# A Parallel Unrestricted Depth Search Algorithm

E. N. Cáceres*
Universidade Federal do Mato Grosso do Sul
Depto. de Computação e Estatística
Campo Grande, MS, 79069, Brazil

S. W. Song†
Universidade de São Paulo
Inst. de Matemática e Estatística
Depto. de Ciência da Computação
São Paulo, SP - 05508-900 - Brazil

J. L. Szwarcfiter‡
Universidade Federal do Rio de Janeiro
Instituto de Matemática and
Núcleo de Computação Eletrônica
Rio de Janeiro, RJ, 21945-970, Brazil

**Abstract** *Unrestricted depth search is an important technique for the design of parallel algorithms. In this paper we present a parallel algorithm for the unrestricted depth search of a graph of $n$ vertices and $m$ edges. We consider the Coarse-Grained Multicomputer Model (CGM) and show that the proposed algorithm requires $O(\log p)$ communication rounds, where $p$ is the number of processors, independent of $n$.*

*Keywords:* Parallel algorithms, coarse-grained multicomputer, unrestricted depth search

## 1 Introduction

In this paper we present a new CGM/BSP parallel algorithm for performing unrestricted depth search in a graph.

Depth search and breadth search in a graph $G = (V, E)$ starting from a given vertex $v \in V$ determines a tree that corresponds to the execution of the respective search method. In these search methods, each edge $e \in E$ is traversed a *constant* number of times. We use the term *restricted* to identify a search method with such a characteristic.

On the other hand, we define *unrestricted search* in a graph $G = (V, E)$ from a given vertex as a systematic process of traversing $G$ in such a way each edge is visited a *finite* number of times. The only restriction is that the process terminate.

Unrestricted depth search has been shown to be an important technique for the design of parallel algorithms. It has been used to generate all the maximal cliques in a circle graph [3].

The sequential algorithm for unrestricted depth search is based on the (restricted) depth search algorithm [9]. The problem of parallel restricted depth search does not appear to have a simple solution as in the sequential case. At the moment the only known solutions in $NC$ are for particular kinds of graphs [7, 11]. In general, it has been proven that lexicographic depth search is *P-complete* [8].

The unrestricted search in a graph generates all the maximal paths starting from a given vertex $r$ called the root of the search. The problem of determining a maximal path is as follows. Given a graph $G = (V, E)$ and a ver-

*Partially supported by PRONEX-SAI.

†Partially supported by FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) Proc. No. 98/06138-2, CNPq Proc. No. 52.3778/96-1 and 46.1230/00-3, and CNPq/NSF Collaborative Research Program Proc. No. 68.0037/99-3.

‡Partially supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico, CNPq, and Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro, FAPERJ, Brazil.

tex $r \in V$, determine a simple path $C$ starting at $r$ such that $C$ cannot be extended without finding a vertex that already belongs to $C$.

The algorithm for computing a maximal path with a greedy method does not have an efficient parallelization [2].

To design a parallel algorithm for unrestricted search, we must avoid the problems that occur when the path encounters a cycle in the graph. Furthermore, we need to know in what parts of the graph the path is located at each step of the algorithm. For this purpose, we use a decomposition of the graph called *kl*-paths, to be seen later.

We consider a more restricted class of graphs, namely the acyclic digraphs (directed graphs). For the acyclic digraphs, we can find a simplified version of the algorithm since we do not have to worry about the extension of the maximal paths between different cycles. To our knowledge there are no known parallel coarse-grained algorithms for performing the unrestricted depth search. The proposed algorithm is based on the coarse-grained multicomputer (CGM) model. We present a CGM algorithm for unrestricted depth search that requires $O(\log p)$ communication rounds, where $p$ is the number of processors, independent of the input size of the given graph.

## 2 Coarse-Grained Multicomputer (CGM) Model

The CGM model was proposed by Dehne [6]. See a nice discussion of this model in [5]. It is similar to the BSP (bulk-synchronous parallel) model [10]. However, it uses only two parameters: the input size $n$ and the number of processors $p$. Each processor has a local memory of size $O(n/p)$. The term *coarse grained* means the size of the local memory is much larger than $O(1)$. Dehne et al. [6] define "much larger" as $n/p \geq p$. Each processor is connected by a router that can send messages in a point-to-point fashion. A CGM algorithm consists of an alternate sequence of computing and communication rounds separated by a barrier synchronization.

A computing round is equivalent to a superstep in the BSP model. In a computing round, we usually use the best sequential algorithm in each processor to process its data locally. In a communication round, each processor exchanges a maximum of $O(n/p)$ data with other processors. In the CGM model, the communication cost is modeled by the number of communication rounds. Some algorithms for computational geometry and graph problems require a constant number or $O(\log p)$ communication rounds [4, 6]. Contrary to a PRAM algorithm, which is often designed for $p = O(n^k)$, with $k \in \mathcal{N}$, and with each processor receiving a small number of input data, in this model we consider the more realistic cases where $n \gg p$. The CGM model is particularly suitable in current parallel machines in which the global computing speed is considerably larger than the global communication speed.

## 3 A Sequential Algorithm for Unrestricted Search in Acyclic Digraphs

In a depth search of a graph, each edge is visited a constant number of times. Such a search process is known as a restricted search. An unrestricted search in a graph $G$ is a systematic process of traversing $G$ such that each edge is visited a finite number of times. The only restriction is that the process terminate.

The sequential unrestricted depth search of a connected graph $G = (V, E)$ is given as follows. The algorithm starts with a given vertex $v$ chosen as root. It uses a stack $Q$.

**Unrestricted Search** $(v)$
Mark $v$
Push $v$ onto stack $Q$
For each vertex $w$ adjacent to $v$ do
    if $w$ is not in $Q$ then
        visit edge $(v, w)$
        Unrestricted Search $(w)$
Pop $v$ from $Q$
Unmark $v$

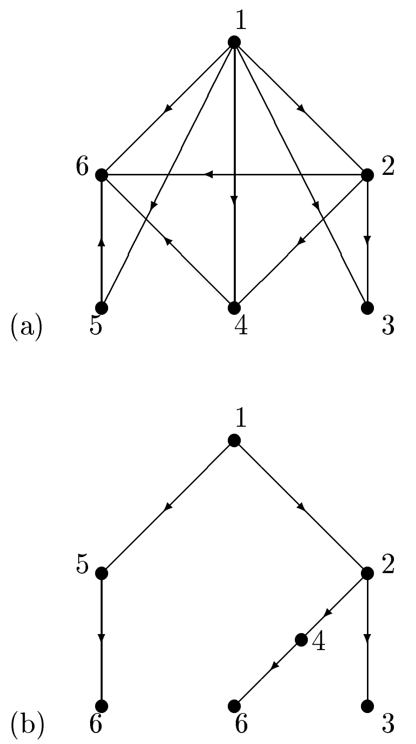


Figure 1: (a) An Acyclic Digraph $G$ and (b) The Maximal Paths Produced by Unrestricted Depth Search

As an example, consider the acyclic digraph of Figure 1 (a). If we run unrestricted depth search on this graph starting at vertex 1, we get Figure 1 (b).

We will present a CGM algorithm for the unrestricted depth search. First we need some definitions to be given in the next section.

# 4 Notation and Definitions

Given a digraph $G = (V, E)$, for each vertex $v \in V$ we define the *successor* of $v$ (suc[$v$]) as a fixed element of the adjacency list of $v$. Let $e = (u, v) \in E$. We define the *successor* of $e$ as the edge ($v$,suc[$v$]). A $kl$-path is a path $C$ in $G$ with initial edge $e = (k, l)$. An edge $e \in E$ belongs to $kl$-path $C$ if $e$ is the successor of some edge on the path $C$, and $e \notin C$. An edge $(u, v)$ can belong to more than one $kl$-path.

For a undirected graph $G$, we consider each edge $(u, v)$ as two distinct directed edges $(u, v)$ and $(v, u)$.

Since the successor of each vertex is fixed, all the edges on a $kl$-path incident with a vertex $v$ have a same successor ($v$,suc[$v$]). A $kl$-path can be a simple path, a cycle, a path together with a cycle or the union of two cycles. In the case when $G = (V, E)$ is an acyclic digraph, the $kl$-paths are formed by simple paths, with an initial vertex $k$ and a final vertex $t$.

# 5 The Parallel Algorithm

We present below an algorithm for unrestricted depth search in acyclic digraphs. The algorithm computes all maximal simple paths from a given root $r$.

The algorithm initially decomposes the digraph $G = (V, E)$ into a set of $kl$-paths. This decomposition is obtained through the definition of the successors of vertices and edges of $G$. The algorithm initializes the successor of each vertex $v \in V$ as the first element of its adjacency list of $v$. After the decomposition, the algorithm explores the edges of the $kl$-path such that $k = r$, where $r$ is the root of the search and $l =$ suc[$r$]. The visited edges form a maximal simple path $C$. Once we have determined a maximal simple path $C = \{v_0, \cdots, v_p\}$, for $r = v_0$ on the $kl$-path, we can obtain a new maximal simple path, if it exists, as follows. We determine the last vertex $v_i \in C$ that has some vertex in its adjacency list that has not been visited yet. The successors of the vertices $v_j \notin \{v_0, \cdots, v_i\}$ are modified to be the first element of the adjacency list of each $v_j$, and the successor of $v_i$ is altered to be the element of the adjacency list of $v_i$ immediately following suc[$v_i$]. The successors of the vertices $\{v_0, \cdots, v_{i-1}\}$ remain unaltered. The new definition of successors determines a new decomposition of the digraph into a set of $kl$-paths. We determine a simple path $C = \{v_i, \cdots, v_t\}$ on a $kl$-path in the new decomposition, with $k = v_i$ and $l =$ suc[$v_i$]. The path $C = \{v_0, \cdots, v_i, \cdots, v_t\}$ formed by the union of the paths $\{v_0, \cdots, v_i\}$ and $\{v_i, \cdots, v_t\}$ is a new maximal simple path. The remaining maximal simple path, if they exist, are computed analogously.

All the vertices $w_i$ that are not reachable from the root vertex $r$ are not included in the search.

**Algorithm: Unrestricted Search**

1. Define a root $r$, construct the adjacency lists of $G$ and initialize the maximal simple path $CM$.
2. Decompose the digraph $G$ into a set of $kl$-paths.
3. Determine a simple path $C$ from $r$ on the $kl$-path, with $k = r$ and $l = \mathrm{suc}[r]$.
4. Compute the maximal simple path $CM = CM \cup C$. Verify the existence of any vertex $v_i \in CM$ that has in its adjacency list a vertex that has not been visited yet.
5. In case $v_i$ exists, $CM = \{v_0, \cdots, v_{i-1}\}$. Alter the successors of the vertices $v_i \notin CM$, and unmark the edges $e \in G$ that do not belong to the path $CM$.
6. Apply steps 2, 3, 4, and 5 to the set of unmarked edges of $G$ for $r = v_i$ until all possible maximal simple paths from $v_0$ have been explored.

First, we rank the adjacency lists of all $v \in V$ and define the successor of $v$ to be the first element of the adjacency list of $v$.

We start with the root $r$ and with the defined successors, this represents a maximal path in $G$. Since our graph is acyclic, using list ranking, we can rank all vertices of this path. After this, we mark the last vertex of the path and change the successor (if there is one) of the tail of this path and compute another maximal path. Otherwise, using a proper data structure, we can backtrack in this path and visit a vertex that was not visited before.

The unrestricted search in this acyclic graph can be done in $O(\log p)$ communication rounds.

# 6 Correctness

The correctness of the proposed algorithm is based on the following.

**Lemma 1** *Let $G = (V, E)$ be a weakly connected digraph. Let $v \in V$ and a $kl$-path in $G$ such that $k = v$ and $l = suc[v]$. Then the $kl$-path is:*

1. *a simple path;*
2. *a cycle; or*
3. *a path connected to a cycle.*

**Proof:** Let $C = \{v_1, \cdots, v_i\}$ be a simple path on the $kl$-path, $k = v_1$ and $l = v_2$. Extend the path $C$ until the successor $w$ of $v_i$. If $w \in C$, we obtain a cycle and by the fact that the successors are fixed, $suc[w] \in C, w = suc[v_i]$. If $w \neq v_1$, then the $kl$-path is a simple path connected to a cycle. If we can extend $C$ and do not obtain a cycle, then as the number of vertices is finite, there exists a sink on the $kl$-path. In this case we obtain a simple path.

**Lemma 2** *Let $G = (V, E)$ be an acyclic weakly connected digraph. Then each $kl$-path is a simple maximal path.*

**Proof:** Based on the fact that the digraph is acyclic, the determined path is simple. Let $v_p$ be the last vertex of the path determined on the $kl$-path. Then $v_k$ is a sink, since otherwise there would exist an element on the adjacency list of $v_k$, and this new edge would be part of the path.

**Lemma 3** *Let $G = (V, E)$ be a connected undirected graph, applied to the algorithm of unrestricted search. Let $v \in V$ included $i$ times in the array $C$ during the process. Denote by $C_i$ the configuration of $C$ that precedes $C[j] = v$, after the inclusion of $v$ in $C$ by the $i$-th time. Then we have necessarily:*

1. *The configuration of $C$, when $v$ is excluded by the $i$-th time from $C$, is also equal to $C_i$.*

*2.* $i \neq j \Rightarrow C_i \neq C_j$.

**Proof:** The array $C$ simulated a stack in the algorithm of unrestricted search. A vertex $v$ is only included in $C$ if it is unmarked. Therefore the configuration of $C$ when $v$ is excluded by the $i$-th time is equal to $C_i$. A vertex $v$ will only be re-explored, in case there exists a path from a vertex $u$ that precedes $v$ in $C$, and the vertices of the path do not belong to $C$. Therefore if $i \neq j$ then $C_i \neq C_j$. This concludes the proof.

The above lemma ensures the termination of the algorithm, because if a vertex $v$ is inserted in $C$ more than once, then necessarily the configuration in $C$, below $v$, is different in each case. Since there exists a finite number of ways to arrange these configurations, the algorithm necessarily reaches the end. This property is based strongly on the fact that a marked vertex cannot be re-explored. The re-exploration of a marked vertex can cause the appearance of cycles.

**Theorem 1** *Let $G = (V, E)$ be a connected graph. Any path $C$ encountered by the algorithm in $G$ starting from the root vertex $r$ in a $kl$-path with $k = r$ is maximal.*

**Proof:** The exploration of a $kl$-path with $k = r$ terminates when a cycle is detected. The vertices of the simple path encountered on the $kl$-path are stored in $C$. The algorithm determines the vertex most distant from $r$ in $C$ that possesses some adjacency that can be explored.

In case the vertex with some adjacency that can be explored is the last vertex of $C$, the algorithm alters the adjacencies of the vertices that do not belong to $C$ and the algorithm extends the path $C$ until a new cycle is detected.

In case the vertex with some adjacency que can be explored is different from the last vertex in $C$, then $C$ is a maximal path.

**Theorem 2** *Let $G = (V, E)$ be connected graph. Any maximal path of $G$ starting at the root vertex $r$ will be found by the algorithm at least once.*

**Proof:** Once a maximal path $C$ is found from $r$, the algorithm determines the vertex $v_i$ in $C$ that is most distant from $r$. All the vertices after $v_i$ in $C$ are removed from $C$ and all the adjacencies of the vertices that do not belong to $C$ are reinitialized. The adjacency of vertex $v_i$ is increased by one, and the graph is decomposed into a new set of $kl$-paths, with $k = r$.

The path $C$ obtained in the new $kl$-path is different from the previous one, because the vertex to be included in $C$ after $v_i$ is different from the vertex previously in $C$.

Since we explore all the possible adjacencies of each path $C$, all the simple maximal paths are computed.

**Theorem 3** *Let $G = (V, E)$ be a connected graph. No maximal path of $G$ starting at the root vertex $r$ will be found by the algorithm more than once.*

**Proof:** By the previous lemma, we have: if a vertex $v$ is included in a path more than once, the configuration of the vertices that precede $v$ are different. Therefore the algorithm does not compute the same path more than once.

## References


[1] R. Anderson. A Parallel Algorithm for the Maximal Path Problem. *Combinatorica* **4** (1987) 315–326.

[2] R. Anderson e Mayr. Parallelism and Greedy Algorithms. Technical Report STAN-CS-84-1003, Computer Science Department, Stanford University Bonn, (1984).

[3] E. N. Cáceres, S. W. Song and J. L. Szwarcfiter. A Coarse-Grained Parallel Algorithm for Maximal Cliques in Circle Graphs. The 2001 International Conference on Computational Science. *Lecture Notes in Computer Science*, V. Alexandrov, J. Dongarra, K. Tan (eds), Springer-Verlag. San Francisco, May 28-30, 2001. To appear.

[4] E. Cáceres, F. Dehne, A. Ferreira, P. Flocchini, I. Rieping, A. Roncato, N. Santoro, and S.W. Song. Efficient Parallel Graph Algorithms For Coarse Grained Multicomputers and BSP. Proceedings ICALP '97 - 24th International Colloquium on Automata, Languages, and Programming. P. Degano et al. (eds). *Lecture Notes in Computer Science* **1256** (1997) 390–400.

[5] F. Dehne (Ed.), Coarse Grained Parallel Algorithms. In: Special Issue of *Algorithmica* **24** (1999) 173–426

[6] F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable Parallel Computational Geometry for Coarse Grained Multicomputers. *Proc. ACM 9th Annual Computational Geometry* (1993) 298–307.

[7] R. K. Ghosh and G.P. Bhattacharjee. A Parallel Search Algorithm for Directed Acyclic Graphs. *BIT* **24** (1984) 90–90.

[8] R. M. Karp and V. Ramachandran.: Parallel Algorithms for Shared-Memory Machines. In: J. van Leeuwen (ed.): *Handbook of Theoretical Computer Science* Vol. A. Chapter 17, The MIT Press/Elsevier (1990) 869–941.

[9] R.E. Tarjan. Depth First Search and Linear Graph Algorithms. *SIAM J. Computing* **1** (1972) 146–160.

[10] L. G. Valiant.: A Bridging Model for Parallel Computation. *Communication of the ACM* **33** (1990) 103–111.

[11] Y. Zhang. *Parallel Algorithms for Problems Involving Directed Graphs.* PhD thesis, Department of Computer Science, Drexel University, 1990.