Improved processor bounds for parallel algorithms for weighted directed graphs *

Nancy Amato

Department of Computer Science and Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

Communicated by G.R. Andrews
Received 1 October 1991
Revised 13 October 1992

Abstract

Amato, N., Improved processor bounds for parallel algorithms for weighted directed graphs, Information Processing Letters 45 (1993) 147-152

We present a parallel algorithm that solves the single-source shortest path problem (SSSP) for a weighted digraph \( G = (V, E) \) in time \( O(\log^3 n) \) using \( M(n) \) processors on an exclusive-read exclusive-write parallel random access machine (EREW PRAM), where \( n = |V| \), edge weights are drawn from the set \( \{0, 1, \ldots, k\} \) for some constant \( k \), and \( M(n) \) is the number of processors necessary to multiply two \( n \times n \) integer matrices over a ring in \( O(\log n) \) time (currently, \( M(n) = n^{2.376} \)). This algorithm is a generalization of the \( O(\log^3 n) \) time, \( M(n) \) processor EREW PRAM algorithm due to Gazit and Miller for the SSSP problem in an unweighted digraph. We also briefly explain how our solution of the SSSP problem for a weighted digraph can be used to reduce the previous known processor bounds for a number of digraph problems to \( M(n) \) from \( \Theta(n^3) \) (within a polylog factor) without increasing the running time.

Keywords: Weighted directed graph; single-source shortest paths; parallel algorithms; matrix multiplication

1. Introduction

In this paper we use the parallel random access machine (PRAM) model of parallel computation, in which it is assumed that each processor has random access to any shared memory location in unit time. In the EREW PRAM model, different processors may not access the same common memory location when reading or writing; in the CREW and CRCW PRAM models, concurrent reads and concurrent reads and writes, respectively, are allowed. The commonly held view is that a PRAM algorithm should attempt to solve the problem in time \( O(\log^4 n) \) for some constant \( k \) (often referred to as polylog time) using \( O(n^{O(1)}) \) processors, where \( n \) is the size of the input; the class NC consists of those problems that can be solved within these bounds. In general, an NC algorithm is said to be optimal if the product of the time and the number of processors used (processor-time product) equals the sequential complexity of the problem. Similarly, an NC algorithm is said to be efficient if the processor-time product exceeds the sequential complexity of the problem by at most a polylog factor. For a detailed treatment of the various PRAM models and the class NC consult [6].
Although efficient parallel algorithms exist for many problems concerned with undirected graphs, the same is not true when dealing with directed graphs (digraphs). The main reason for this disparity is that currently there are more efficient techniques for computing reachability information in undirected graphs than in directed graphs. In particular, although there are optimal techniques for deciding whether one vertex is reachable from another in an undirected graph (see, e.g., [6]), the most efficient method known today for determining this information in a digraph is to compute the transitive closure of its incidence matrix by repeated matrix multiplication. Working over the semiring \((N, \min, +\rangle\), the standard matrix multiplication algorithm requires \(\Theta(n^3)\) operations. However, when working over a ring, or with Boolean matrices that may be embedded in a ring, there exist more economical techniques requiring \(O(n^\alpha)\) operations, where \(\alpha < 3\); currently the best of these achieves \(\alpha = 2.376\) [2]. All of these sequential techniques for matrix multiplication may be implemented efficiently in parallel; the standard algorithm requires \(O(1)\) time and \(O(n^3)\) processors on a CRCW PRAM, or \(O(\log n)\) time and \(O(n^3/\log n)\) processors on an EREW or CREW PRAM, and the more economical algorithms require \(O(\log n)\) time and \(O(n^\alpha)\) processors on an EREW or CREW PRAM. In this paper, we adopt the common convention of denoting \(n^\alpha\) by \(M(n)\).

Thus, although the parallel solution of many problems dealing with digraphs seems to require the transitive closure of the relevant incidence matrices, if the computation can be restricted to matrices over a ring, or Boolean matrices that can be embedded in a ring, then fewer processors will be required yielding a lower processor-time product. The first contribution to this effort was made by Gazit and Miller [4] when they provided an \(O(\log^2 n)\) time and \(M(n)\) processor EREW PRAM algorithm for solving the single-source shortest path (SSSP) problem for an unweighted digraph \(G = (V, E)\), where \(|V| = n\). The main contribution of this paper is an extension of the result of Gazit and Miller to include weighted digraphs in which edge weights are drawn from the set \(\{0, 1, \ldots, k\}\), for some constant \(k\); this generalized algorithm also runs in time \(O(\log^2 n)\) using \(M(n)\) processors on an EREW PRAM. The best previous solution to this more general problem used a parallel implementation of the standard matrix multiplication algorithm and required time \(O(\log^2 n)\) and \(O(n^3/\log n)\) processors in the EREW and CREW PRAM models. In addition, we provide a proof of correctness for the generalized algorithm that is simpler than the proof given in [4] for the original algorithm; consequently, we also supply a new, simpler proof of the technique of [4] for SSSP in an unweighted digraph. This new proof has the added benefit that it provides a more intuitive characterization of how the algorithm computes the shortest path. Finally, we explain how our solution of the SSSP problem for a weighted digraph can be used to reduce the previous known processor bounds for a number of digraph problems from \(O(n^3)\) (within a polylog factor) to \(M(n)\) without increasing the running time; the problems we consider are finding an ear-decomposition in a directed graph, the transitive reduction problem, and finding a minimum weight branching in a digraph in which the edge weights are non-negative integers bounded by some constant \(k\). Since the above problems are often useful techniques when solving digraph problems, it is anticipated that the results in this paper may prove useful for reducing processor bounds for other digraph algorithms as well.

2. The single-source shortest path problem

In a digraph \(G = (V, E)\) with source \(s\), the single-source shortest path (SSSP) problem is to find, for every vertex \(v \neq s\), the length of the shortest path from \(s\) to \(v\). Since the solution of the SSSP problem can be translated into a single-source breadth-first search (BFS) tree of the digraph \(G\) in constant time using \(O(|E|)\) processors, the SSSP and BFS problems can be viewed as slightly different instances of the same problem; we will refer to this problem as the SSSP/BFS problem. In this section, \(n\) denotes \(|V|\).

When working with matrices over the semiring \((N, \min, +\rangle\), the SSSP/BFS problem can be
solved in $O(\log n)$ time using $O(n^3)$ processors on a CRCW PRAM, or $O(\log^2 n)$ time with $O(n^3/\log n)$ processors on a CREW or EREW PRAM by repeated $(\min, +)$ matrix multiplication; this technique actually solves the all-pairs shortest path problem. Gazit and Miller have given an EREW PRAM algorithm that solves the SSSP/BFS problem for an unweighted digraph in time $O(\log^3 n)$ using $M(n)$ processors by performing the computations over a ring [4]. We now show how their method may be extended to solve the problem for a weighted digraph, where edge weights are drawn from the set $\{0, 1, \ldots, k\}$ for some constant $k$; this generalized algorithm also runs in time $O(\log^2 n)$ using $M(n)$ processors on an EREW PRAM.

Our algorithm for solving the SSSP/BFS problem for weighted digraphs, in which edge weights are in $\{0, 1, \ldots, k\}$ for a constant $k$, is described below. The central idea of this algorithm is a reduction from the original digraph with edge weights in the set $\{0, 1, \ldots, k\}$, to a digraph $G'$ with edge weights restricted to the set $\{0, 1\}$; an important property of this reduction is that the new digraph $G'$ has only $O(n)$ vertices. The reduction is accomplished in two stages: the first stage deals with edges with weights larger than one (Step 1), and the second stage processes those edges with weight zero (Step 2). In the following discussion we use $wt(x, y)$ and $d(x, y)$ to refer to the weight of edge $(x, y)$ and the shortest distance between vertices $x$ and $y$, respectively.

**Algorithm SSSP/BFS**

**Input:** A digraph $G$ with edge weights $w \in \{0, 1, \ldots, k\}$, for some fixed $k$, and a source $s$.

**Output:** The lengths of the shortest paths from $s$ to all other vertices in $G$.

**Step 1:** Transform $G$ into another weighted digraph $G' = (V', E')$ such that edge weights in $G'$ are drawn from the set $\{0, 1\}$, and the distances between the vertices of $G$ are preserved in $G'$. For each $v \in V$, let $E_v$ denote the edges of $G$ that originate at $v$, and let $W_v$ denote the set of unique weights associated with the edges in $E_v$.

(a) Initially $V' = V$ and $E' = \{(x, y) | (x, y) \in E, \text{ and } wt(x, y) \in \{0, 1\}\}$; these same edge weights are retained in $G'$.

(b) For all $v \in V$, and for all $w \in W_v$, add $l - 1$ new vertices to $V'$ to form a directed path of length $l - 1$ originating at $v$; assign each edge of this path a weight of one and denote its final vertex by $v_{l-1}$. For all $v, u \in V$ such that $(v, u) \in E$ and $wt(v, u) = l > 1$ in $G$, introduce edge $(v, u)$ of weight 1 in $G'$ (see Fig. 1). Let $n' = |V'|$.

**Step 2:** In this step we process the edges of weight zero; first all paths of length zero are computed, and then these paths are concatenated with all incident edges of weight one in $G'$ to identify all paths of length one. Let $C$ be the Boolean incidence matrix of the subgraph of $G'$ formed by including only those edges of weight 0. Identify all paths of length zero by computing $C^n$, the transitive closure of $C$. Next, pre and post multiply the Boolean incidence matrix of $G'$ by $C^n$; denote the resulting matrix by $B_0$. Note that $B_0$ identifies all edges and paths in $G'$ of weight or length, respectively, zero or one.

**Step 3:** Next, calculate approximations of the distances between all pairs of vertices as follows. Compute the first $[\log n']$ matrices of the form...
$B_{i+1} = B_i \ast B_i = B_i^2$, where $\ast$ represents multiplication of Boolean matrices embedded in a ring. For each matrix $B_i$, $0 \leq i \leq \lfloor \log n' \rfloor$ construct an approximation matrix $A_i$ as follows:

$$A_i[x, y] = \begin{cases} 0 & \text{if } C''[x, y] = 1, \\ 2^i & \text{if } B_i[x, y] = 1 \text{ and } C''[x, y] \neq 1, \\ \infty & \text{otherwise.} \end{cases}$$

Consider vertices $x, y \in V$ and let $l$ denote the length of the shortest path between $x$ and $y$ in $G$. Note that $A_i[x, y] = 0$, if and only if $l = 0$, $A_i[x, y] = 2^i$, if and only if $1 \leq l \leq 2^i$, and $A_i[x, y] = \infty$ if and only if $l > 2^i$. Thus, if $d(x, y) > 0$ and $j$ is the least value such that $A_j[x, y] = 2^i$, then $2^{j-1} < d(x, y) \leq 2^i$. Thus, the matrices $A_i$, $0 \leq i \leq \lfloor \log n' \rfloor$, provide rough approximations of the lengths of the shortest paths between all pairs of vertices in $G$.

**Step 4:** This step consists of $\lfloor \log n' \rfloor + 1$ stages so that after the $i$th stage ($i$ decreasing), $d(s, x)$ has been computed if it is a multiple of $2^i$, $\forall x \in V'$. Specifically, these distances are represented by the row vectors $V_i$, $\lfloor \log n' \rfloor < i < 0$, of length $n'$ in which $V_i[x] = d(s, x)$ if and only if $d(s, x)$ is a multiple of $2^i$, $\forall x \in V'$. Initially, $V_{\lfloor \log n' \rfloor + 1}[x] = 0$ and $V_{\lfloor \log n' \rfloor + 1}[x] = \infty$ for all $x \neq s$. For $i = \lfloor \log n' \rfloor$ down to 0, we compute $V_i' = V_{i+1}' \cdot A_i$, where $\cdot$ represents matrix multiplication over the semiring $(\mathbb{N} \cup \{\infty\}, \min, +)$.

The algorithm of [4] for solving the SSSP/BFS problem in an unweighted digraph essentially consists of Steps 3 and 4 of the above algorithm, with $n$ replacing $n'$, the incidence matrix of $G$ replacing $B_0$, and the condition $x = y$ replacing the condition $C'[x, y] = 1$.

**Theorem 1.** Algorithm SSSP/BFS can be implemented in time $O(\log^* n)$ using $M(n)$ processors on an EREW PRAM.

**Proof.** In Step 1, for each of the $n$ original vertices in $G$, we add an additional $O(k^2 n) = O(n)$ vertices; this procedure can be accomplished in $O(\log n)$ time using $O(k^2 n) = O(n)$ processors for each of the $n$ vertices of $G$. Clearly the graph $G'$ preserves the distances among the vertices of $G$; note that the obvious method of edge subdivision would not work because this approach could potentially add $O(|E| k) = O(n^2)$ vertices to $G'$ and would require $M(n^2)$ processors for the Boolean matrix multiplications rather than the desired $M(n)$. Computing the transitive closure of the (Boolean) matrix $C$ in Step 2, and constructing all $B_i$ matrices in Step 3, can be done in $O(\log^2 n)$ time using $M(n)$ processors. In Step 3, the transformation of the $B_i$ matrices into the approximation matrices $A_i$ can be accomplished in time $O(\log n)$ using $O(n^2)$ processors. Each of the multiplications over the semiring $(\mathbb{N} \cup \{\infty\}, \min, +)$ in Step 4 can be performed by making $n'$ copies of $V$, computing $n'^2 + n'$ operations, and then computing $n'^2 \cdot \min$ operations, each on a set of $n'$ numbers. Thus, Step 4 can be implemented in time $O(\log^2 n)$ using $O(n^2/\log n)$ processors. Hence, the total complexity of Algorithm SSSP/BFS is $O(\log^2 n)$ time using $M(n)$ processors. $\square$

The correctness of Algorithm SSSP/BFS is established by the following theorem which also provides a new proof of the correctness of the algorithm presented in [4] for SSSP/BFS in an unweighted digraph. This theorem has the added benefit that its proof is simpler than the corresponding proof of its counterpart in [4]; the main reason for this is that this theorem only requires that $V_i'[x] = d(s, x)$ if $d(s, x)$ is a multiple of $2^i$, whereas the theorem in [4] requires that $V_i'[x] = 2^i [d(s, x)/2^i]$, $\forall x \in V$. Furthermore, although this theorem is weaker, its proof provides a better characterization of the progress that the algorithm makes in computing the shortest paths. Specifically, if $d(s, x)$ is a multiple of $2^i$, but not of $2^{i-1}$, then in stage $i$ ($i$ decreasing) the algorithm calculates $d(s, x)$ from some previously calculated $d(s, w)$ and $A_i[w, x]$, where $d(s, w)$ is a multiple of $2^{i-1}$ (i.e., $V_{i+1}'[w] = d(s, w)$) and $d(w, x) = A_i[w, x] = 2^i$. Hence, if $d(s, y)$ is not a multiple of $2^{i-1}$, the value in $V_i'[y]$ is irrelevant to the computation at stage $i$ and thus there is no need to bound $V_{i+1}'[y]$. 

150
Theorem 2. If \( d(s, x) \) is a multiple of 2\(^i\), then \( V_i[x] = d(s, x) \), \( \forall x \in V' \) and \( \log n' \geq i \geq 0 \).

**Proof.** It is immediate to verify (i) \( A_i[x, y] = \min_{l \in [0, 2i]} \{ d(x, y) \leq l \}, \forall x, y \in V' \) and (ii) \( d(x, x) \leq V_i[x] \leq V_i+1[x], \forall x \in V' \) and \( 0 \leq i \leq \log n' \); these will be referred to as Facts (i) and (ii), respectively.

The statement of the theorem is proven by induction on \( i \), for \( i \) decreasing. Since \( V_{\log n'} \) is the 5th row of \( A_{\log n'} \), the basis is established by Fact (i). We now assume the statement holds for \( i + 1 \) and show that it holds for \( i \); let \( x \) be a vertex such that \( d(s, x) = c \cdot 2^i \).

We first note that if \( c \) is even, then \( d(s, x) = c/2 \cdot 2^{i+1} \) and thus by the hypothesis \( V_i[x] = d(s, x) \) and Fact (ii) ensure that \( V_i[x] = d(s, x) \). If \( c \) is odd, then there must be a vertex \( y \) that lies on a shortest path from \( s \) to \( x \) such that \( d(x, y) = 2^i \) and \( d(s, y) = d(s, x) - d(y, x) = c \cdot 2^i - 2^i = (c - 1) \cdot 2^i \). Moreover, since \( (c - 1) \) is even, \( d(s, y) = (c - 1)/2 \cdot 2^{i+1} \) and thus by the hypothesis, \( V_{i+1}[y] = d(s, y) \). Finally, \( V_i[x] \leq V_{i-1}[y] + A_i[y, x] = (c - 1) \cdot 2^i + 2^i = c \cdot 2^i = d(s, x) \) since, by Fact (i), \( A_i[y, x] = 2^i \); this and Fact (ii) ensures that \( V_i[x] = d(s, x) \). □

The proof of this theorem indicates that any additional generalization of this technique to include digraphs with polynomially bounded edge weights is unlikely without major alterations to the structure of the algorithm (i.e., Steps 3 and 4). This is due to the fact that if \( d(s, x) = c \cdot 2^i \) and odd, then when calculating \( d(s, x) \) at stage \( i \), the algorithm depends on the fact that there is some \( y \) that lies on a shortest path from \( s \) to \( x \) such that \( d(s, y) = (c - 1)/2 \cdot 2^{i+1} \) and \( d(y, x) = 2^i \); clearly this is unlikely to be the case in a digraph with polynomially bounded edge weights. (The technique employed here would not work because \( G' \) could have as many as \( O(nw_{\text{max}}^2) \) vertices, where \( w_{\text{max}} \) is the maximum edge weight.)

3. Applications of SSSP/BFS in weighted digraphs

In this section we list a few EREW and CREW PRAM algorithms in which Algorithm SSSP/BFS can be used to reduce the processor bounds from \( \Theta(n^3) \) (within a polylog factor) to \( M(n) \), without increasing the running time. The problems we consider are constructing a minimum weight branching of a digraph (in which edge weights are non-negative integers bounded by a constant), the transitive reduction problem, and constructing an ear-decomposition of a digraph. In addition, since breadth-first search, ear-decompositions, and minimum weight branchings are techniques that are used as subroutines when solving other digraph problems, it is possible that these results will imply similar processor reductions for other digraph algorithms as well.

A branching (also known as a forward or out-branching) of a digraph is a rooted spanning subgraph such that every vertex except the root has indegree 1; in a reverse or in-branching, every vertex except the root has outdegree 1. In a weighted digraph \( G = (V, E) \), a minimum weight branching is a collection of edges of \( E \) such that these edges form a branching of \( G \) and the sum of the weights on these edges is minimum. In \([7]\), Lovász shows that this problem, in which the edge weights are polynomially bounded, is in NC by giving a CRCW PRAM algorithm that finds a branching in time \( O(\log^2 n \log w) \) using \( O(n^3) \) processors, where \( w \) is the number of bits used to represent the weights on the edges; an EREW or CREW PRAM version of this algorithm would require at least \( O(\log^3 n \log w) \) time and \( O(n^3/\log n) \) processors. In \([1]\), we show that a slight variation of Lovász's algorithm can be implemented more efficiently when the edge weights are bounded by a constant; our technique uses Algorithm SSSP/BFS and an efficient algorithm for finding a spanning forest in an unweighted digraph \([6]\). This improved implementation yields an EREW PRAM algorithm requiring \( O(\log^3 n) \) time and \( M(n) \) processors for finding a minimum weight branching in a weighted digraph, where the weights are non-negative integers bounded by some constant \( k \).

Given a strongly connected digraph, the transitive reduction problem is to determine a minimal strongly connected spanning subgraph of it; this spanning subgraph is minimal in the sense that if any of its edges were removed it would not be
strongly connected. A definition of this problem and algorithms, both parallel and sequential, for its solution are given in [5]. The parallel algorithm runs in time $O(\log^4 n)$ with $O(n^3)$ processors on a CREW PRAM. The subtask in this algorithm responsible for the processor bound is the computation of a minimum weight branching in a digraph with edge weights of 0 and 1. Consequently, by using the minimum weight branching algorithm referred to above, the processors necessary can be reduced to $M(n)$ without increasing the running time.

An ear-decomposition of a digraph $G = (V, E)$ is a partition of $E$ into a sequence of ears (edge-disjoint simple directed paths or simple directed cycles), $D = [P_0, P_1, \ldots, P_k]$, such that $P_0$ is a simple directed cycle and, for $1 \leq i < k$, each endpoint of $P_i$ belongs to a smaller numbered ear while the interval vertices of $P_i$ do not belong to any smaller numbered ear. For undirected graphs, efficient ear-decomposition algorithms imply efficient solutions to a number of other problems such as 2-edge connectivity and biconnectivity [6]. It is known that a digraph is strongly connected if and only if it has an ear-decomposition [7]; this and the prospect of other applications, as in the undirected case, has lead to interest in developing ear-decomposition techniques for digraphs. Lovász shows that finding an ear-decomposition of a strongly connected digraph $G = (V, E)$ is in NC by giving a CRCW PRAM algorithm that runs in time $O(\log^2 n)$ using $O(n^4)$ processors [7]; the general strategy of this algorithm is to merge a forward and reverse branching of $G$ by an assignment of labels to the edges so that edges with the same label designate an ear. In [1], we show that Lovász's algorithm can be implemented in time $O(\log^2 n)$ using $M(n)$ processors on an EREW PRAM by using Algorithm SSSP/BFS and a judicious combination of well-known efficient parallel algorithms such as pointer jumping and the Euler tour technique [6]. Lucas and Sackrowitz [8] have proposed another $O(\log^2 n)$ time, $M(n)$ processor algorithm that is also similar to that of Lovász but uses a slightly different technique for labeling the edges; the advantage of their approach is that it requires SSSP/BFS only in an unweighted digraph, whereas that of Lovász uses SSSP/BFS in a weighted digraph.

Acknowledgment

The author would like to thank Vijaya Ramachandran for many useful discussions and suggestions.

References