A Divide-and-conquer Algorithm for Identifying Strongly Connected Components ⋆

Don Coppersmith [a] and Lisa Fleischer [a,2] and
Bruce Hendrickson [b] and Ali Pınar [c,1]

[a] *T. J. Watson Research Center, IBM*

[b] *Discrete Algorithms & Math Dept., Sandia National Laboratories*

[c] *Computational Research Division. Lawrence Berkeley National Laboratory*

Abstract

The strongly connected components of a directed graph can be found in an optimal linear time, by algorithms based on depth first search. Unfortunately, depth first search is difficult to parallelize. We describe two divide-and-conquer algorithms for this problem that have significantly greater potential for parallelization. We show the expected serial runtime of our simpler algorithm to be $O(m \log n)$, for a graph with n vertices and m edges. We then show that the second algorithm has $O(m \log n)$ worst–case complexity.

Key words: Design and analysis of algorithms, graph algorithms, parallel algorithms, strongly connected components, divide-and-conquer, discrete ordinates method.

1 Introduction

The decomposition of a directed graph into strongly connected components is a fundamental tool in graph theory with applications in compiler analysis, data mining, scientific computing, and other areas. Tarjan [18] devised an optimal

⋆ This work was funded by the Applied Mathematical Sciences program, U.S. Department of Energy, Office of Energy Research and performed at Sandia, a multiprogram laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the U.S. DOE under contract number DE-AC-94AL85000.

A preliminary version of the current work appeared in [8].

Email addresses: lkf@us.ibm.com (Lisa Fleischer), bah@sandia.gov (Bruce Hendrickson), apinar@lbl.gov (Ali Pınar).

[1] Supported by the Director, Office of Science, Division of Mathematical, Information, and Computational Sciences of U.S. Department of Energy under contract DE-AC03-76SF00098.

[2] Partially supported by NSF through CAREER grant CCR-0049071.

$O(m+n)$ algorithm for identifying strongly connected components for a graph with n vertices and m edges, which is based on depth first search (DFS) of the graph. Sharir [17]'s algorithm for identifying strongly connected components is also built on DFS and is widely used in textbooks as an example of the power of DFS [6]. Recently, Gabow [9] described an algorithm that avoids vertex labeling, which is again based on DFS.

For large problems, a parallel algorithm for identifying strongly connected components would be useful. One application of particular interest to us is discussed below. Unfortunately, DFS is difficult to parallelize. Reif showed that the lexicographical DFS problem is P-Complete [16]. However, Aggarwal and Anderson [1] and Aggarwal et al. [2] describe randomized NC algorithms for finding a DFS of undirected and directed graphs, respectively. The expected runtime of this latter algorithm is $O(\log^7 n)$ and it requires an impractical $O(n^{2.376})$ processors. To our knowledge, the deterministic parallel complexity of DFS for general, directed graphs is an open problem. Chaudhuri and Hagerup studied the problem for acyclic [5] and planar graphs [11], respectively. More practically, DFS is a difficult operation to parallelize and we are aware of no algorithms or implementations that perform well on large numbers of processors. Consequently, the utility of Tarjan's algorithm in parallel is questionable.

Alternatively, there exist several parallel algorithms for the strongly connected components problem (SCC) that avoid the use of depth first search. Gazit and Miller devised an NC algorithm, which is based upon matrix–matrix multiplication [10]. This algorithm requires $O(n^{2.376})$ processors and $O(\log^2 n)$ time. Kao developed a more complicated NC algorithm for planar graphs that requires $O(\log^3 n)$ time and $n/\log n$ processors [12]. More recently, Bader has an efficient parallel implementation of SCC for planar graphs [3], which uses a *packed–interval* representation of the boundary of a planar graph. When n is much larger than p, the number of processors, Bader's approach has been observed to have an $O(n/p)$ performance in practice [3]. But Bader's approach does not apply to general graphs.

Our interest in the SCC problem is motivated by the discrete ordinates method for modeling radiation transport. In this method, the object to be studied is modeled as a union of polyhedral finite elements. Each element is a vertex in our graph and an edge connects any pair of elements that share a face. The radiation equations are approximated by an angular discretization. For each angle in the discretization, the edges in the graph are directed to align with the angle. The computations associated with an element can be performed if all its predecessors have been completed. Thus, for each angle, the set of computations are sequenced as a topological sort of the directed graph. A problem arises if the topological sort cannot be completed, i.e., the graph has a cycle. If cycles exist, the numerical calculations need to be modified, typically by using old information along one of the edges in each cycle, thereby removing the dependency. So

identifying strongly connected components quickly is essential. Since radiation transport calculations are computation- and memory-intensive, parallel implementations are necessary for large problems. Also, since the geometry of the grid can change after each time-step for some applications, the SCC problem must be solved in parallel. Efficient parallel implementations of the topological sort step of the radiation transport problem have been developed for *structured* grids, oriented grids that have no cycles [4,7]. Some initial attempts to generalize these techniques to unstructured grids are showing promise [14,15]. It is these latter efforts that motivated our interest in the SCC problem.

In this paper, we describe simple divide-and-conquer algorithms for finding strongly connected components. We show one algorithm has expected serial complexity $O(m \log n)$, and a modification has worst case serial complexity $O(m \log n)$. Our approach has good potential for parallelism for two reasons. First, the divide-and-conquer paradigm generates smaller problems that can be solved independently. Second, the basic step in our algorithm is a reachability analysis, which is similar to topological sort in its parallelizability. In fact, McLendon, et al. [13] have implemented a parallel version of our algorithm and they report very encouraging results.

2 Preliminaries

Let $G = (V, E)$ be a directed graph with vertex set V and directed-edge set E. Let $n = |V|$ and $m = |E|$. An edge $(i, j) \in E$ is directed from vertex i to vertex j. A vertex v is *reachable* from a vertex u if there is a sequence of directed edges $(u, x_1), (x_1, x_2), \ldots, (x_k, v) \in E$. A vertex is reachable from itself. A *strongly connected component* of G is a maximal subset S of V such that each vertex in S is reachable from every other vertex in S. The set of strongly connected components of G is denoted $SCC(G)$. Note that $SCC(G)$ is a partition of V. The unique element of $SCC(G)$ that contains v is denoted $SCC(G, v)$. $V \setminus X$ denotes the subset of vertices in V which are not in a subset X. The size of vertex set X is denoted by $|X|$. Given a vertex $v \in V$, the *descendants* of v, $Desc(G, v)$, is the subset of vertices in G that are reachable from v. Similarly, the *predecessors* of v, $Pred(G, v)$, is the subset of vertices from which v is reachable. The set of vertices that is neither reachable from v nor reach v is called the *remainder*, denoted by $Rem(G, v) = V \setminus (Desc(G, v) \cup Pred(G, v))$. Given a graph $G = (V, E)$ and a subset of vertices $V' \subseteq V$, the *induced subgraph* $G' = (V', E')$ contains all edges of G connecting vertices of V', i.e. $E' = \{(u, v) \in E : u, v \in V'\}$. We will use $\langle V' \rangle = G' = (V', E')$ to denote the subgraph of G induced by vertex set V'.

Lemma 2.1 *Let G be a directed graph with vertex v. Then $SCC(G, v) = Desc(G, v) \cap Pred(G, v)$; and every other strongly connected component in $SCC(G)$ is a strongly connected component of exactly one of $\langle Desc(G, v) \setminus SCC(G, v) \rangle$, $\langle Pred(G, v) \setminus SCC(G, v) \rangle$, or $\langle Rem(G, v) \rangle$.*

```
DCSC(G)
    If G has no edges then
        forall v ∈ V Output {v}.
    Else
        Select a random vertex v from V
        SCC ← Pred(G, v) ∩ Desc(G, v)
        Output SCC
        DCSC(⟨Pred(G, v) \ SCC⟩)
        DCSC(⟨Desc(G, v) \ SCC⟩)
        DCSC(⟨Rem(G, v)⟩)
```

Fig. 1. A divide-and-conquer algorithm that outputs all of the strongly connected components of a directed graph.

Proof: The equality $SCC(G, v) = Desc(G, v) \cap Pred(G, v)$ follows immediately from the definitions. Let u and w be two vertices of the same strongly connected component in G. By definition, u and w are reachable from each other. The proof involves establishing $u \in Desc(G, v) \iff w \in Desc(G, v)$ and $u \in Pred(G, v) \iff w \in Pred(G, v)$, which then implies $u \in Rem(G, v) \iff w \in Rem(G, v)$. Since the proofs of these two statements are symmetric, we give just the first: If $u \in Desc(G, v)$ then u must be reachable from v. But then w must also be reachable from v, so $w \in Desc(G, v)$. ∎

3 A Divide-and-conquer Algorithm

In this section, we describe a divide-and-conquer algorithm that finds all of the strongly connected components of a directed graph and show that it has expected run time $O(m \log n)$. The algorithm, called DCSC (for Divide-and-conquer Strong Components), is sketched in Figure 1. The basic idea is to select an arbitrary vertex v, which we call a *pivot* vertex, and find its descendant and predecessor sets. The intersection of the predecessor and descendant sets is $SCC(G, v)$ by Lemma 2.1. After outputting $SCC(G, v)$, the vertices in $G \setminus SCC(G, v)$ are divided into three sets: $Desc(G, v)$, $Pred(G, v)$, and $Rem(G, v)$. By Lemma 2.1, any additional strongly connected component must be entirely contained within one of these three sets, so we can divide the problem and recurse.

The run time of the algorithm depends on the choice of the pivot vertex. For the analysis in the next section, we select the pivot vertex uniformly at random.

As mentioned above, this algorithm is amenable to practical parallelization on two levels. First, the recursive invocations are completely independent and so can execute independently. Second, the searches for predecessors and descendants allow for much more parallelism than does a depth first search.

3.1 Complexity of Algorithm DCSC

The cost of algorithm DCSC consists of time spent in predecessor and descendant searches, plus the time to report strongly connected components. The last term takes linear time in the number of vertices over the course of the algorithm. The remaining terms can be bounded as follows.

The performance of the algorithm depends on the sizes of the recursive calls, and hence on the choice of the pivot vertex. The worst case occurs when the predecessor or descendant set is very large, which leads to $\Theta(mn)$ runtime. The best case occurs when the remainder set is very large, which leads to $\Theta(m)$ runtime. Below we show that by choosing v to be chosen uniformly at random from V, the expected complexity of the algorithm can be bounded as $\Theta(m \lg n + n)$.

Lemma 3.1 *For a directed graph G, there is a numbering π_G of the vertices from 1 to n for which the following is true. All vertices v_j in $\langle Pred(G, v) \setminus SCC(G, v)\rangle$ satisfy $\pi_G(v_j) < \pi_G(v)$; and all vertices v_j in $\langle Desc(G, v) \setminus SCC(G, v)\rangle$ satisfy $\pi_G(v_j) > \pi_G(v)$.*

Proof: If G is acyclic, then a topological sort provides a numbering with this property. If G has cycles, then each strongly connected component can be contracted into a single vertex, and the resulting acyclic graph can be numbered via topological sort. An ordering of the vertices is obtained by replacing each strongly connected component by a list of the vertices it contains. ∎

It is important to note that we do not need to construct an ordering with this property; we just need to know that it exists. In the remainder of this section, we refer to a vertex ordering according to this lemma as a *consistent ordering* of the vertices.

Theorem 3.2 *Algorithm DCSC has expected time complexity $O(m \lg n + n)$.*

Proof: We will show that the expected number of times an edge e is explored in a descendant search is $O(\log n)$. This then holds symmetrically for predecessor searches and proves the theorem.

Let $f(G, e)$ be the average number of times edge e gets explored. Let $d(r)$ be the maximum of $f(G, e)$ over all edges e and all graphs G with r vertices. The proof of the theorem is established by showing that

$$d(r) \leq H_r = 1 + 1/2 + 1/3 + ... + 1/r = \Theta(\lg r). \tag{1}$$

How many times does an edge $e = (u, w)$ get explored in descendant searches? It is last explored when the strongly connected component containing u is dis-

covered. All other visits occur from searches originating on a vertex v that is before u in the consistent ordering. There are at most $\pi_G(u) - 1$ such vertices.

Consider the subgraph G' identified in such a search $\langle Desc(G, v) \setminus SCC(G, v)\rangle$. This subgraph has a consistent ordering in which $\pi_{G'}(u)$ is at most $\pi_G(u) - \pi_G(v)$. This leads to the following recurrence for $d(n)$.

$$\begin{aligned} d(n) &\leq 1 + \frac{1}{n}\sum_{i=1}^{n} d(n-i) \\ &= 1 + \frac{1}{n}\sum_{i=0}^{n-1} d(i) \end{aligned} \tag{2}$$

We claim that $H_r = 1 + 1/2 + ... + 1/n$ satisfies this recursion. It follows that $d(r) \leq H(r) = \Theta(\lg n)$ and the theorem follows.

To show that the quantity $d_r = H_r$ satisfies the recursion multiply (2) by r, and subtract the similar equation with $r - 1$ replacing r:

$$\begin{aligned} r * d(r) &= r + d(0) + d(1) + ... + d(r-2) + d(r-1) \\ - [\ (r-1) * d(r-1) &= (r-1) + d(0) + d(1) + ... + d(r-2)\ ]. \end{aligned}$$

This yields $r * d(r) - (r-1) * d(r-1) = r + d(r-1) - (r-1)$, or equivalently, $d(r) = d(r-1) + 1/r$. ∎

3.2 A Tight Example

The following example shows the bound $O(m \log n)$ is tight. Start with a directed path of length n. Add edges from the last vertex in the path to a set of r new vertices. And then add edges from each of these r vertices to a set of r additional vertices to form a complete bipartite subgraph $K_{r,r}$ with $\sqrt{n} \leq r \leq n^{\alpha}$ for $\alpha < 1$.

On this graph, the algorithm will choose a decreasing sequence of vertices on the path, each exercising the whole bipartite graph, before picking some vertex in the bipartite graph (and breaking it). For simplicity of notation, let $x = 2r$. Let $g(n, x)$ be the expected length of this sequence when the directed path has length n. We show by induction on n that $g(n, 2r) = H_{n+2r} - H_{2r}$, which is $\Theta(\log n)$.

We may write $g(n, x)$ as

$$g(n, x) = \sum_{i=1}^{n} \frac{1}{n+x}[1 + g(i-1, x)]. \tag{3}$$

Note that $g(1,x) = \frac{1}{x+1}$. Use induction to replace $g(i-1,x)$ in (3) with $H_{i-1+x} - H_x$ to get

$$\begin{aligned} g(n,x) &= \frac{n}{n+x} + \frac{1}{n+x}\sum_{i=1}^{n-1}\frac{n-i}{x+i} \\ &= \frac{1}{n+x} + \sum_{i=1}^{n-1}[\frac{1}{n+x} + \frac{n-i}{(x+i)(n+x)}] \\ &= \frac{1}{n+x} + \sum_{i=1}^{n-1}\frac{1}{x+i} \;=\; H_{n+x} - H_x. \end{aligned}$$

4 A Worst Case $O(m \log n)$-time Algorithm

In this section, we modify the algorithm from §3 to achieve a worst case $O(m \log n)$ time complexity. The basic observation here is that we don't need to find both the predecessor and descendant sets, but rather just the intersection of these two sets, which gives $SCC(G,v)$. This intersection can be found by first finding the predecessor (or the descendant) set completely, and then searching for descendants within the predecessor set. Once we have $Pred(G,v)$ (or $Desc(G,v)$) and $SCC(G,v)$, the algorithm can recurse on $\langle Pred(v) \setminus SCC(v)\rangle$ and $\langle G \setminus Pred(v)\rangle$. Choosing the smaller of the predecessor or the descendant sets will be more efficient, since it decomposes the problem with less work. However, we don't know which set is smaller, so instead, we will simultaneously search for both and abandon the second search when the first one finishes. In this modified algorithm, the work for each decomposition is proportional to the smaller of the number of edges in the predecessor set and the number of edges in the descendant set, whereas it is proportional to the sum of the number of edges in these two sets in the original algorithm. The resulting algorithm $WDCSC$ is sketched in Figure 2. The algorithm is somewhat more complex than that of §3, and its parallelization would be more difficult, but it has a deterministic performance guarantee.

Theorem 4.1 *Algorithm WDCSC runs in worst case $O(m \log n)$-time.*

Proof: Let $T(m)$ be the run time of WDCSC on a graph with m edges. Let s to denote the total number of edges in the subgraph induced by the vertex set $SCC(G,v)$. Let p denote the total number of edges in the subgraph induced by the vertex set $Pred(G,v)$, minus s. Symmetrically, let d denote the total number of edges in the subgraph induced by the vertex set $Desc(G,v)$, minus s. By symmetry, we can assume $p \leq d$. Then $T(m)$ can be expressed as

$$T(m) \leq T(p) + T(m-p-s) + c_2(p+s), \tag{4}$$

```
WDCSC(G)
    If G is not empty then
        Select v uniformly at random from V
        Simultaneously search for Pred(G, v) and Desc(G, v)
        If Pred(G, v) finishes first
            Search for Desc(G, v) only in Pred(G, v).
            SCC ← Desc(G, v) ∩ Pred(G, v).
            WDCSC(⟨Pred(G, v) \ SCC⟩)
            WDCSC(⟨V \ Pred(G, v)⟩)
        Else If Desc(G, v) finishes first
            Search for Pred(G, v) only in Desc(G, v).
            SCC ← Desc(G, v) ∩ Pred(G, v).
            WDCSC(⟨Desc(G, v) \ SCC⟩)
            WDCSC(⟨V \ Desc(G, v)⟩)
        Output Pred(G, v) ∩ Desc(G, v)
```

Fig. 2. A worst case $O(m \log n)$ time algorithm to find strongly connected components

where c_2 is a constant ≥ 0, since the amount of work in one iteration is proportional to the minimum of the number of edges in the predecessor set and the number of edges in the descendant set. That is, the number of steps is twice $p + s$ until the algorithm completes one search, and then it is at most s. Note that by hypothesis $p \leq m/2$. We use induction to show $T(m) = c_1 m \log m$ for some constant $c_1 \geq 0$.

$$\begin{aligned} T(m) &\leq T(p) + T(m-p-s) + c_2(p+s) \\ &\leq c_1 p \lg p + c_1(m-p-s)\lg(m-p-s) + c_2(p+s) \\ &\leq c_1 p \lg p + c_1(m-p-s)\lg m + c_2(p+s) \\ &= c_1 m \lg m + [c_1 p \lg p - c_1(p+s)\lg m + c_2(p+s)] \\ &= c_1 m \lg m + [p(c_1 \lg(p/m) + c_2) + s(c_2 - c_1 \lg m)] \end{aligned}$$

To finish the proof, it suffices to show that the term in brackets $p(c_1 \lg(p/m) + c_2) + s(c_2 - c_1 \lg m)$ is not positive. As it turns out, we can choose c_1 such that each of the two terms inside the brackets is negative: Recall that $p < m/2$; thus $\lg(p/m) \leq -1$, and the first term is negative. For $c_1 \geq 2c_2$, the second term will also be negative. ∎

Acknowledgments

We benefited from general discussions about algorithms for parallel strongly connected components with Steve Plimpton, Will McLendon and David Bader. We are also grateful for helpful comments from Cindy Phillips.

References

[1] A. Aggarwal and R. J. Anderson, *A random NC algorithm for depth first search*, Combinatorica, 8 (1988), pp. 1–12.

[2] A. Aggarwal, R. J. Anderson, and M.-Y. Kao, *Parallel depth-first search in general directed graphs*, SIAM J. Comput., 19 (1990), pp. 397–409.

[3] D. A. Bader, *A practical parallel algorithm for cycle detection in partitioned digraphs*, Tech. Rep. Technical Report AHPCC-TR-99-013, Electrical & Computer Engng. Dept., Univ. New Mexico, Albuquerque, NM, 1999.

[4] R. S. Baker and K. R. Koch, *An S_n algorithm for the massively parallel CM-200 computer*, Nuclear Science and Engineering, 128 (1998), pp. 312–320.

[5] P. Chaudhuri, *Finding and updating depth-first spanning trees of acyclic digraphs in parallel*, The Computer Journal, 33 (1990), pp. 247–251.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press and McGraw-Hill, Cambridge, MA, 1990.

[7] M. R. Dorr and C. H. Still, *Concurrent source iteration in the solution of 3-dimensional, multigroup discrete ordinates neutron-transport equations*, Nuclear Science and Engineering, 122 (1996), pp. 287–308.

[8] L. K. Fleischer, B. Hendrickson and A. Pinar, *On Identifying Strongly Connected Components in Parallel*, in Solving Irregularly Structured Problems in Parallel: 7th International Symposium, Irregular '00, Lecture Notes in Computer Science, Vol. 1800, pp. 505–511, Springer-Verlag, 2000.

[9] H. N. Gabow, *Path-based depth-first search for strong and biconnected components*, Information Processing Letters 74 (2000), pp. 107-114.

[10] H. Gazit and G. L. Miller, *An improved parallel algorithm that computes the BFS numbering of a directed graph*, Inform. Process. Lett., 28 (1988), pp. 61–65.

[11] T. Hagerup, *Planar depth-first search in $O(\log n)$ parallel time*, SIAM J. Comput., 19 (1990), pp. 678–704.

[12] M.-Y. Kao, *Linear-processor NC algorithms for planar directed graphs I: Strongly connected components*, SIAM J. Comput., 22 (1993), pp. 431–459.

[13] W. McLendon III, B. Hendrickson, S. Plimpton, and L. Rauchwerger. Finding strongly connected components in distributed graphs. *J. Parallel Distrib. Comput.*, 65:901–910, 2005.

[14] S. Pautz. *An algorithm for parallel S_n sweeps on unstructured meshes*, Nuclear Science Engineering, 140 (2002), pp. 111–136.

[15] S. Plimpton, B. Hendrickson, S. Burns and W. McLendon III, *Parallel Algorithms for Radiation Transport on Unstructured Grids*, In *Proc. SC'00*, Portland, Oregon, 2000.

[16] J. H. Reif, *Depth-first search is inherently sequential*, Inform. Process. Lett., 20 (1985), pp. 229–234.

[17] M. Sharir, *A strong-connectivity algorithm and its applications in data-flow analysis*, Computers and Mathematics with Applications, (1981), pp. 67–72.

[18] R. E. Tarjan, *Depth first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.