# Characterization of Efficiently Parallel Solvable Problems on a Class of Decomposable Graphs

Sun-Yuan Hsieh

Department of Computer Science and Information Engineering
National Cheng Kung University
No 1. University Road, Tainan 701, Taiwan
hsiehsy@mail.ncku.edu.tw

**Abstract.** In this paper, we sketch characteristics of those problems which can be systematically solved on decomposable graphs. Trees, series-parallel graphs, outerplanar graphs, and bandwidth-$k$ graphs all belong to decomposable graphs. Let $T_d(|V|,|E|)$ and $P_d(|V|,|E|)$ denote the time complexity and processor complexity required to construct a parse tree representation $T_G$ for a decomposable $G=(V,E)$ on a PRAM model $M_d$. We define a general problem-solving paradigm to solve a wide class of subgraph optimization problems on decomposable graphs in $O(T_d(|V|,|E|)+\log|V(T_G)|)$ time using $O(P_d(|V|,|E|)+|V(T_G)|/\log|V(T_G)|)$ processors on $M_d$. By using our paradigm, we show the following parallel complexities: (a) The maximum independent set problem on trees can be solved in $O(\log|V|)$ time using $O(|V|/\log|V|)$ processors on an EREW PRAM. (b) The maximum matching problem on series-parallel graphs can be solved in $O(\log|E|)$ time using $O(|E|/\log|E|)$ processors on an EREW PRAM. (c) The efficient domination problem on series-parallel graphs can be solved in $O(\log|E|)$ time using $O(|E|/\log|E|)$ processors on an EREW PRAM.

## 1 Introduction

A class of graphs is *recursive* if every graph of the class can be constructed by a finite number of applications of *composition operations* starting with a finite set of *basis* graphs. The recursive class $\Gamma$ of graphs is said to be *decomposable* if each graph in $\Gamma$ has a set of some specified vertices called *terminals*, and each composition operation is defined in terms of certain primitive operations on terminals. Trees, series-parallel graphs, outerplanar graphs, protoHalin graphs, and bandwidth-$k$ graphs are all decomposable graphs [3]. Also, every decomposable graph has a fixed upper bound on the *treewidth* of the graphs in the class, and graphs with treewidth at most $k$ for fixed $k$ are partial $k$-trees [8].

Properties of decomposable graphs are studied by many researchers [2,3,7,8, 9,11,12] which resulted in sequential algorithms to solve quite a few interesting graph-theoretical problems on this special class of graphs. However, there are few results in the viewpoint of parallel computation. Given a graph problem, we say it belongs to the class of *subgraph optimization* problem if the object of this

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3036, pp. 223–230, 2004.

problem is to find a subgraph of the input graph to satisfy the given properties which includes an optimization constraint. For example, the problem of finding a maximum independent set is a subgraph optimization problem.

In this paper, we propose a different parallel strategy on the deterministic parallel random access machine (PRAM) [6]. Given a decomposable graph represented by its parse tree form, we define a class of subgraph optimization problems, called the *$(k, \Theta)$-regular problem*, and show such a class of problems can be efficiently parallelized by applying the binary tree contraction technique to the given parse tree. Let $T_d(|V|, |E|)$ and $P_d(|V|, |E|)$ denote the time complexity and processor complexity required to construct a parse tree $T_G$ of a decomposable graph $G = (V, E)$ on a PRAM model $M_d$. We show that a $(k, \Theta)$-regular problem can be solved in $O(T_d(|V|, |E|) + \log |V(T_G)|)$ time using $O(P_d(|V|, |E|) + |V(T_G)|/\log |V(T_G)|)$ processors on $M_d$. Moreover, each $(k, \Theta)$-regular problem can be solved in $O(\log |V(T_G)|)$ time using $O(|V(T_G)|/\log |V(T_G)|)$ processors on an EREW PRAM if $T_G$ is given to be an input instance. Based on the technique, we obtain the following results: (a) The maximum independent set problem on trees can be solved in $O(\log |V|)$ time using $O(|V|/\log |V|)$ processors on an EREW PRAM, (b) The maximum matching problem can be solved in $O(\log |E| \log^* |E|)$ time using $O(|E|/\log |E| \log^* |E|)$ processors on an EREW PRAM, and (c) The efficient domination problem on series-parallel graphs can be solved in $O(\log |E| \log^* |E|)$ time using $O(|E|/\log |E| \log^* |E|)$ processors on an EREW PRAM. Given a parse tree of a series-parallel graph, the problems in (b) and (c) can be optimally solved in $O(\log |E|)$ time using $O(|E|/\log |E|)$ processors on an EREW PRAM. To our knowledge, no NC algorithm exists for solving the problem in (c) in the literature.

## 2 Preliminaries

This paper considers finite, simple[1] and undirected graphs $G = (V, E)$, where $V$ and $E$ are the vertex and edge sets of $G$, respectively. Let $n = |V|$ and $m = |E|$. For two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the *union of $G_1$ and $G_2$*, denoted by $G_1 \cup G_2$, is the graph $(V_1 \cup V_2, E_1 \cup E_2)$. We say that a graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. Given a set $V' \subseteq V$, the subgraph of $G$ *induced* by $V'$ is the graph $G' = (V', E')$, where $E' = \{(u, v) \in E |\ u, v \in V'\}$. Let $G[X]$ denote the subgraph of $G$ induced by $X \subseteq V$. For a vertex $v \in V$ of a graph $G = (V, E)$, the *neighborhood of $v$* is $N_G(v) = \{u \in V |\ (u, v) \in E\}$ and the *closed neighborhood of $v$* is $N_G[v] = N_G(v) \cup \{v\}$. The subscript $G$ in the notations used in this paper can be omitted when no ambiguity arises. Given a node $v$ in a rooted tree $T$, let $T(v)$ be a subtree of $T$ rooted at $v$. For graph-theoretic terminologies and notations not mentioned here, see [5].

We follow the notations used in [8] to define the class of decomposable graphs.

[1] We only consider simple graphs in this paper although some of the results also apply to multigraphs.

**Definition 1.** Let $G = (V, E, S)$ be a graph with vertex $V$, edge set $E$, and an ordered list $S$ of $t$ *terminals* chosen from $V$ for some fixed integer $t$. We note that the elements of $S$ are not necessary distinct.

**(1)** Let $B = \{B_1, B_2, \ldots, B_l\}$ be a finite set of *basis graphs*, where each $B_i$ is a finite graph having an ordered list of $t$ (not necessary distinct) terminals.

**(2)** Let $O = \{*_1, *_2, \ldots, *_q\}$ be a finite set of binary rules of *composition*, whereby two graphs $G_i = (V_i, E_i, S_i)$ and $G_j = (V_j, E_j, S_j)$ can be combined to produce new graphs $G_i *_c G_j$, $1 \leq c \leq q$. Each rule of composition $*_c$ consists of three suboperations on the terminals $S_i$ and $S_j$:

- **(i)** Choose a subset ${S_i}'$ of distinct terminals from the list $S_i$ and identify each $x \in {S_i}'$ with a unique $y \in S_j$. Let ${S_j}'$ denote the subset of affected terminals from the list $S_j$.
- **(ii)** Add any subset of the edges $\{(x, y) | x \in \overline{{S_i}'}, y \in \overline{{S_j}'}\}$ to $G_i *_c G_j$, where $\overline{{S_i}'}$ is the subset of terminals in the list $S_i$ but not in ${S_i}'$, and $\overline{{S_j}'}$ is defined similarly.
- **(iii)** Select an ordered list of $t$ (not necessarily distinct) terminals from the list $S_i$ and $S_j$ to the terminals of $G_i *_c G_j$.

**(3)** The class $\Gamma$ of decomposable graphs is recursively defined as follows:

- **(i)** Any $B_i \in B$ is in $\Gamma$.
- **(ii)** If $G_i$ and $G_j$ are in $\Gamma$ and $*_c$ is an operation in $O$, then the graph $G_i *_c G_j$ is also in $\Gamma$.

**Definition 2.** Let $\Gamma$ be the class of decomposable graphs. The *parse tree* $T_G$ of a graph $G \in \Gamma$ is a tree in which the leaves correspond to the basis graphs from which $G$ is constructed, and each internal node represents the result of applying a composition operation to the graphs represented by the subtrees rooted at its children. Let $G_v$ be the subgraph of $G$ corresponding to a node $v$ of a parse tree. Note that $T_G(v)$ is a parse tree of $G_v$.

## 3 A General Problem-Solving Paradigm

### 3.1 The $(k, \Theta)$-Parse Tree

Given a graph $G$, let $\mathcal{U}_{V(G)}$ (respectively, $\mathcal{U}_{E(G)}$) be the set consisting of all subsets of $V(G)$ (respectively, $E(G)$). Given $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_l\}$, where $Q_i \in \mathcal{U}_{V(G)}$ (respectively, $Q_i \in \mathcal{U}_{E(G)}$), we define $\textsc{Min}_v$ (respectively, $\textsc{Min}_e$) to be an operator on $\mathcal{Q}$ that returns a minimum-cardinality set $Q_j$ for some $1 \leq j \leq l$. The operators $\textsc{Max}_v$ (respectively, $\textsc{Max}_e$) can be defined similarly. For two lists $L_1 = \langle l_1, l_2, \ldots, l_i \rangle$ and ${L_1}' = \langle {l_1}', {l_2}', \ldots, {l_j}' \rangle$, we define the *concatenation of* $L_1$ *and* ${L_1}'$, denoted by $L_1 \bullet {L_1}'$, to be the list $\langle l_1, l_2, \ldots, l_i, {l_1}', {l_2}', \ldots, {l_j}' \rangle$.

**Definition 3.** Let $G = (V, E)$ be a decomposable graph and let $T_G$ be a parse tree of $G$. Given a positive integer $k$, and an operator $\Theta \in \{\textsc{Min}_v, \textsc{Min}_e, \textsc{Max}_v, \textsc{Max}_e\}$, $T_G$ is a $(k, \Theta)$-*parse tree of* $G$ if the following conditions hold. Let $v$ be a node of $T_G$ and let $N_i$ be the set of integers from 1 to $i$.

(1) If $v$ is an internal node, then it is associated with $k$ integers $a_{v,1}, a_{v,2}, \ldots, a_{v,k}$

from $N_k$, and the following $2k$ functions $f_i : \{v\} \times N_{a_{v,i}} \mapsto N_k$ and $g_i : \{v\} \times N_{a_{v,i}} \mapsto N_k$, $1 \le i \le k$.
(2) Node $v$ is also associated with a list of $k$ subgraphs[2] $R_v = \langle R_{v,1}, R_{v,2}, \ldots, R_{v,k} \rangle$, called the *target subgraphs of* $v$, which are defined as follows.

CASE 1: $v$ is a leaf. $R_v$ is a list of $k$ subgraphs selected from $\mathcal{U}_{V(G_v)}$ (respectively, $\mathcal{U}_{E(G_v)}$) if $\Theta \in \{\text{MIN}_v, \text{MAX}_v\}$ (respectively, $\Theta \in \{\text{MIN}_e, \text{MAX}_e\}$).
CASE 2: $v$ is an internal node. Let $u$ and $w$ be two children of $v$. Then, $R_{v,i} = \Theta\{R_{u,f_i(u,1)} \cup R_{w,g_i(w,1)}, R_{u,f_i(u,2)} \cup R_{w,g_i(w,2)}, \ldots, R_{u,f_i(u,a_{v,i})} \cup R_{w,g_i(w,a_{v,i})}\}$, where $1 \le i \le k$.

**Definition 4.** Let $T_G$ be a $(k,\Theta)$-parse tree. The $(k,\Theta)$*-parse tree problem* is the problem to find the $k$ target subgraphs of the root of $T_G$.

**Lemma 1.** *The* $(k,\Theta)$*-parse tree problem can be solved in* $O(k^2 n)$ *time, where* $n$ *is the number of vertices of the given tree.*

### 3.2 Parallel Complexities of the $(k,\Theta)$-Regular Problem

In this section, we apply the binary tree contraction technique described in [1] to parallelize the $(k,\Theta)$-regular problem. This technique recursively applies two operations, *prune* and *bypass*, to a given binary tree. $Prune(u)$ is an operation which removes a leaf node $u$ from the current tree, and $bypass(v)$ is an operation (following a prune operation) that removes a node $v$ with exactly one child $w$ and then lets the parent of $v$ become the new parent of $w$. We define a *contraction phase* to be the consecutively execution of prune and bypass operations.

Let $T$ be an $n$-leave binary tree with the root $r$. Given a Euler tour starting from $r$ of $T$, the algorithm initially numbers the leaves from 1 to $n$ according to the order of their appearances in the tour. Then, the algorithm repeats the following steps. In each step, *prune* and *bypass* work only on the leaves with odd index and their parents. Hence, these two operations can be performed independently and delete $\lfloor \frac{l}{2} \rfloor$ leaves together with their parents on the binary tree in each step, where $l$ is the number of the current leaves. Therefore, the tree will be reduced to a three-node tree after repeating the steps in $\lceil \log n \rceil$ times.

**Lemma 2.** *[1] If the prune operation and bypass operation can be performed by one processor in constant time, the binary tree contraction algorithm can be implemented in* $O(\log n)$ *time using* $O(n/\log n)$ *processors on an EREW PRAM, where* $n$ *is the number of nodes in an input binary tree.*

Consider a node $x$ in a rooted tree $T$. Any node $y$ on the unique path from $x$ to the root is called an *ancestor* of $x$. If $y$ is an ancestor of $x$, then $x$ is a *descendant* of $y$. Further, $x$ is a *proper descendant* of $y$ when $x \neq y$. Note that every node is both an ancestor and a descendant of itself. For convenience, we allow $\mathcal{U}_G$ to represent one of $\mathcal{U}_{V(G)}$ and $\mathcal{U}_{E(G)}$ if it is not particularly specified.

[2] In this paper, a subgraph $H$ of $G$ is represented by a set $Q$: If $Q \in \mathcal{U}_{V(G)}$, then $H = (Q, \emptyset)$; If $Q \in \mathcal{U}_{E(G)}$, then $H = (\{x | x \text{ is an endpoint of an edge in } Q\}, Q)$.

**Definition 5.** Let $u$ and $v$ be two nodes of a $(k, \Theta)$-parse tree $T$ such that $u$ is a descendant of $v$. A $k$-ary function $h : \mathcal{U}_{G_u}{}^k \mapsto \mathcal{U}_{G_v}$ possesses the *canonical form*, if $h(X_1, \dots, X_k) = \Theta\{X_{b_1} \cup C_1, X_{b_2} \cup C_2, \dots, X_{b_a} \cup C_a\}$, where $b_i \neq b_j$ for two distinct $1 \leq i, j \leq a$, and $C_i \in (\mathcal{U}_{G_v} \setminus \mathcal{U}_{G_u})$.

The following lemma can be shown by the set theory and properties of the function composition.

**Lemma 3.** *Let* $\Theta \in \{\text{MIN}_v, \text{MIN}_e\text{MAX}_v, \text{MAX}_e\}$, *and let* $h_0 : \mathcal{U}_{G_u}{}^k \mapsto \mathcal{U}_{G_v}$ *be a function with the canonical form, where* $u$ *is a descendant of* $v$. *If* $k$ *functions* $h_i : \mathcal{U}_{G_w}{}^k \mapsto \mathcal{U}_{G_u}$ *possess the canonical form, where* $1 \leq i \leq k$ *and* $w$ *is a descendant of* $u$, *then the function obtained from the composition* $h_0 \circ (h_1, h_2, \dots, h_k) : \mathcal{U}_{G_w}{}^k \mapsto \mathcal{U}_{G_v}$ *possesses the canonical form.*

We next develop a parallel algorithm for the $(k, \Theta)$-parse tree problem. For a node $x$ in the current tree $H$, let $par_H(x)$ (respectively, $child_H(x)$) denote the *parent* (*children*) *of* $x$, and let $sib_H(x)$ denote the *sibling of* $x$. The subscript $H$ can be omitted if no ambiguity arises. Recall that $H(x)$ be the subtree of $H$ rooted at $x$, and $R_x = \langle R_{x,1}, \dots, R_{x,k} \rangle$ is the list of the target subgraphs associated with $x$.

During the process of executing the tree contraction, we aim at constructing $k$ $k$-ary functions $h_{x,1}, h_{x,2}, \dots, h_{x,k}$ associated with each node $x$ of the current tree such that $h_{x,i}$'s possess the canonical form and satisfy the condition described below. Let $v$ be an internal node in the current tree whose left child and right child are $u$ and $w$, respectively. Also let $u'$ be the left child and $w'$ be the right child of $v$ in the original tree. For the remainder of this section, we call $u'$ and $w'$ *replacing ancestors of* $u$ *and* $w$ *with respect to* $v$, respectively. Once $R_{u,i}$ and $R_{w,i}$, $1 \leq i \leq k$, are provided as the inputs of $h_{u,i}$ and $h_{w,i}$, respectively, the target subgraphs of $v$ can be obtained from $R_{u'} = \langle R_{u',1}, \dots, R_{u',k} \rangle = \langle h_{u,1}(R_{u,1}, \dots, R_{u,k}), \dots, h_{u,k}(R_{u,1}, \dots, R_{u,k}) \rangle$, and $R_{w'} = \langle R_{w',1}, \dots, R_{w',k} \rangle = \langle h_{w,1}(R_{w,1}, \dots, R_{w,k}), \dots, h_{w,k}(R_{w,1}, \dots, R_{w,k}) \rangle$, using the formula

$$R_{v,i} = \Theta\{R_{u',f_i(u',1)} \cup R_{w',g_i(w',1)}, R_{u',f_i(u',2)} \cup R_{w',g_i(w',2)}, \dots, R_{u',f_i(u',a_{v,i})} \cup R_{w',g_i(w',a_{v,i})}\}. \quad (1)$$

where, $R_{u',f_i(u',j)} = h_{u,f_i(u',j)}(R_{u,1}, \dots, R_{u,k})$ and $R_{w',g_i(w',j)} = h_{w,g_i(w',j)}(R_{w,1}, \dots, R_{w,k})$ for $1 \leq j \leq a_{v,i}$.

We call the functions $h_{x,i}$, $1 \leq i \leq k$, computed for each node $x$ in the current tree *the crucial functions of* $x$.

We next describe the details of our algorithm. Initially, for each node $v$ in the given tree we construct $k$ functions $h_{v,i}(X_1, \dots, X_k) = \Theta\{X_i \cup \emptyset\}$, $1 \leq i \leq k$. Clearly, these functions are crucial functions.

In the execution of the tree contraction, assume that $prune(u)$ and $bypass(par(u))$ are performed consecutively. Let $par(u) = v$ and $sib(u) = w$ in the current tree. Let $u'$ and $w'$ be the replacing ancestors of $u$ and $w$ with respect to $v$, respectively. Assume that $h_{u,i}$ and $h_{w,i}$, $1 \leq i \leq k$, are crucial functions of $u$

and $w$ in the current tree. Thus $R_{u'} = \langle h_{u,1}(R_{u,1}, \ldots, R_{u,k}), \ldots, h_{u,k}(R_{u,1}, \ldots, R_{u,k})\rangle$ and $R_{w'} = \langle h_{w,1}(R_{w,1}, \ldots, R_{w,k}), \ldots, h_{w,k}(R_{w,1}, \ldots, R_{w,k})\rangle$. Since $u$ is a leaf, $R_{u,i}$'s are associated with $u$ before executing the tree contraction algorithm. Therefore, the above $k$ target subgraphs $R_{u'}$ can be obtained through function evaluation. On the other hand, since $w$ is not a leaf in the current tree, $R_{w,i}$, $1 \le i \le k$, is an indeterminate value represented by variable $X_i$. Hence. $R_{w'}$ can be represented by $\langle h_{w,1}(X_1, \ldots, X_k), \ldots, h_{w,k}(X_1, \ldots, X_k)\rangle$. By Equation 1, we construct $k$ intermediate functions representing $k$ target subgraphs $R_v$ from $R_{u'}$ and $R_{w'}$ by:

$$R_{v,i} = \Theta\{R_{u',f_i(u',1)} \cup R_{w',g_i(w',1)}, R_{u',f_i(u',2)} \cup R_{w',g_i(w',2)}, \ldots, R_{u',f_i(u',a_{v,i})} \cup R_{w',g_i(w',a_{v,i})}\}, \quad (2)$$

where $R_{w',g_i(w',j)} = h_{w,g_i(w',j)}(X_1, \ldots, X_k)$, $1 \le j \le a_{v,i}$

As with the proof similar to that of Lemma 3, Equation 2 can be further simplified as

$$R_{v,i} = \Theta\{X_{b_1} \cup C_1, X_{b_2} \cup C_2, \ldots, X_{b_a} \cup C_a\}, \quad (3)$$

where $b_i \ne b_j$ for two distinct $1 \le i, j \le a$, $X_{b_i}$ are variables drawn from $\mathcal{U}_w$, and $C_i \in (\mathcal{U}_{G_v} \setminus \mathcal{U}_{G_w})$.

Therefore, the above functions (constructed after executing $prune(u)$) possess the canonical form. Given those functions $R_{v,i}$'s, the contribution to the $k$ target subgraphs of $par(v)$ is obtained by function composition $h_{v,i}(R_{v,1}, \ldots, R_{v,k})$ for all $1 \le i \le k$. These functions are constructed for $w$ after executing $bypass(par(v))$. By Lemma 3, $h_{v,i}(R_{v,1}, \ldots, R_{v,k})$, $1 \le i \le k$, possesses the canonical form. Hence, we have the following lemma.

**Lemma 4.** *During the process of executing the binary tree contraction on a $(k, \Theta)$-parse tree to remove some nodes, the crucial functions of the remaining nodes of the current tree can be constructed in $O(k^3)$ time using one processor.*

**Theorem 1.** *The $(k, \Theta)$-parse tree problem can be solved in $O(k^3 \log n)$ time using $O(n/\log n)$ processors on an EREW PRAM, where $n$ is the number of nodes of the input tree.*

**Definition 6.** Let $G$ be a decomposable graph and let $T_G$ be a parse tree. A problem $\mathcal{P}$ is said to be a $(k, \Theta)$-*regular problem* on $G$ if $\mathcal{P}$ can be reduced to a $(k, \Theta)$-parse tree problem $\mathcal{B}$ on $T_G$ such that the solution of $\mathcal{B}$ is exactly the solution of $\mathcal{P}$. Moreover, the reduction scheme takes $O(k^3 \log |V(T_G)|)$ time using $O(|V(T_G)|/\log |V(T_G)|)$ processors on an EREW PRAM.

Note that each $(k, \Theta)$-regular problem corresponds to a $(k, \Theta)$-parse tree. This tree is obtained from a parse tree $T_G$ in which some additional data structures are associated with $V(T_G)$ (refer to Definition 3). In Section 4, we assume that a parse tree is given for solving a $(k, \Theta)$-regular problem on a decomposable graph.

The following result directly follows from Definition 6 and Theorem 1.

**Theorem 2.** *Given a parse tree of a decomposable graph $G$, a $(k,\Theta)$-regular problem on $G$ can be solved in $O(k^3 \log|V(T_G)|)$ time using $O(|V(T_G)|/\log|V(T_G)|)$ processors on an EREW PRAM.*

**Corollary 1.** *A $(k,\Theta)$-regular problem of a decomposable graph $G=(V,E)$ can be solved in $O(T_d(|V|,|E|)+\log|V(T_G)|)$ time using $O(P_d(|V|,|E|)+|V(T_G)|/\log|V(T_G)|)$ processors on $M_d$.*

## 4 $(k,\Theta)$-Regular Problems

Given a problem $\mathcal{P}$, a graph $G_1$, a subgraph $G_2$ of $G_1$, and a subset $Q$ of vertices in $G_2$, $\mathcal{P}_Q(G_1,G_2)$ is a solution to the input graph $G_1$ such that this solution contains all vertices in $Q$ and is in $G_2$. For the case of $Q=\emptyset$, i.e., $\mathcal{P}_\emptyset(G_1,G_2)$, the notation represents a solution to $G_1$ and this solution is contained in $G_2$. For brevity, let $\mathcal{P}_Q(G,G)=\mathcal{P}_Q(G)$.

An *independent set* of a graph is a subset of its vertices such that no two vertices in the subset are adjacent. The *maximum independent set problem* $\mathcal{I}$ is the problem of finding a maximum-cardinality independent set in the input graph. Using our notation, given an input graph $G$, a solution is $\mathcal{I}_\emptyset(G)$. For a basis rooted tree $G=(\{r\},\{\},(r))$, $\mathcal{I}_\emptyset(G)$ and $\mathcal{I}_{\{r\}}(G)$ are both equal to $\{r\}$, and $\mathcal{I}_\emptyset(G[V\setminus\{r\}])=\emptyset$.

**Lemma 5.**
*Assume $G=(V_1\cup V_2, E_1\cup E_2\cup\{(r_1,r_2)\},(r_1))$ is obtained from $G_1=(V_1,E_1,(r_1))$ and $G_2=(V_2,E_2,(r_2))$.*

(1) $\mathcal{I}_\emptyset(G)=\textsc{Max}_v\{\mathcal{I}_{\{r_1\}}(G_1)\cup\mathcal{I}_\emptyset(G_2[V_2\setminus\{r_2\}]),$
$\mathcal{I}_\emptyset(G_1[V_1\setminus\{r_1\}])\cup\mathcal{I}_{\{r_2\}}(G_2),\mathcal{I}_\emptyset(G_1[V_1\setminus\{r_1\}])\cup\mathcal{I}_\emptyset(G_2[V_2\setminus\{r_2\}])\}$;
(2) $\mathcal{I}_{\{r\}}(G)=\mathcal{I}_{\{r_1\}}(G_1)\cup\mathcal{I}_\emptyset(G_2[V_2\setminus\{r_2\}])$;
(3) $\mathcal{I}_\emptyset(G[V\setminus\{r\}])=\mathcal{I}_\emptyset(G_1[V_1\setminus\{r_1\}])\cup\mathcal{I}_\emptyset(G_2)$.

*Proof.* Straightforward. □

By the above result, it is not difficult to obtain the following two theorems.

**Theorem 3.** *The maximum independent set problem is a $(3,\textsc{Max}_v)$-regular problem on trees.*

**Theorem 4.** *The maximum independent set problem on trees can be solved in $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM, where $n$ is the number of vertices of the input graph.*

Given an undirected graph $G=(V,E)$, a *matching* is a subset of edges $M\subseteq E$ such that for all vertices $v\in V$, at most one edge of $M$ is incident on $v$. The *maximum matching problem* $\mathcal{M}$ is the problem of finding a matching of maximum cardinality. For a basis series-parallel graph $G=(\{l,r\},\{(l,r)\},(l,r))$, $\mathcal{M}_\emptyset(G)=\{(l,r)\},\mathcal{M}_{\{l\}}(G[V\setminus\{r\}])=\emptyset,\mathcal{M}_{\{r\}}(G[V\setminus\{l\}])=\emptyset,\mathcal{M}_{\{l,r\}}(G)=\{(l,r)\},\mathcal{M}_\emptyset(G[V\setminus\{l,r\}])=\emptyset$. We can further show that the maximum matching problem is a $(5,\textsc{Max}_e)$-regular problem on series-parallel graphs.

By the methods described in [4,10] to construct parse trees of series-parallel graphs, we have the following theorem.

**Theorem 5.** *The maximum matching problem on series-parallel graphs can be solved in sequential* $O(n+m)$ *time, and in parallel in* $O(\log m \log^* m)$ *time using* $O(m/\log m \log^* m)$ *processors on an EREW PRAM.*

Given a simple graph $G = (V, E)$, a vertex $v \in V$ is said to *dominate* itself and all vertices adjacent to $v$. A subset $D$ of $V$ is called an *efficient dominating set* of $G$ if every vertex in $V$ is dominated by exactly one vertex in $D$. Note that not all graphs have efficient dominating sets. Moreover, if a graph possesses an efficient dominating set, then all these sets have the same cardinality. The *efficient domination problem* $\mathcal{D}$ is the problem to find an efficient dominating set of a given graph if such a set exists. Using our paradigm, we can also show the following result.

**Theorem 6.** *The efficient domination problem on series-parallel graphs can be solved in linear* $O(n + m)$ *time, and in parallel in* $O(\log m \log^* m)$ *time using* $O(m/\log m \log^* m)$ *processors on an EREW PRAM.*

## References


1. K. Abrahamson, N. Dadoun, D. G. Kirkpatrick, and T. Przytycka, A simple parallel tree contraction algorithm, *Journal of Algorithms*, 10, pp. 287-302, 1989.
2. S. Arnborg and A. Proskurowski, Linear time algorithms for NP-hard problems restricted to partial $k$-trees, *Discrete Applied Mathematics*, 23, pp. 11-24, 1989.
3. M. W. Bern, E. L. Lawler, and A. L. Wong, Linear-time computation of optimal subgraphs of decomposable graphs, *Journal of Algorithms*, 8:216-235, 1987.
4. H. L. Bodlaender and B. van Antwerpen-de Fluiter, Parallel algorithms for series parallel graphs and graphs with treewidth two, *Algorithmica*, 29(4):534-559, 2001.
5. M. C. Golumbic, *Algorithmic graph theory and perfect graphs*, Academic press, New York, 1980.
6. R. M. Karp and V. Ramachandran, Parallel algorithms for shared memory machines, *Handbook of Theoretical Computer Science*, North-Holland, Amsterdan, pp. 869-941, 1990.
7. S. Mahajan and J. G. Peters, Algorithms for regular properties in recursive graphs, in: *Proceedings of the 25th Allerton Conference on Communication, Control, and Computing*, pp. 14-23, 1987.
8. S. Mahajan and J. G. Peters, Regularity and locality in $k$-terminal graphs, *Discrete Applied Mathematics*, 54:229-250, 1994.
9. K. Takamizawa, T. Nishizeki, and N. Saito, Linear-time computability of combinatorial problems on series-parallel graphs, *Journal of the ACM*, 29:623-641, 1982.
10. J. Valdes, R. E. Tarjan, and E. L. Lawler, The recognition of series-parallel digraphs, *SIAM Journal on Computing*, 11:298-313, 1982.
11. T. V. Wimer, Linear algorithms on $k$-terminal graphs, Ph.D. Thesis, Clemson University, Clemson, SC, 1987.
12. T. V. Wimer and S. T. Hedetniemi, $K$-terminal recursive families of graphs, *Congressus Numerantium*, 63:161-176, 1988.