ON THE POSSIBILITIES OF DFS TREE CONSTRUCTIONS:
SEQUENTIAL AND PARALLEL ALGORITHMS

by

E. Korach and Z. Ostfeld

Technical Report #508

May 1988
On the POSSIBILITIES of DFS TREE CONSTRUCTIONS:

SEQUENTIAL and PARALLEL ALGORITHMS

(Extended abstract)

Ephraim Korach and Zvi Ostfeld

Computer Science Department
Technion - Israel Institute of Technology
Haifa 32000, Israel

ABSTRACT

The Depth First Search (DFS) algorithm is one of the basic techniques which is used in a very large variety of graph algorithms. Every application of the DFS involves, beside traversing the graph, constructing a special structured tree, called a "DFS tree" that may be used subsequently. We raise two important questions regarding the structure of the DFS tree that is obtained, and discuss their solutions.

In some problems the degree of some vertices in a DFS tree obtained in a certain run are crucial and therefore we consider the following problem: Let $G=(V,E)$ be a connected undirected graph where $|V|=n$ and let $d \in N^*$ be a degree sequence upper bound vector. Is there any DFS tree $T$ with degree sequence $d_T$ that violates $d$, (i.e. $d_T \not\subseteq d$ which means: $\exists j$ such that $d_T(j) > d(j)$)? We show that this problem is NP-Complete even for the case where we restrict the degree of only one specific vertex (i.e. $d = (n-1,\ldots,n-1,k,n-1,\ldots,n-1)$). However, for the special interesting case where $d = (2,2,\ldots,2)$ (i.e. the underline graph of every DFS tree is an Hamiltonian path) we give a complete characterization of all graphs that support this restriction. This characterization enables us to solve this problem by efficient and simple algorithms both in the sequential and in the parallel cases.

In a previous work ([KO]) we have shown that the family of graphs in which every spanning tree is a DFS tree is quite limited. Therefore the question: Given an undirected graph $G=(V,E)$ and a spanning tree $T$, is $T$ a DFS tree (T-DFS) in $G$? is naturally raised. This question is important in many applications. For example when we would like to run a DFS in an undirected graph where the weights of the edges are all distinct and obtain the unique minimum spanning tree $T$ as a DFS tree. We present a parallel algorithm which solves this problem in $O( \log |V|)$ time complexity and uses $O(|E|)$ processors on a CREW PRAM. We study this problem for directed graphs. We present a linear time algorithm for solving it in the sequential case and we show an efficient parallel implementation of it.
1. INTRODUCTION

The Depth First Search (DFS) algorithm is one of the basic techniques which is used in a very large variety of graph algorithms. The history of this algorithm (in a different form) goes back to 1882 when Tremaux’ algorithm for the maze problem was first published (see [BLW, page 18]). The impact of DFS grew rapidly since the Hopcroft and Tarjan version of it was published (see [Ta], [HT a], [HT b] and [HT c]). In many areas of computer science, this algorithm is used, and recently it also has penetrated the field of parallel and distributed algorithms (e.g. [AA], [Aw], [LMT], [Re] and [Ti]). Every use of the DFS, beside traversing the graph, constructs a special structured directed rooted tree, called a DFS tree, that may be used subsequently.

In this paper, we raise two important questions, regarding the structure of the DFS tree that is obtained, and discuss their solutions. In some problems the degree of some vertices in a DFS tree obtained in a certain run might be too high. For example, in distributed and parallel algorithms the degree in the tree might be limited because of technical limitations (e.g. if the DFS tree is going to be used as a massive communication tree then each vertex may have a limited number of ports to be used in the tree). This leads us to the following problem: Let \( G = (V, E) \) be a connected undirected graph where \( |V| = n \) and let \( d \in \mathbb{N}^* \) be a degree sequence upper bound vector. Is there any DFS tree \( T \) with degree sequence \( d_T \) that violates \( d \), (i.e. \( d_T \not\leq d \) which means: \( \exists j \) such that \( d_T(j) > d(j) \)?)

We show that this problem is \( NP \)-Complete even for the case where we restrict the degree of only one specific vertex (i.e. \( d = (n-1, \ldots, n-1, k, n-1, \ldots, n-1) \)). However, for the special interesting case where \( d = (2, 2, \ldots, 2) \) (i.e. the underline graph of every DFS tree is an Hamiltonian path) we give a complete characterization of all graphs that support this restriction. This characterization enables us to solve this problem by efficient and simple algorithms both in the sequential and in the parallel cases.

Previous results ([KO]) have shown that the family of graphs in which every spanning tree, is a DFS tree is quite limited. Therefore the question: Given an undirected graph \( G = (V, E) \) and a spanning tree \( T \), is \( T \) a DFS tree (T-DFS) in \( G \)? was naturally raised and answered by linear time algorithms in [KO] and independently in [HN]. This question is important in many applications. For example when we would like to run a DFS in an undirected graph where the weights of the edges are all distinct and obtain the unique minimum spanning tree \( T \) as a DFS tree.

In this work we present a parallel algorithm which solves this problem in \( O(\log |V|) \) time complexity and uses \( O(|E|) \) processors on a concurrent read exclusive write parallel random access machine (CREW PRAM).

In the last part of this paper we study this problem for directed graphs. In section 5 we present a linear time algorithm for solving it in the sequential case and in section 6 we show an efficient parallel implementation of it based on parallel implementation of matrix multiplication. For example, by using Strassen method (see [St]) we get
2. SOME DEFINITIONS and CONVENTIONS

Let $T$ be an undirected spanning tree in an undirected graph $G = (V, E)$ and let $s \in V$. $T_s$ is the tree $T$ with an orientation that makes $s$ the root of $T_s$. $T$ is called a DFS tree ($T$-DFS) in $G$ if there exists a vertex $s \in V$ such that $T_s$ is a DFS tree ($T$-DFS) in $G$ (i.e. $T_s$ can be constructed by a DFS run in $G$).

Let $T$ be a directed spanning tree in a digraph $G$. $T$ is a DFS tree ($T$-DFS) in $G$ if it can be constructed by a DFS run in $G$.

Let $(a, b, c, d, e, f, g, h)$ be vertices in a directed tree $T$. If there is a directed path in $T$ from $a$ to $b$, we say that $a$ is an ancestor of $b$ and $b$ is a descendant of $a$. A vertex is an ancestor and a descendant of itself.

d is called second-ancestor if there is a tree edge $c \rightarrow d$ and $d$ is an ancestor of $e$ in $T$.

$f$ is called the lowest common ancestor of $g$ and $h$ if (i) $f$ is a common ancestor of $g$ and $h$ in $T$. (ii) Any common ancestor of $g$ and $h$ in $T$ is an ancestor of $f$.

To simplify the discussion we assume that all graphs in this paper are without loops and parallel edges. However, the results are easily extended to graphs with loops and parallel edges.

3. DEGREE RESTRICTED DFS TREES

3.1 The general problem and its complexity

Degree Restricted DFS Trees (DRDT) problem: Let $G=(V,E)$ be a connected undirected graph where $|V|=n$ and let $d \in \mathbb{N}^n$ be a degree sequence upper bound vector. Is there any DFS tree $T$ with degree sequence $d_T$ that violates $d$?, (i.e. $d_T \not\subseteq d$ which means: $\exists j$ such that $d_T(j) > d(j)$)?

This question is important since in some problems the degree of some vertices in a DFS tree obtained in a certain run might be too high. For example, in distributed and parallel algorithms the degree in the tree might be limited because of technical limitations (e.g. if the DFS tree is going to be used as a massive communication tree then each vertex may have a limited number of ports to be used in the tree).

Theorem 3.1.1: The DRDT problem is NP-Complete.

Proof: It is easy to see that DRDT$\in$NP, because a nondeterministic algorithm needs only guess a spanning tree $T$ in $G$ and check in polynomial time that $T$ is a DFS tree in $G$ and that $d_T$, the degree sequence of $T$ violates $d$. The rest of the proof follows from Lemma 3.1.2 where we show that this problem is NP-Complete even for the case where we restrict the degree of only one specific vertex (i.e. $d=(n-1,\ldots,n-1,k,n-1,\ldots,n-1)$).

Lower Bound Degree (LBD) problem: Given a graph $G=(V,E)$ and a vertex $v \in V$. Is there a DFS tree where
the degree of $v \geq k \geq 3$.

Lemma 3.1.2: The LBD problem is NP–Complete.

Proof: It is obvious that $LBD \in NP$. To show completeness we transform Hamiltonian path to $LBD$. □

3.2 Hamiltonian DFS graphs

By Theorem 3.1.1 the DRDT problem is an NP–Complete. However, for the special interesting case where $d = (2,2,...,2)$ (i.e. the underline graph of every DFS tree is an Hamiltonian path) we give a complete characterization of all graphs that support this restriction. This characterization enables us to solve this problem by efficient and simple algorithms both in the sequential and in the parallel cases.

Definition 3.2.1: An undirected connected simple graph is called Hamiltonian–DFS–Graph (H–DFS–G) if every DFS tree in it is an Hamiltonian directed path.

Theorem 3.2.2: A graph $G$ is H–DFS–G if and only if it is a simple circuit, or a $K_n$, or a $K_{n,n}$ where $K_n$ is the complete graph on $n$ vertices and $K_{n,n}$ is the complete bipartite graph on $2n$ vertices with two equal parts.

Proof: The proof of the if part is obvious. The only if part follows from Corollary 3.2.7 and Lemma 3.2.8 below. □

Proposition 3.2.3: In every DFS tree $T_*$ of an H–DFS–G with at least three vertices there is a back edge from the leaf to the root $s$.

Proof: Let $G=(V,E)$ be an H–DFS–G and let the Hamiltonian path $T_* = s \rightarrow k \rightarrow v_3 \rightarrow v_4 \rightarrow \cdots \rightarrow v_n$, $(V = \{s, k, v_3, v_4, \cdots, v_n\})$ be a DFS tree in $G$. Assume that $(s, v_n) \notin E$. Consider a DFS run starting from $k$ in which the first part of the search is done along the path $k \rightarrow v_3 \rightarrow v_4 \rightarrow \cdots \rightarrow v_n$. Since $(s, v_n) \notin E$ and since $s$ is the only vertex that has not yet been discovered, it is clear that we have to back in the search from $v_n$. Therefore the resulting DFS tree will not be a directed Hamiltonian path, a contradiction. □

Corollary 3.2.4: Every DFS run in an H–DFS–G with at least three vertices creates an Hamiltonian circuit by adding to the tree the edge between the leaf and the root.

Lemma 3.2.5: If $G=(V,E)$ is an H–DFS–G and has a subgraph which is $K_3$ then $G$ is a complete graph.

Proof: Let $G'=(V', E')$ be a maximal complete subgraph of $G$ with size $k \geq 3$. If $G = G'$ (i.e. $k = n$) we are done. Otherwise let $(a, b, c) \subseteq V'$ and $d \in V-V'$ such that $(a, d) \in E$ and $(b, d) \notin E$ (since $3 \leq k < n$ there must be such $a, b, c$ and $d$). Since $\{(a, b), (b, c), (c, a)\} \subseteq E$ it follows, using Corollary 3.2.4 that there is an Hamiltonian circuit in $G$ of the form $d \rightarrow a \rightarrow b \rightarrow c \rightarrow \cdots \rightarrow d \rightarrow a \rightarrow b \rightarrow c + \Pi_{c,d}$. Now, consider a DFS run starting from $a$ along the path $a \rightarrow c + \Pi_{c,d}$ (recall that $(a, c) \in E$). Since $(b, d) \notin E$ and since $b$ is the only vertex that has not yet been discovered, it is clear that we have to back in the search from $d$. Therefore the resulting DFS tree will not be a directed Hamiltonian path, a contradiction. □
Lemma 3.2.6: Let $G=(V,E)$ be an $H$-$DFS$-$G$. If $G$ has a simple circuit of length $k$ where $4<k<n$ then $G$ has a simple circuit of length $k-2$.

Proof: Let $v_1-v_2-v_3-v_4-...-v_{k-1}-v_k-v_1 = \pi_{v_1,v_2} + \pi_{v_2,v_3} + ... + (v_{k-1}-v_k-v_1)$, where $\{v_1, \ldots, v_k\} = V' \subset V$ be a circuit of length $k$ in $G$. Since $G$ is an $H$-$DFS$-$G$ it is possible to extend the path $\pi_{v_1,v_2} = \pi_{v_1,v_2} + \pi_{v_2,v_3} + ... + (v_{k-1}-v_k)$ to an Hamiltonian path in $G$ by a $DFS$ run in $G$ starting at $v_1$, then moving along $\pi_{v_1,v_2}$ and then moving along the path $v_k-u_1-u_2-...-u_{n-k}-v_k = (v_k-u_1) + \pi_{u_1,u_2},$ where $\{u_1, \ldots, u_{n-k}\} = V'-V'$.

By Proposition 3.2.3, $(v_1,u_{n-k})\in E$ and therefore the following path exists in $G$: $v_3-v_4+\pi_{v_4,v_1}+(v_{k-1}-v_k-v_{n-k})+\pi_{u_{n-k},u_1}$ (where the path $\pi_{u_{n-k},u_1}$ is the path $\pi_{u_1,u_{n-k}}$ in reverse order). If $(v_2,u_1)\notin E$ it is impossible to extend this path to an Hamiltonian path in contradiction to the definition of $G$. Hence $(v_2,u_1)\in E$.

By analogue arguments $(v_{k-1},u_{n-k})\in E$. It follows that $\pi_{v_{k-1},v_3} = \pi_{v_{k-1},v_k-1} + (v_{k-1}-u_{n-k}) + \pi_{u_{n-k},u_1} + (u_1-v_2) + (v_2-v_3)$ is a path in $G$ and therefore there is a $DFS$ run in $G$, the first part of which is this path. $\pi_{v_{k-1},v_3}$ includes all the nodes in $G$ excluding $v_1$ and $v_k$. Since $G$ is an $H$-$DFS$-$G$ one of the following must hold:

(i) $(v_3,v_1)\in E$ which implies the existence of the following triangle $(a K_3)$ in $G$: $(v_1,v_2,v_3)$ and by Lemma 3.2.5 - $G$ is complete. (ii) $(v_3,v_3)\in E$ which implies the existence of the following circuit in $G$: $(v_3-v_4)+\pi_{v_4,v_3}+(v_{k-1}-v_k-v_3)$.

Since $k>4$ it follows that in both of these cases $G$ has a circuit of length $k-2$. □

Corollary 3.2.7: Let $G=(V,E)$ be an $H$-$DFS$-$G$ which is neither a simple circuit nor a bipartite graph, then it is a complete graph.

Proof: First we claim that $G$ contains a triangle. If $|V| \leq 5$ the claim is easily checked. If $|V| \geq 6$ then the claim follows by Lemma 3.2.6. The claim and Lemma 3.2.5 imply that $G$ is complete. □

Lemma 3.2.8: Let $G=(V,E)$, with $|V|\geq 3$ be an $H$-$DFS$-$G$ which is bipartite but not a simple circuit. Then $G$ is the complete bipartite $K_{n,n}$.

Proof: The proof is based on Corollary 3.2.4 and on Lemma 3.2.6 and is similar to the proof of Lemma 3.2.5. The complete proof will be given in the full paper. □

Corollary 3.2.10 below is an extension of Theorem 3.2.2. In order to prove it we need the following proposition:

Proposition 3.2.9: If $G=(V,E)$ has a separating vertex $v \in V$ with degree $\geq 3$ then there is a $DFS$ tree $T$ in $G$ where the degree of $v \geq 3$.

Proof: Let $(x,y,z) \subseteq V$ such that $(x,v),(y,v),(z,v) \subseteq E$ where $x$ and $y$ are in different 2-connected components. The $DFS$ run beginning with the path $x-v-x$ will create a tree where the degree of $v$ is at least 3. □

Corollary 3.2.10: Let $G=(V,E)$ be an undirected graph. The underline graph of every $DFS$ tree in $G$ is an Hamil-
tonian path if and only if \( G \) is one of the following four possibilities: a simple circuit, a \( K_n \), a \( K_{n,n} \) or a simple path.

Proof: The if part is trivial. To show the only if part let us first assume that \( G \) has no separating vertex. In this case the root of every DFS tree in \( G \) has degree one. Therefore every DFS tree in \( G \) is an Hamiltonian directed path and by Theorem 3.2.2 the proof of this case is completed. The correctness of the corollary for graphs which have separating vertices is an immediate consequence of Proposition 3.2.9. □

Corollary 3.2.11: The characterization given in Corollary 3.2.10 enables us to solve the DRDT problem for a graph \( G=(V,E) \) where \( d=(2,2,\ldots,2) \) by a simple sequential algorithm with \( O(|E|) \) time complexity and by a simple parallel algorithm with \( O(log\ |V|) \) time complexity and \( O(|E|) \) processors on an exclusive read exclusive write parallel random access machine (EREW PRAM). □

4. PARALLEL RECOGNITION of DFS TREES

Previous results ([KO]) have shown that the family of graphs in which every spanning tree, is a DFS tree is quite limited. Therefore the question: Given an undirected graph \( G=(V,E) \) and a spanning tree \( T \), is \( T \) a DFS tree (T-DFS) in \( G \)? was naturally raised and answered by linear time algorithms in [KO] and independently in [HN].

In this work we present a parallel algorithm which solves this problem in \( O(log\ |V|) \) time complexity and uses \( O(|E|) \) processors on a concurrent read exclusive write parallel random access machine (CREW PRAM).

Definition 4.1: Let \( T \) be an undirected spanning tree in an undirected graph \( G=(V,E) \) and let \( s \in V \). \( T_s \) induces a partition of \( E \) into three types of edges: (i) Tree edges. (ii) Back edges: An edge \((a,b)\in E-T \) is a back edge with respect to \( T_s \) if \( a \) is either an ancestor of \( b \) or a descendant of \( b \) in \( T_s \). (iii) Cross edges: The rest of the edges in \( E \).

Observation 4.2: Let \( T \) be an undirected spanning tree in an undirected graph \( G=(V,E) \) where \( \{s,a,b\} \subset V \) and let \( \pi_{s,a} \) and \( \pi_{s,b} \) be the paths in \( T \) from \( s \) to \( a \) and from \( s \) to \( b \) respectively. \((a,b)\) is a cross edge with respect to \( T_s \) if and only if \( b \notin \pi_{s,a} \) and \( a \notin \pi_{s,b} \). □

Proposition 4.3: Let \( T \) be a spanning tree in \( G=(V,E) \). Let \( \{r,s,u,v\} \subset V \) and let \( T_r \) and \( T_s \) be two orientations of \( T \) rooted at \( r \) and \( s \) respectively. A non tree edge \( e=(u,v)\in E \) is a cross edge in \( T \) if and only if one of the following conditions holds in \( T_e \):

(1) \( e \) is a cross edge and \( r \) is neither a descendant of \( u \) nor a descendant of \( v \).

(2) \( e \) is a back edge and there is a vertex \( w \in V \) such that: (i) \( w \) is second-\( \pi_{u,v} \). (ii) \( r \) is a descendant of \( w \) and is not a descendant of \( v \).

Proof: Follows from Observation 4.2. □

Proposition 4.4 [Ta, Theorem 1]: Let \( T \) be an undirected spanning tree in an undirected graph \( G=(V,E) \) and let \( s \in V \). \( T_s \) is a T-DFS in \( G \) if and only if every edge \((a,b)\in E-T \) is a back edge in \( T_s \). □
Corollary 4.5: Let \( T \) be a spanning tree in an undirected graph \( G=(V, E) \), then \( T \) is a \( T\text{-DFS} \) in \( G \) if and only if there is some \( r \in V \) such that \( G \) contains no cross edges in \( T_r \).

Proof: Follows directly from Proposition 4.4 and the definition of a \( T\text{-DFS} \).

In the following, we present an algorithm for checking whether a given undirected spanning tree is a \( T\text{-DFS} \) in an undirected graph \( G \). The algorithm has an efficient parallel implementation and uses some ideas from [HN].

**PAR_CHECK** \( (G, T) \) (Check in parallel whether \( T \) is a \( T\text{-DFS} \) in \( G \))

**input:** An undirected graph \( G \) and an undirected spanning tree \( T \) in \( G \).

**output:** A decision whether \( T \) is a \( T\text{-DFS} \) in \( G \).

**variables:** \( f \), \( count \), \( sum \) are three arrays indexed by the vertices of \( V \).

1. Choose a vertex \( s \in V \) and compute \( T_s \).
2. For every vertex \( x \in V \), set \( f(x) \) to be the father of \( x \) in \( T \). If \( f(s) = \text{null} \) and set \( count(x) \) to be 0.
3. For every edge \( e \in E - T \) begin
   (3.1) If \( e=(u,v) \) is a cross edge in \( T \), then:
   \[
   \begin{align*}
   &count(u):=\text{count}(u)-1, \\
   &count(v):=\text{count}(v)-1, \\
   &count(s):=\text{count}(s)+1.
   \end{align*}
   \]
   (3.2) Else if \( e=(x,y) \) is a back edge in \( T \), then:
   \[
   \begin{align*}
   &\text{if } x \text{ is an ancestor of } y \text{ then } \{ u:=x, v:=y \}, \\
   &\text{else } y \text{ is an ancestor of } x \text{ then } \{ u:=y, v:=x \}, \\
   &\text{find } w \in V \text{ such that } w \text{ is second-cest-} \pi_{u,v}. \\
   &\text{count}(v):=\text{count}(v)-1, \\
   &\text{count}(w):=\text{count}(w)+1.
   \end{align*}
   \]
   end.
4. For every vertex \( x \in V \) compute \( sum(x) \) which is the sum of values of \( count(u) \) for all vertices \( u \in V \) where \( u \) is an ancestor of \( x \) in \( T_s \).

Decision (of algorithm **PAR_CHECK**): \( T \) is a \( T\text{-DFS} \) in \( G \) if and only if there is at least one vertex \( x \in V \) such that \( sum(x)=0 \).

**Theorem 4.6:** Algorithm **PAR_CHECK** \( (G=(V, E), T) \) is correct.

Proof: Let \( e=(u,v) \in E - T \) and \( \{s,u,v,x\} \subseteq V \) where \( s \) is the vertex chosen by the algorithm at step (1). Assume that \( e \) is a cross edge in \( T_s \). Since we add one to \( count(s) \) it is clear that if \( x \) is either a descendant of \( u \) or a descendant of \( v \) then \( sum(x) \) is not affected by the changes in step (3.1). Otherwise \( e \) contributes one to \( sum(x) \). Now assume that \( e \) is a back edge with respect to \( T_s \), where \( u \) is an ancestor of \( v \), and let \( w \in V \) be \( \text{second-cest-} \pi_{u,v} \). Then the operations in step (3.2) affect \( sum(x) \) only if \( x \) is a descendant of \( w \) and not a descendant of \( v \). In the latter case \( e \) contributes one to \( sum(x) \). Hence, by Proposition 4.3, for every vertex \( x \in V \) \( sum(x) \) is the number of cross edges in \( T_x \). Therefore the decision of the algorithm is correct by Corollary 4.5.

**Theorem 4.7:** Algorithm **PAR_CHECK** \( (G=(V, E), T) \) can be implemented in \( O(\log n) \) time complexity using \( O(m) \) processors on a CREW PRAM (where \( n=|V| \) and \( m=|E| \)).

Proof: Step (2) is trivially done by \( O(n) \) processors in \( O(1) \) time complexity. Step (1) of the algorithm is computed by \( O(n) \) processors in \( O(\log n) \) time complexity using the Euler tour techniques as presented in [TV]. Those techniques with the same time and processors complexity are used in step (3) as shown by [Vi] to form a data-structure
which enables the retrieval of the lowest common ancestor of any pair of vertices in $O(\log n)$ time by a single processor. This way we recognize cross and back edges and with a slight modification we compute $w$ in step (3.2). The parallel additions and subtractions are easily done in $O(\log n)$ time complexity with $O(m)$ processors (we compute the value of count which is an array of length $n$ while the number of items to be add and subtract is $O(m)$). The computation of step (4) is done in $O(\log n)$ time with $O(n)$ processors by using a "doubling" technique [Wy 79]. For each vertex $v \in V$, we initialize $\text{sum}(v):=\text{count}(v)$ and then repeat the following step, in parallel on all the vertices of $V$, until all of the $f$ values are null: If $f(v)$ is not null then set $\text{sum}(v)$ to $\text{sum}(v)+\text{sum}(f(v))$ and replace $f(v)$ by $f(f(v))$. It is obvious that we repeat this step no more than $\lceil \log n \rceil$ times (since the depth of $T_e$ is at most $n-1$).

Finally, the decision can be done in $O(\log n)$ time with $O(n)$ processors.

5. RECOGNITION of DFS TREES in DIGRAPHS

Definition 5.1: A directed spanning tree $T$ in a digraph $G=(V,E)$ induces a partitions of $E$ into four types of edges:

(i) Tree edges ($\overline{T}$).
(ii) Forward edges ($\overline{F}$): An edge $x \rightarrow y \in E - \overline{T}$ is a forward edge if $x$ is an ancestor of $y$ in $T$.
(iii) Back edges ($\overline{B}$): An edge $x \rightarrow y \in E$ is a back edge if $y$ is an ancestor of $x$ in $T$.
(iv) Cross edges ($\overline{C}$): The rest of the edges in $E$.

Definition 5.2: Let $V$ be a set of nodes and let $f: V \rightarrow \{1,2,\ldots,|V|\}$ be a bijection. We say that $V$ has an order induced by $f$.

Definition 5.3: Let $G=(V,E)$ be a digraph where $V$ has an order induced by $f$. The order is compatible (in $G$) if for every edge $x \rightarrow y \in E$ $f(x)<f(y)$.

Definition 5.4: Let $T(V,E)$ be a directed tree. An order of $V$ induced by a bijection $f$ is called $\text{DFS-T-order}$ (or preorder numbering) if there is a DFS run on the tree such that for every vertex $v \in V$, $f(v)=i$ if and only if $v$ is the $i$-th node to be discovered during the DFS run.

Proposition 5.5: A directed spanning tree $T$ in a digraph $G=(V,E)$ is a DFS tree $(T-\text{DFS})$ if and only if $T$ has a $\text{DFS-T-order}$ induced by a bijection $f: V \rightarrow \{1,2,\ldots,|V|\}$ that is compatible in $\overline{G}=(V, \overline{T} \cup \overline{F} \cup \overline{B} \cup \overline{C})$ where $\overline{T}$ are the tree edges, $\overline{F}$ are the forward edges, $\overline{B}$ are the back edges in reverse direction and $\overline{C}$ are the cross edges in reverse direction.

Definition 5.6: Let $T$ be a directed tree and let $\{x,y\}$ be two vertices in $T$. $T_x$ is the directed sub tree induced by all the descendants of $x$ in $T$ ($x$ is the root of $T_x$). Two directed sub trees $T_x$ and $T_y$ are called brother sub trees if $x$ and $y$ are brothers in $T$ ($x$ and $y$ have a common father in $T$).

Definition 5.7: Let $T$ be a directed spanning tree in a digraph $G=(V,E)$ where $e=x \rightarrow y$ is a non tree edge in $G$ and $z \in V$ is the lowest common ancestor of $x$ and $y$ in $T$. We define the following elementary reduction operations
(5.7.1) If \( e \in \overline{P} \cup \overline{B} \) then \( \Phi_1(G, T, e) = (V, E-e) \in \Phi_T(G) \) (i.e. \( e \) is deleted).

(5.7.2) If \( e \in \overline{C} \) then \( \Phi_2(G, T, e) = (V, E \cup \{\hat{x} \rightarrow \hat{y}\} - e) \in \Phi_T(G) \), where \( \hat{x} \in V \) is second-\( \pi_{x,x} \) and \( \hat{y} \in V \) is second-\( \pi_{x,y} \) (i.e. \( e \) is replaced by another cross edge \( \hat{x} \rightarrow \hat{y} \) where \( x \) and \( y \) are in brother sub trees \( T_x \) and \( T_y \) respectively).

Definition 5.8: Let \( T \) be a directed spanning tree in a digraph \( G=(V,E) \). We define the following set \( \Phi_T^+(G) \):
(i) \( G \in \Phi_T^+(G) \). (ii) If \( G' \in \Phi_T^+(G) \) then \( \Phi_T(G') \subseteq \Phi_T^+(G) \).

Definition 5.9: \( G' \) is a minor digraph of \((G,T)\) if \( G' \in \Phi_T^+(G) \).

Lemma 5.10: \( T \) is a DFS tree in \( G=(V,E) \) if and only if it is a DFS tree in every minor digraph of \((G,T)\).

Proof: One can see that any single implementation of \( \Phi_T \) does not change the compatibility of a DFS-\( T \)-order induced by any bijection \( f : V \rightarrow \{1,2,\ldots,|V|\} \).

Definition 5.11: Let \( T \) be a spanning tree of a digraph \( G=(V,E) \). A minor digraph \( G'=(V,E') \) of \((G,T)\) is a minimal minor if \( \Phi_T^+(G') = \{G'\} \).

Observation 5.12: Let \( T \) be a directed spanning tree of a digraph \( G=(V,E) \) and let \( G' \) be a minimal minor digraph of \((G,T)\). Then \( G' \) contains neither forward edges of \( T \) nor back edges of \( T \) and \( x \rightarrow y \in E' \) is a cross edge of \( T \) only if \( x \) and \( y \) are brothers in \( T \).

Lemma 5.13: Let \( T \) be a directed spanning tree of a digraph \( G=(V,E) \) then the minimal minor digraph \( G'=(V,E') \) of \((G,T)\) is unique and can be obtained by a finite number of elementary reduction operations.

Observation 5.14: A directed circuit in a minimal minor digraph contains only cross edges.

This leads us to the next lemma.

Lemma 5.15: Let \( T \) be a directed spanning tree of a digraph \( G'=(V,E) \) and let \( G \) be a minimal minor digraph of \((G',T)\). Then \( T \) is a \( T \)-DFS in \( G \) if and only if \( G \) is acyclic.

The algorithm for checking whether a given directed spanning tree \( T \) is a \( T \)-DFS has two phases. In phase one we build the minimal minor digraph and in phase two we check whether it is acyclic. The second phase can be done with time complexity \( O(|E|) \) using the algorithm of Tarjan [Ta] for finding strongly connected components (clearly, a digraph \( G'=(V',E') \) is acyclic if and only if it has \( |V'| \) different strongly connected components).

The structure of the algorithm is as follows:

DI_CHECK 
Input: A digraph \( G \) and a directed spanning tree \( T \) in \( G \).
Output: A decision whether \( T \) is a \( T \)-DFS in \( G \).

PHASE ONE: BMM \((G,T)\) "Build Minimal Minor"
Input: A digraph \( G \) and a directed spanning tree \( T \) in \( G \).
Output: The minimal minor digraph of \((G,T)\).
(1) Creating a minor digraph \( G' \) of \((G,T)\) where \( G' \) consists of the tree and all the cross edges of \( G \) (i.e. deleting all the forward and back edges in \( G \)).
Creating a minor digraph $G_2$ of $(G_1, T)$ by using 5.7.2 for every cross edge $e$ in $G_1$. $G_2$ is the output of phase one ($G_2=\text{BMM}(G,T)$).

**PHASE TWO: CNSCC($G_2$) (Count the Number of Strongly Connected Components) [Ta]**

Input: A digraph $G_2$ (the output of phase one).

Output: The number of strongly connected components in $G_2$.

Decision (of algorithm DI_CHECK): $T$ is a $T$--DFS in $G$ if and only if $\text{CNSCC}(G_2)=|V|$.

Lemma 5.16: $\text{BMM}(G,T)$ computes the (unique) minimal minor digraph of $(G,T)$.

We now present an efficient sequential implementation of algorithm BMM.

Step 1 of the algorithm is done by using a DFS algorithm in $G$. In step 2 of the algorithm we want to replace each cross edge $e=x\rightarrow y$ by another cross edge $R[e]=\hat{x}\rightarrow \hat{y}$ where $x$ and $y$ are in the brother sub trees $T_i$ and $T_j$ respectively and $R$ is an array indexed by the cross edges of $G_1$.

First we find the lowest common ancestors of $(x,y)$ for every cross edge $e=x\rightarrow y$ in $G_1$ (by using the algorithm in [AHU]). The results are organized in an array $LCA$ indexed by the cross edges of $G_1$.

After computing $LCA$ we use a modification of the DFS algorithm for computing $R$ as follows:

**CNCE($G_1,T,LCA$) (Compute the New Cross Edges)**

Input: A digraph $G_1=(V,E)$ (the output of step 1), $T$ (a spanning tree in $G_1$), and an array $LCA$ (computed as above).

Output: An array $R$ indexed by the cross edges of $G_1$. For every cross edge $e=x\rightarrow y$ in $G_1$ where $x$ and $y$ are in the brother sub trees $T_i$ and $T_j$ respectively, $R[e]=\hat{x}\rightarrow \hat{y}$.

1. Mark all the edges of $T$ "unused". $v:=r$ (where $r$ is the root of $T$).
2. If all the tree edges emanating from $v$ are used then go to (4).
3. Choose an unused tree edge $v\rightarrow u$. Mark $e$ "used". $f(u):=v$, $s(v):=u$, $v:=u$ and go to (2).
4. For every cross edge $e$ where $v$ is either the tail or the head of $e$ do begin\[z:=LCA[e], \hat{v}:=s(z)\mathrm{ (clearly, \ s(z)\ is \ second-\pi_{x,y})}, \]
   If $v$ is tail($e$) then tail($R[e]$):=$\hat{v}$
   Else ($v$ is head($e$)) head($R[e]$):=$\hat{v}$.
   end.
5. If $v=r$ then $v:=f(v)$ and go to (2).
   Else ($v=r$, all the vertices have been scanned) halt.

$R$ is the output of algorithm CNCE ($R=\text{CNCE}(G_1,T,LCA)$).

After computing $R$ we create a digraph $G_2=(V,\hat{E} \cup R)$ which is the output of phase 2. $G_2$ is the result of replacing every cross edge $e$ in $G_1$ by the cross edge $R[e]$ and is the minimal minor digraph of $(G_1,T)$.

Lemma 5.17: $\text{BMM}(G=(V,E),T)$ has time complexity $O(1E1)$.

Proof: It is obvious that the complexity of step 1 is $O(1E1)$.

As for step 2, the computation of $LCA$ is done using the algorithm of [AHU] for finding lowest common ancestors in a static tree in an off line model. This algorithm with the improvement of [GT] has time complexity $O(1E1)$ as stated in [GT]. The computation of $R$ (algorithm CNCE) is in fact a modified DFS algorithm and has time complexity $O(1E1)$. The rest of step 2 (the creation of $G_2$) is linear in the number of cross edges of $G$. □
Since both phases of \( DI_\text{CHECK} \) are linear in the number of edges of \( G \) we can conclude:

Corollary 5.18: Algorithm \( DI_\text{CHECK} (G=(V,E),T) \) is correct and has time complexity \( O(\text{I } E) \). \( \square \)

6. PARALLEL RECOGNITION of DFS TREES in DIGRAPHS

In this section we describe how to implement algorithm \( DI_\text{CHECK} (G,T) \) in \( O(\log^2 n) \) time with \( O(n^{2.81}) \) processors on a CREW PRAM (where \( n \) is the number of vertices in \( G \)). The algorithm has two phases which are identical to the phases of the algorithm presented in section 5. In phase one we build the minimal minor digraph and in phase two we check whether it is acyclic. Phase two can be done efficiently using parallel implementation of matrix multiplication (note that a digraph is acyclic if and only if it has no directed paths of length \( n \)). For example, by using Strassen method (see [Si]) we get \( O(\log^2 n) \) time complexity using \( O(n^{2.81}) \) processors.

The implementation of phase one (algorithm BMM) is as follows:

(i) Compute \( z(e) \), the lowest common ancestor of \( x \) and \( y \) in \( T \) for every non tree edge \( e=x \rightarrow y \) in \( G \).

(ii) Delete the back and forward edges of \( G \) (step 1 of algorithm BMM). Note that \( e=x \rightarrow y \) is a forward edge if and only if \( z(e)=x \) and \( e \) is a back edge if and only if \( z(e)=y \).

(iii) (Step 2 of BMM) Replace every cross edge \( x \rightarrow y \) by another cross edge \( \hat{x} \rightarrow \hat{y} \) where \( x \) and \( y \) are in the brother sub trees \( T_x \) and \( T_y \), respectively.

Lemma 6.1: The implementation of algorithm BMM \( (G,T) \) has \( O(\log n) \) time complexity where \( O(m) \) processors are used on a CREW PRAM (\( m \) is the number of edges in \( G \)).

Proof: We use the algorithm of [Vi]. The complexity analysis is similar to that of Theorem 4.7. \( \square \)

Corollary 6.2: Given a digraph \( G \) and a directed spanning tree \( T \) in \( G \) we can check in \( O(\log^2 n) \) time with \( O(n^{2.81}) \) processors (where \( n \) is the number of vertices in \( G \)) on a CREW PRAM whether \( T \) is a \( T\text{-DFS} \) in \( G \). \( \square \)

Note: Improving the complexity of checking the acyclicity of a digraph will improve the complexity of our solution.

7. OPEN PROBLEMS

7.1: We have shown that on one hand the \( DRDT \) problem is \( NP-C \) even for the case where \( g = (n-1,\ldots,n-1,k,n-1,\ldots,n-1) \). On the other hand the problem can be solved by efficient and simple algorithms in the case where \( g = (2,2,\ldots,2) \). It is left open to classify the complexity of this problem for all vectors between those two extreme cases.

7.2: Is it possible to improve the speed up of the parallel solutions for the problems solved here?

7.3: Extensions of the problems to mixed graphs.
ACKNOWLEDGMENT

This research was partially supported by Technion V.P.R. Fund No. 120-0683

REFERENCES


