before assigning any bracket sequence, we first compute the reduced leftist binarized tree $T_{blr}(G)$, and, then, for every internal N-node t of $T_{blr}(G)$ we compute a bracket sequence $B(t)$ based on the vertices of $S \cup K$ which are leaves and children of t in $T_{blr}(G)$. Using this information, we generate a bracket sequence $B(t_{root})$ of the root t_{root} of $T_{blr}(G)$. Finally, we construct the path trees by finding all matchings of $B(t_{root})$ and then we return vertex sequences produced by the inorder traversal of the path trees. Recall that the latter corresponds to a minimum path cover of G. We next give the detailed description of the algorithm.

Parallel_Minimum_Path_Cover

Input: A P_4-sparse graph G and its modular decomposition tree $T(G)$;

Output: A minimum path cover of the P_4-sparse graph G;

1. Compute the binarized tree $T_b(G)$ of $T(G)$ and the number $L(t)$ for each internal node t of $T_b(G)$, and then the leftist binarized tree $T_{bl}(G)$;
2. Compute the number of paths $\lambda(t)$ in the minimum path cover of $G[M(t)]$ for each internal node t of $T_{bl}(G)$, and then the reduced leftist binarized tree $T_{blr}(G)$;
3. Generate the sequence of brackets $B(t_{root})$ of the root t_{root} of $T_{blr}(G)$ based on [20] (for P-nodes or S-nodes) and on Eqs (11)–(14) of Section 5 (for N-nodes);
4. Construct the path trees by finding all matchings of $B(t_{root})$;
5. Return a minimum path cover from the path trees;

We mention here that Step 4 needs a post-processing function. The bracket assignment procedure for an S-node, as described in [20], may lead to path trees which result to paths having edges that do not appear in the graph G. This post-processing function corrects any *illegal* path tree which has been produced by the children of the S-node. The correct path trees are calculated with a detailed technique, as a post-processing step, proposed in [20].

Time and Processor Complexity. Next, we analyze the time and processor complexity of the proposed algorithm on the PRAM model; for details on PRAM techniques, see [13, 23]. We assume that the input graph G and its modular decomposition tree $T(G)$ are given in adjacency list representations.

Steps 1 and 2 are executed in $O(\log n)$ time using $O(n/\log n)$ EREW processors (see Lemma 2.4 and Lemma 2.5). Step 3 computes the sequence of brackets $B(t_{root})$ of the root t_{root} of $T_{blr}(G)$. This computation can be efficiently done by first computing the bracket sequence $B(t)$ of each internal node t of $T_{blr}(G)$ (see results of [20] and Eqs (11)–(14)), and then contracting the tree $T_{blr}(G)$ into a three-node tree consisting of the root and two nodes. We use standard parallel techniques (i.e., prefix sums, array packing, concatenation) to construct the bracket sequences in each internal node, and the *rake* operation to reduce the tree $T_{blr}(G)$ into a three-node tree (see [13, 23]). Thus, this step can be computed in $O(\log n)$ time using $O(n/\log n)$ EREW processors. Step 4 can be performed in $O(\log n)$ time using $O(n/\log n)$ EREW processors by finding the matching pairs of the sequence $B(t_{root})$, where t_{root} is the root of $T_{blr}(G)$. Note that $B(t_{root})$ has $O(n)$ brackets since the tree $T_{blr}(G)$ has $O(n)$ nodes, and also that each assignment of bracket of an internal node $u \in T_{blr}(G)$ does not exceed the number of its children in $T_{blr}(G)$. Step 5 is accomplished with Euler-tours on the path trees. Since the nodes of the path trees are the vertices of the input graph G, it follows that they contain n nodes, and, thus, Step 5 can be executed in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM model.

By Corollary 2.1, the path cover problem of an n-vertex P_4-sparse graph represented by its md-tree must take $\Omega(\log n)$ time on the CREW PRAM model. Our algorithm Parallel_Minimum_Path_Cover shows that this time lower bound is tight for the class of P_4-sparse graphs. Summarizing, we obtain the following result.

Theorem 6.1. *Let G be a P_4-sparse graph on n vertices and let $T(G)$ be its modular decomposition tree. A minimum path cover of G can be computed time- and work-optimally in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors.*

7 Other Classes of Graphs

In this section we extend our results to a proper superclass of P_4-sparse graphs, namely the P_4-tidy graphs. The class of P_4-tidy graphs was introduced by I. Rusu in order to illustrate the notion of P_4-domination in perfect graphs; see [9].

A graph G is P_4*-tidy* if for any induced P_4, say, $abcd$, there exists at most one vertex $v \in V(G) - \{a,b,c,d\}$ such that the subgraph $G[\{a,b,c,d,v\}]$ has at least two P_4's (i.e., the P_4 has at most one *partner*). The P_4-tidy graphs strictly contain the cographs, P_4-reducible, P_4-sparse, P_4-extendible, and P_4-lite graphs. The P_4-lite graphs were defined by Jamison and Olariu in [14]: A graph G is P_4*-lite* if every induced subgraph H of G with at most six vertices either contains at most two P_4's, or is a 3-sun, or is the complement of a 3-sun (a *3-sun* is a thick spider on six vertices with $R = \emptyset$). They remark that every P_4-sparse graph is P_4-lite and prove that every P_4-lite graph is brittle and, thus perfect. We mention here that the P_4-lite graphs coincide with the C_5-free P_4-tidy graphs.

The modular decomposition of P_4-tidy graphs has a structural property, as in the case of P_4-sparse graphs (Lemma 2.3), which is shown by the following result (Theorem 3.2 in [9]):

Theorem 7.1. (Giakoumakis et al. [9]): *Let G be a graph and let $T(G)$ be its modular decomposition tree. The graph G is P_4-tidy iff for every N-node t of $T(G)$, $G(t)$ is either*

(i) a P_5, a $\overline{P_5}$, or a C_5, and no vertex of $G(t)$ is an internal node in $T(G)$, or

(ii) a prime spider (S,K,R) with at most one vertex of $S \cup K$ which is an internal node having two children which are leaves in $T(G)$.

The above theorem implies that every N-node t of the md-tree $T(G)$ of a P_4-tidy graph G has either five vertices which are leaves in $T(G)$ and $G(t) \in \{P_5, \overline{P_5}, C_5\}$ or $G(t) = (S,K,R)$ is a prime spider with at most one vertex, say, t, of $S \cup K$ replaced by a $2K_1$ or a K_2 (i.e., t is a P- or S-node with two children which are leaves in $T(G)$). Based on this result, the path cover problem for the class of P_4-tidy graphs was solved in sequential linear time by describing the paths that occur in every internal node of $T(G)$ [9].

Let G be a P_4-tidy graph, $T(G)$ be its md-tree and t be an N-node of $T(G)$. Here we describe the bracket sequence $B(t)$ that generates the corresponding path trees which produce the paths of a minimum path cover of $G[M(t)]$. According to Theorem 7.1 we distinguish the following cases.

Case A1. $G(t)$ is a P_5, a $\overline{P_5}$ or a C_5.

It is easy to see that $\lambda(t) = 1$, since a path on five vertices (not necessarily chordless) occurs in the three possible graphs (see also [9]). Thus, $\lambda(t)$ can be computed optimally and, hence, it is possible to construct optimally the tree $T_{blr}(G)$. Let $v_1v_2v_3v_4v_5$ be such a path of $G[M(t)]$. Then the corresponding path tree $T(v_1, v_5)$ is constructed as in the case where we have a thick spider with $R = \emptyset$ (Section 4, Case 2.1). Thus, it can be viewed as the Hamilton path of a thick spider and the corresponding bracket sequence $B(t)$ is given in details in Section 5, Case 2.1.

Case A2. $G(t) = (S, K, R)$ is a prime spider.

In this case and if no vertex of $S \cup K$ is replaced by an S_2 or a K_2, the paths of a minimum path cover of $G[M(t)]$ are obtained by considering the cases of a prime spider of a P_4-sparse graph (Section 3). Here, we also have to consider the fact that a vertex of $S \cup K$ can be substituted by an S_2 or a K_2 which will eventually change the value of $\lambda(t)$. Notice that in any case we have to distinguish whether we have a thin or a thick spider and whether the set $R = \emptyset$ or not. For example, if $G(t)$ is a thick spider with $R = \emptyset$ and a vertex of S is replaced by the two vertices of S_2, say, s_1 and s'_1, then the paths of a minimum path cover of $G[M(t)]$ are a Hamilton path (see Section 3) and an isolated vertex (one of the two vertices, say, s'_1, of the set S_2). Thus, in this case, the value of $\lambda(t)$ is equal to the value of the case where $G(t)$ is a thick spider (see Eq. 1) plus one.

In a similar manner, we can consider all the other cases and establish the corresponding paths of a minimum path cover of $G[M(t)]$; details about the $\lambda(t)$ and the paths in a minimum path cover of $G[M(t)]$ can be found in [9]. In order to obtain the paths of $G[M(t)]$ we can construct the path trees and the bracket sequence $B(t)$ in a way similar to that described in Section 5.

Based on the above description, we can show the following result for the class of P_4-tidy graphs: Given a P_4-tidy graph G on n vertices and its modular decomposition tree $T(G)$, a minimum path cover of G can be optimally computed in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM model.

8 Concluding Remarks

We have presented an optimal parallel algorithm for solving the minimum path cover problem on P_4-sparse graphs; our algorithm runs in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM model, and thus is time- and work-optimal due to the results of [20]. We also described the way we can solve the same problem on the class of P_4-tidy graphs, which forms a proper superclass of P_4-sparse graphs.

An interesting open question would be to see if similar techniques can be efficiently used for finding a minimum path cover for other classes of graphs and solving other related algorithmic problems such as the *terminal path cover* problem: Given a graph G and a subset S of its vertices, the terminal path cover problem is to find a minimum path cover $\mathcal{P}$ of the graph G such that all the vertices of S are endpoints of the paths in $\mathcal{P}$.

References

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

[2] G.S. Adhar and S. Peng, Parallel algorithm for path covering, Hamiltonian path, and Hamiltonian cycle in cographs, *Int'l Conference on Parallel Processing*, Vol. III (1990) 364–365.

[3] K. Asdre, S.D. Nikolopoulos and Ch. Papadopoulos, Optimal algorithms for the path cover problem on P_4-sparse graphs, *Workshop on Graphs and Combinatorial Optimization (CTW'05)*, Vol. 1 (2005) 79–83.

[4] A. Brandstädt, V.B. Le, and J. Spinrad, *Graph Classes – A Survey*, SIAM Monographs in Discrete Mathematics and Applications, SIAM, Philadelphia, 1999.

[5] D.G. Corneil, Y. Perl, and L.K. Stewart, A linear recognition algorithm for cographs, *SIAM J. Comput.* **14** (1985) 926–984.

[6] E. Dahlhaus, Efficient parallel modular decomposition, *21st Int'l Workshop on Graph Theoretic Concepts in Computer Science (WG'95)*, LNCS **1017** (1995) 290–302.

[7] E. Dahlhaus, J. Gustedt and R.M. McConnell, Efficient and practical algorithms for sequential modular decomposition, *J. Algorithms* **41** (2001) 360–387.

[8] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, W.H. Freeman, San Francisco, 1979.

[9] V. Giakoumakis, F. Roussel and H. Thuillier, On P_4-tidy graphs, *Discrete Math. and Theoret. Comput. Science* **1** (1997) 17–41.

[10] V. Giakoumakis and J-M. Vanherpe, On extended P_4-reducible and P_4-sparse graphs, *Theoret. Comput. Science* **180** (1997) 269–286.

[11] C. Hoàng, Perfect graphs, PhD Thesis, McGill University, Montreal, Canada, 1985.

[12] W. Hochstättler and G. Tinhofer, Hammiltonicity in graphs with few P_4's, *Computing* **54** (1995) 213–225.

[13] J. Jájá, An introduction to parallel algorithms, Addison-Wesley, Reading, MA, 1992.

[14] B. Jamison and S. Olariu, A new class of brittle graphs, *Studies Appl. Math.* **81** (1989) 89–92.

[15] H. Lerchs, On cliques and kernels, Tech. Report, Department of Computer Science, University of Toronto, March 1971.

[16] R. Lin and S. Olariu, A fast parallel algorithm to recognize P4-sparse graphs, *Discrete Appl. Math.* **81** (1998) 191–215.

[17] R. Lin, S. Olariu and G. Pruesse, An optimal path cover algorithm for cographs, *Comput. Math. Appl.* **30** (1995) 75–83.

[18] R. Lin, S. Olariu, J.L. Schwing and J. Zhang, A fast EREW algorithm for minimum path cover and hamiltonicity for cographs, *Parallel Algorithms Appl.* **2** (1994) 99–113.

[19] R.M. McConnell and J. Spinrad, Modular decomposition and transitive orientation, *Discrete Math.* **201** (1999) 189–241.

[20] K. Nakano, S. Olariu and A.Y. Zomaya, A time-optimal solution for the path cover problem on cographs, *Theoret. Comput. Science* **290** (2003) 1541–1556.

[21] S.D. Nikolopoulos, Parallel algorithms for Hamiltonian problems on quasi-threshold graphs, *J. Parallel and Distributed Computing* **64** (2004) 48–67.

[22] I. Parfenoff, An efficient parallel algorithm for maximum matching for some classes of graphs, *J. Parallel and Distributed Computing* **52** (1998) 96–108.

[23] J. Reif (editor), *Synthesis of Parallel Algorithms*, Morgan Kaufmann, Inc., 1993.