STIC-D : Algorithmic Techniques For Efficient Parallel Pagerank Computation on Real-World Graphs

Paritosh Garg
International Institute of Information Technology, Hyderabad
Gachibowli, Hyderabad, India 500 032
paritosh.garg@students.iiit.ac.in

Kishore Kothapalli
International Institute of Information Technology, Hyderabad
Gachibowli, Hyderabad, India 500 032
kkishore@iiit.ac.in

ABSTRACT

Computing metrics on nodes of a graph is an essential step in understanding the properties of the graph. Pagerank is one such metric that is popular and is being used to measure the importance of nodes in not only web graphs but also in social networks, biological networks, road networks, and the like. The core of the computation of pagerank can be seen as an iterative approach that updates the pageranks of nodes until the values converge.

However, as real-world graphs such as road networks and the web have a large size, one needs to design efficient techniques to address the challenges of scale. In addition to parallelism that can be exploited, it is important to also look for specific properties of graphs and their impact on the algorithm.

In this paper, we present four algorithmic techniques that optimize the pagerank computation on real-world graphs. The techniques are presented with the aim of exploiting the nature of the real-world graphs and eliminating redundancies in the pagerank computation. Our techniques also have the advantage that with little extra effort one can quickly identify which of the techniques will be suitable for a given input graph.

We implement our algorithm on an Intel i7 980x CPU running 12 threads using OpenMP Version 3.0. We study our techniques on four classes of real-world graphs: web graphs, social networks, citation and collaboration networks, and road networks. Our implementation achieves an average speedup of 32% compared to a baseline implementation.

CCS Concepts

•**Computing methodologies** → **Parallel algorithms;**

Keywords

Pagerank; Real-World Graphs; Algorithmic Optimizations; Parallel Computing; Strongly Connected Components

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICDCN '16, January 04-07, 2016, Singapore, Singapore

© 2016 ACM. ISBN 978-1-4503-4032-8/16/01…$15.00

DOI: http://dx.doi.org/10.1145/2833312.2833322

1. INTRODUCTION

Pagerank computation introduced by Brin and Page [22] is considered to be a fundamental advancement in the development of search and related technologies. Informally, the pagerank computation assigns a rank to every webpage where the rank indicates the importance of a web page. Pagerank type metrics have since become popular and are used in evaluating the relative importance of nodes in other classes of networks including social networks, citation networks, biological networks, road networks, and the like [17]. It is therefore not surprising that there has been a lot of research interest in the past decade on pagerank computation [11, 14, 19, 20, 24, 28], to mention a few.

The pagerank computation proceeds in iterations and in each iteration the current pagerank of each node is updated by using the pagerank of its incoming neighbors. The computation stops when the pagerank values at all nodes change only by a small value across successive iterations. In each iteration, along each directed edge $e = (u, v)$, a fraction of the pagerank value of u is transferred to the *new* pagerank value of v. This computation offers opportunities for optimization where certain edges of the graph can be identified as *redundant* for one or more iterations. These optimizations can be achieved by a deeper understanding of the structure of popular real-world graph classes such as web graphs, road networks, and citation networks.

There have been a number of works on parallel algorithms to compute the pagerank of nodes in a graph. However, some of the techniques are directed only towards web graphs e.g., [4, 14, 19] and may not translate to other real-world graphs. With pagerank being used for other purposes beyond its original intent [17], it is important to have an efficient algorithm for pagerank computation applicable for several classes of real-world graphs.

In this direction it is to be noted that as real-world graphs increase in size, several scalability issues come to the fore in the process of algorithm design and implementation. Exploiting the parallelism in the computation provides only a limited succor. One needs to deploy efficient techniques to address the scalability issue arising out of the large sizes of the real-world graphs. Such a line of investigation has been pursued recently in finding the strongly connected components by Olukotun et al. [18], graph traversals and shortest paths [6, 7], graph connectivity [12, 26], and the like on a variety of multicore and heterogeneous execution platforms.

In this paper, we present techniques to identify and eliminate the redundancies in the pagerank computation. We name our techniques with the acronym STIC-D for SCC

and Topological Ordering, Identical Nodes, Chain Nodes, and Dead Nodes. The first technique makes use of the observation that sparse directed graphs tend to have a large number of strongly connected components (SCCs). Further, we observe that the block graph [13] of a graph plays a big role in how the pagerank values of nodes converge to their final values. We call this technique as *SCC and Topological Order*. Our second technique, called *Identical Nodes* is based on identifying nodes with an identical incoming neighbourhood. It can be observed that such nodes have an identical pagerank value, hence the computation can be done for only one of the nodes in every class of identical nodes. Our third technique, called *Chain Nodes* is based on the fact that one can shortcut nodes along paths as their contribution to the computation of pagerank can be captured by a succinct formula. We also introduce a fourth technique based on approximation, called *Dead Nodes*, that involves retiring nodes that do not have a big change in their pagerank value across iterations.

While each of the above techniques are simple, there are additional challenges we address to translate these techniques to efficient algorithms. Note that the pagerank computation itself has near-linear work in practice. For most real-world graphs of varying sizes, the number of iterations required for the pagerank computation to converge is usually less than 100. This means that the preprocessing time and any post-processing time required as part of the proposed optimizations have to be very light-weight so as not to offset the gains accrued from the proposed optimizations. Further, it is likely that a specific subset of the above techniques will be appropriate for a given graph. As we aim for techniques that are general-purpose, one also needs to have fast, simple, and effective mechanisms that can identify the applicable techniques for a given graph.

To validate our techniques, we consider four types of real-world graph classes: web graphs, social networks, citation and collaboration networks, and road networks, and show that our techniques can improve the computation of pagerank by a factor of 32% on average.

1.1 Motivation

One of the key motivations of our work is to understand the structural properties of real-world graphs and their impact on algorithms. Specific to the computation of pagerank, we seek properties that can allow us to reduce the number of computations that are invoked for each node of the graph. In this direction, we first note that real world graphs being sparse in nature tend to have several strongly connected components (SCCs). Figure 1 shows evidence for the same for some real-world graphs. As can be noticed from Figure 1, real-world graphs have a large number of SCCs with a small size, and very few SCCs of a large size. This property indicates that a decomposition based on SCCs can be useful in parallel algorithms for pagerank computation. (See also Table 1.)

Now, consider two SCCs H_1 and H_2 of a graph G such that nodes in H_2 have some incoming edges from nodes in H_1. It can be noticed that nodes in H_2 can start computing the pagerank only after nodes in H_1 have converged to their final pagerank value. This property establishes a topological order in which the SCCs of G can be processed during the pagerank computation. Further, once nodes in H_1 converge to their final pagerank values, their contribution to nodes in H_2 does not change across iterations of the pagerank computation for nodes in H_2. These observations form the basis of two of our techniques.

Other structural properties of real-world graphs that we make use of in this work include identifying nodes with identical incoming neighborhoods and also nodes that lie on long directed paths. We show in the subsequent sections that these structural properties offer good reduction in the amount of computation required to arrive at the pagerank values of nodes in a graph.

1.2 Related work

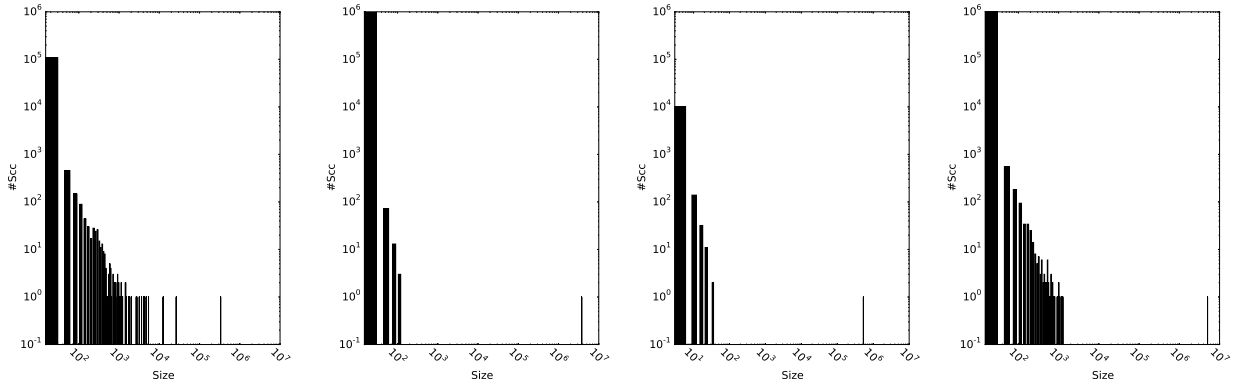
Computing metrics on graphs in parallel has been witnessing a renewed interest in recent years. In the context of pagerank, Broder et al. [10] characterized the structure of web graphs as having a bow-tie nature with one large strongly connected component that also has a large number of incoming and outgoing edges. Arasu et al. [4] use the characterization of Broder et al. [10] and represent the pagerank computation as solving a set of independent matrix equations on a block upper triangular matrix. Kamvar et al. [19] extend the characterization from Broder et al. [10] to partition the input graph into blocks where a block corresponds to a collection of nodes that are also physically related in the context of a web graph by being in the same domain. They proceed to compute a page rank within each block, called the local pagerank, a pagerank for the blocks called as BlockRank, and finally the global pageranks using the local pageranks and the BlockRanks. It is to be noted however that the final ranks computed by Kamvar et al. [19] are *approximate* ranks and can differ from the pagerank values as computed without using the block structure of the graph.

Similar approximation based approaches can be seen in the work of SiteRank by Wu and Aberer [28], the U-Model work of Broder et al. [11], the ServerRank work of Wang et al. [27], and the HostRank/DirRank work of Eiron et al. [15]. Kohlschutter et al. [20] extend the results of Kamvar et al. [19] to obtain exact pageranks. But one limitation of these works [15, 19, 20, 27, 28] is that their algorithms are tailored for web graphs and may not work well for general purpose graphs. As pagerank and related computations gain prominence in other domains, generic techniques are of interest.

Parallel computation of pagerank on other emerging architectures such as GPUs has been studied by Duong et al. [14]. The work of [14] does not introduce any algorithmic optimizations in the computation of pagerank. On the other hand, we believe that our algorithmic techniques will be applicable to other architectures too.

Some of the techniques that we propose are found to be relevant in computing centrality metrics on graphs. For instance, Sariyuce et al. [25] use graph decomposition into biconnected components, removing identical nodes and nodes of degree one, to compute the betweenness-centrality measure of nodes in a graph.

Expressing the pagerank computation as a Markov chain and solving for the steady state transition probabilities of the underlying Markov chain is a mechanism used by many authors. Pandurangan et al. study such an approach in the distributed setting [24].



(a) web-BerkStan (b) soc-LiveJournal1 (c) coPapersDBLP (d) italy_osm

Figure 1: Histogram of SCCs vs their sizes for four graphs. The X-axis show the size of SCCs and the Y-axis shows the number of SCCs. Note that the Y-axis is in logarithmic scale.

1.3 Organization of the Paper

The rest of the paper is organized as follows. In Section 2, we give an algorithmic background for pagerank. In Section 3, we present the algorithmic techniques. Experimental results are presented in Section 4. Section 5 concludes the paper and also identifies directions for further work.

2. PRELIMINARIES

Let $G = (V, E)$ be a directed graph with $n = |V|$ vertices and $m = |E|$ edges. Let $IN(u)$ denote the set of nodes that have an incoming edge to u. Let outdeg(u) denote the number of edges from u to other nodes. The pagerank of a node u, denoted $pr(u)$, is then given as follows.

$$pr(u) = \sum_{v \in IN(u)} \text{contribution}(v \rightarrow u) + \frac{d}{n} \qquad (1)$$

In Equation 1, d is called the damping constant and has its genesis in the manner in which pagerank is usually interpreted. The pagerank values represent a probability distribution where the pagerank of a node denotes the probability of a random walk to visit that node. The damping constant d can be interpreted as the probability that the random walk stays at the same node in the next step. The value of d is taken to be 0.15 usually. The quantity contribution($v \rightarrow u$) is defined as follows.

$$\text{contribution}(v \rightarrow u) = (1 - d) \cdot \frac{pr(v)}{\text{outdeg}(v)} \qquad (2)$$

One can also combine Equations 1,2 to arrive at the following simplified equation.

$$pr(u) = (1 - d) \cdot \sum_{v \in IN(u)} \frac{pr(v)}{\text{outdeg}(v)} + \frac{d}{n} \qquad (3)$$

However this formula is cyclic in nature as two nodes can make contributions to each other if they are part of a directed cycle. One way to resolve this dependency is to apply the formula from Equation 3 iteratively over all the nodes until the pagerank values converge. We say that the pagerank values of nodes have converged when there is very little change in their values across an iteration. This can be measured by a function such as the maximum difference of pagerank values or the total difference of pagerank values across an iteration.

2.1 Baseline Algorithm

Algorithm 1 translates Equation 3 into an iterative pagerank computation. We refer to Algorithm 1 as the baseline algorithm against which we compare our techniques.

Algorithm 1 Base Pagerank(G)

```
1: procedure MAIN(Graph G = (V, E), Array outdeg )
2:     pr=Compute((V, E), outdeg)
3:     return pr
4: end procedure
5:
6: procedure COMPUTE(Graph G,Array outdeg )
7:     error = ∞
8:     for all u ∈ V do
9:         prev(u) = d/n
10:    end for
11:    while error > threshold1 do
12:        for all u ∈ V in parallel do
13:            pr(u) = d/n
14:            for all v ∈ V such that (v, u) ∈ E do
15:                pr(u) = pr(u) + prev(v)/outdeg(v) * (1 − d)
16:            end for
17:        end for
18:        for all u ∈ V do
19:            error = max(error, abs(prev[u] − pr[u]))
20:            prev(u) = pr(u)
21:        end for
22:    end while
23:    return pr
24: end procedure
```

In Algorithm 1, initialize the error as ∞ and the initial pagerank values of all nodes to $\frac{d}{n}$. For reasons of efficiency of parallel computation of pagerank, in Algorithm 1, for each

node u, the new pagerank value of u is computed as the sum of contributions from the incoming neighbors of u. The variable `Threshold1` is a constant that reflects the accuracy of the pagerank needed by an application.

Lines 12–17 iterate over all the nodes in parallel. Each node updates its pagerank based on the contributions of its incoming edges. Lines 18–21 calculate the error function as the L_1 norm of the change in the pagerank values of nodes across one iteration. The variable `error` is used to decide whether to proceed for the next iteration or declare convergence.

3. OUR ALGORITHMIC TECHNIQUES

In this section, we describe our algorithmic techniques that help speedup the pagerank computation by eliminating redundancies. The acronym *STIC* stands for our techniques based on *SCCs and Topological Ordering, Identical Nodes, and Chain Nodes*. In brief, the SCCs and Topological Ordering breaks the pagerank computation into computations on smaller subgraphs that are strongly connected and processed in a particular order. The Identical Nodes optimization refers to eliminating the redundant computation for nodes with an identical incoming neighborhood. The Chain Nodes optimization shortcuts directed paths by a directed edge that connects the end points of the path. More details of these optimizations are presented in the following sections.

3.1 SCC and Topological Ordering

Recall from Equation 3 that in an iterative computation pagerank values of nodes depend cyclically on one another. It is also to be observed that if the pagerank value of all incoming neighbors of a node v converge, then the pagerank value of v will also converge by the next iteration. This observation can be extended to a decomposition of a directed graph into its strongly connected components. For a directed graph $G = (V, E)$, a maximal subset of vertices $U \subseteq V$ such that every pair of nodes in U have at least one directed path between them is said to be a strongly connected component (SCC) (cf [13]). One can also define the block graph H of G where H has one node for each SCC of G. For two nodes $u, v \in H$, there is an edge from u to v if and only if there exist vertices a and b in the corresponding SCCs C_u and C_v of G such that the edge $(a, b) \in E(G)$. The graph H is directed as defined and is also acyclic.

It can be noted that in a topological sort of the nodes of H, if a node $v \in V(H)$ comes after a node $u \in V(H)$, then nodes in the SCC C_v cannot contribute to the pagerank values of nodes in the SCC C_u. Viewed differently, the pagerank computation will benefit if we start processing nodes in C_v after the nodes in all SCCs that have an incoming edge to $v \in H$ converge.

Our SCC and Topological Ordering technique uses the above observations as follows. In a preprocessing step, we find the SCCs of the input graph G and perform a topological sort of the block graph H of G. This results in an ordering of the SCCs of G as $C_1, C_2, \cdots$ so that the pagerank values can be computed for C_1, followed by C_2, and so on.

Let us call edges that have end points in different SCCs as cross edges. Figure 2 illustrates the above ideas. In Figure 2, cross edges are shown as dashed lines. As Figure 2(b) shows, there are three levels in the topological order of the block graph of the graph in Figure 2(a). We can compute the pageranks of nodes in component C_1, followed by components C_2 and C_3 in Level 2 in parallel, and finally component C_4 in Level 3.

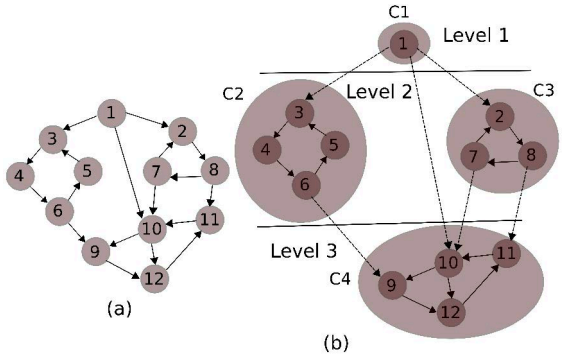


Figure 2: The SCCs of a graph and our processing order.

There is one additional redundant computation that can be identified as we compute pageranks according to a topological order of nodes in the block graph. Consider a node $a \in V(G)$ that appears in SCC C of G such that the node $u \in V(H)$ corresponding to C has a rank of r in a topological sort of H. Node a may have incoming neighbors in SCCs with a rank strictly less than r. In our scheme, the pagerank value of such nodes would have already been computed as we process the nodes in component C. Therefore, in the pagerank computation of node a, the contribution of incoming neighbors of a in components with rank strictly less than r does not change across iterations. Hence, the computation corresponding to such cross edges need be performed only once. In other words, the pagerank of nodes in SCC C can be initialized to the sum of their contributions from incoming neighbors from SCCs with a strictly smaller rank than the rank of C.

3.2 Identical Nodes

In this optimization, we notice that the pagerank of a node is completely dependent on its incoming neighbors. So, it follows that two nodes that have identical incoming neighbors would also have the same pagerank value. Indeed, such is the case at the end of each iteration also. We therefore call two nodes as identical if they have the same set of incoming neighbors. The notion of identical nodes allows one to compute the pagerank of one *representative* node in every class of identical nodes. The pagerank value of the representative node for each class is referred to by all the nodes in that class when they need the pagerank value.

We illustrate the above in Figure 3 where nodes a, b and c have u and v as common neighbors. Notice also any two nodes that are identical will always belong to the same SCC or will be independent SCC's of size one. This helps in the implementation since it will be easy to all identical nodes will be processed along with the component.

3.3 Chain Nodes

This algorithmic optimization concerns nodes along directed paths. On a directed path $P = \langle u = u_0, u_1, u_2, \cdots, u_k = v \rangle$, notice that nodes u_i with $1 \leq i < k$ have exactly one incoming and one outgoing node. Similar to the pagerank computation described in Equation 3, it is now possible to

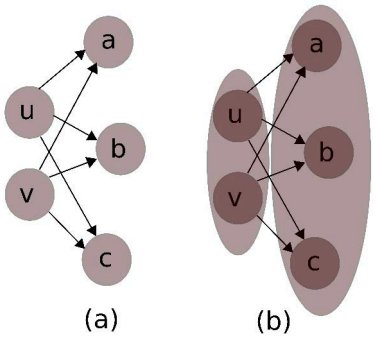


Figure 3: Figure (b) illustrates identical nodes in the graph in Figure (a).

compute the contribution of the pagerank of u to the pagerank of v in on step instead of propagating the contribution of u to v via the edges of P in k iterations. Lemma 1 shows the exact contribution of u to the pagerank of v.

Lemma 1. *Let P=$\{u = u_0, u_1 \cdots, u_k = v\}$ be a directed path. Then the contribution of u on v is given as follows.*

$$\text{contribution}(u \to v) = \frac{pr[u]}{outdeg[u]} * (1-d)^k + \frac{(1-d)*(1-(1-d)^{k-1})}{N}$$

Proof. We prove the lemma by induction. Notice that for $k = 1$, the lemma essentially restates Equation 2. Assuming the induction hypothesis for a directed path of length k, we now show the induction step. By the induction hypothesis, on a directed path $u_0, u_1, \cdots, u_k$, by Equation (1), we can say that:

$$\begin{aligned} pr[u_k] &= \text{contribution}(u \to u_k) + \tfrac{d}{n} \\ pr[u_{k+1}] &= \text{contribution}(u \to u_{k+1}) + \tfrac{d}{n} \end{aligned} \quad (4)$$

Also by Equation (3) and the fact that the in-degree of u_{k+1} and out-degree of u_k is one, we get that $pr[u_{k+1}] = pr[u_k] * (1-d) + \frac{d}{n}$. From above two equations, we have that $\text{contribution}(u \to u_{k+1}) = pr[u_k] * (1-d)$.

By Equation (4) and the previous equation, we can get:

$$\begin{aligned} &\text{contribution}(u \to u_{k+1}) \\ &= \left(\text{contribution}(u \to u_k) + \tfrac{d}{n}\right) \cdot (1-d) \\ &= \left(\frac{pr[u] \cdot (1-d)^k}{outdeg[u]} + \frac{(1-d)\cdot(1-(1-d)^{k-1})+d}{n}\right) \cdot (1-d) \\ &= \frac{pr[u] \cdot (1-d)^{k+1}}{outdeg[u]} + \frac{(1-d)-(1-d)^k+d}{n} \cdot (1-d) \\ &= \frac{pr[u] \cdot (1-d)^{k+1}}{outdeg[u]} + \frac{(1-d)\cdot(1-(1-d)^k)}{n} \end{aligned}$$

□

Figure 4 illustrates this optimization. As can be seen, on a directed path of k nodes, we can remove all but one edge and the end points of the path.

Using Lemma 1, it is possible to perform two related optimizations. On the path P, nodes u_i for $1 \le i < k$ can be removed from the graph and hence the pagerank computation. Nodes u and v can be joined with a directed edge (u, v). During the pagerank computation, we use the Lemma 1 to set the contribution of u to the pagerank of v.

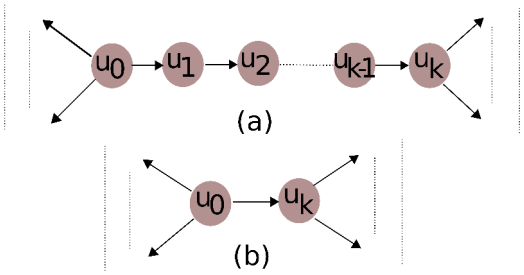


Figure 4: A directed chain of nodes can be compressed to a single edge as illustrated in the figure.

3.4 Marking Nodes Dead

The final optimization that we introduce is to mark certain nodes as Dead Nodes. This is done as follows. While calculating the pagerank values of nodes a graph using the iterative method, it is likely that the pagerank values of some nodes do not change considerably across iterations. We consider the possibility that for such nodes, their contribution to their outgoing neighbors will be minimal. Therefore, we mark such nodes as *Dead Nodes* and do not include them in the subsequent iterations. For reasons of efficiency, for each node v that is not marked as a dead node, a check is performed every t_{num} iterations to identify whether the pagerank value of v changes beyond a pre-specified threshold (`threshold2` in Algorithm 4). If the change in the pagerank value of v is within the threshold, then v is marked as a dead node. Further, the computed pageranks using this optimization differ by less than $\pm$`threshold1`, indicating that the computed pagerank values are very close to their actual values. The value of `threshold1` can be adjusted by the application.

3.5 Putting Together Everything

Using the observations from Section 3.1–3.3, we now visualize the computation of pagerank as a three step process involving preprocessing, the actual computation of pagerank, and post-processing. Post-processing is required for the optimization corresponding to Sections 3.2 and 3.3. Algorithm 2 shows the pseudocode of our approach for pagerank.

Algorithm 2 STIC-D(G)

```
1: procedure MAIN(Graph G = (V, E))
       {{C11, C12 ··}, {C21, C22 ··} ···} = SCC+TOPO(G)
2:     for i = 0 to levels do
3:         for all C_ij ∈ {C_i1, C_i2 ··· } in parallel do
4:             (D_ij, initial, rep)=Preprocess(C_ij, G)
5:             pr=Compute(D_ij, initial, rep)
6:             Post-Process()
7:         end for
8:     end for
9:     return pr
10: end procedure
11:
```

Algorithm 2 is arranged as three steps with Algorithm 3, Algorithm 4 and Algorithm 5 forming the preprocessing, computation and post-processing steps respectively. To sim-

Algorithm 3 STIC-D(G)

1: **procedure** PREPROCESS($Graph\ C_i = (V_i, E_i)$,$Graph\ G = (V, E)$)
2: rep = FindEquiNodes(C_i)
3: **if** $f_1(rep, |V_i|, |E_i|) < thres1$ **then**
4: $rep[u] = u, \forall u \in V$
5: **end if**
6: (D_i) = Compress(C_i)
7: **if** $f_2(|V(D_i)|, |V_i|, |E_i|) < thres2$ **then**
8: $D_i = C_i$
9: **end if**
10: $initial$ = Calinitial(D_i, G)
11: return $(D_i, initial, rep)$
12: **end procedure**

plify our presentation, we consider the graph G as a weighted directed graph where each node has a weight. When using the Chain Node optimization, this weight allows us to set the contribution of the starting node of the path to the end node of the path as derived in Lemma 1.

Algorithm 4 STIC-D(G)

1: **procedure** COMPUTE($Graph\ G = (V, E)$,$Array\ initial$,$Array\ rep$)
2: $error = \infty$, $iterations = 0, adead = true$
3: $prev[u] = \frac{1}{n}, dead[u] = \frac{1}{n}, \forall u \in V$
4: **while** $error > threshold1$ **do**
5: **for all** $u \in V_{nc}$ $s.t$ $rep[u] = u, dead[u] > 0$ in parallel **do**
6: $pr[u] = initial[u]$
7: **for all** $v \in V$ such that $(v, u) \in E$ **do**
8: **if** $v \in V_c$ *and* $rep[v] = v$ **then**
9: $pr[v] = initial[v] + \frac{prev[rep[red[v]]]}{outdeg[red[v]]} * weight[v] + \frac{1-d-weight[v]}{n}$
10: **end if**
11: $pr[u] = pr[u] + \frac{prev[rep[v]]}{outdeg[v]}$
12: **end for**
13: **end for**
14: $iterations = iterations + 1$
15: **if** $iterations\%tnum = 0$ & $adead = true$ **then**
16: Markdead($pr, dead, threshold2, adead$)
17: **end if**
18: **for all** $u \in V_{nc}$ with $rep[u] = u, dead[u] > 0$ **do**
19: $error = max(error, abs(prev[u] - pr[u]))$
20: $prev[u] = pr[u]$
21: **end for**
22: **end while**
23: return pr
24: **end procedure**

In Line 2 of Algorithm 3, the function FindEquinodes returns an array `rep` which stores for each node the representative node of the identical node set of which it is a part of. The function f_1 in Line 3 of Algorithm 3 calculates the percentage of nodes identified as identical nodes. If the percentage of identical nodes is below `thres1`, this optimization is not included.

In Line 6 of Algorithm 3, the function Compress removes all nodes of in-degree and out-degree one and their incident edges and fills the `red` array. This stores the alias node from which the chain nodes calculate the pagerank in the post-processing phase. The decision whether to include this optimization is done based on the function f_2 (Line 7) and the threshold `thres2`.

The function Calinitial (Line 10) sets the initial pagerank values of nodes in an SCC by considering the pagerank of endpoints of incoming cross edges whose pagerank converged already.

In Algorithm 4, V_c and V_{nc} refers to chain and non-chain nodes respectively. Lines 15, 16 of Algorithm 4 check for dead nodes by calling function MarkDead every `tnum` iterations. The function Markdead performs the check by comparing the change in pagerank in the previous `tnum` iterations against `threshold2`. The array dead stores a negative value if the node is dead or stores the pagerank that is at most `tnum` iterations old. The function Markdead also extrapolates the total number of dead nodes from the number of dead nodes seen till now and sets the value of `adead` as false if it less than 8% of n.

3.6 Implementation Details

In this section, we describe the implementation details of our algorithm. Since there could be many components on a level, work must be distributed between threads according to the sizes of the components. For this, if a component size(number of edges) is greater than 10^5, then all the threads work on this component else the component is worked on by only one thread. The work distribution is identical for both the baseline algorithm and our algorithm.

All our preprocessing steps are using standard sequential algorithms. We use Kosaraju's algorithm (cf. [13]) for finding the strongly connected components, and a BFS based algorithm for a topological ordering of the block graph. In FindEquinodes, we only consider nodes with in-degree 1 and 2 as these nodes account for a majority of the nodes that are identical for the purposes of the pagerank computation.

The accuracy of the baseline implementation and that of Algorithm 2 is controlled by the value of the variable `threshold1`. In our implementation, we set the value of `threshold1` to be 10^{-10}. In Algorithm 3, the values of the various thresholds are set as follows. The value of `thres1` and `thres2` are set at 7% and 15% of the size of the graph respectively by following our empirical observations. chain-eps-converted-to.pdfIn Algorithm 4, the variable `tnum` was set to be 22 and `threshold2` is set $\frac{\texttt{threshold1} \cdot \texttt{tnum}}{2 \cdot n}$.

Algorithm 5 STIC-D(G)

1: **procedure** POSTPROCESS($Graph\ G = (V, E)$,$Array\ initial$,$Array\ rep$)
2: **for all** $u \in V_c$ such that $rep[u] = u$ in parallel **do**
3: $pr[u] = initial[u] + \frac{prev[rep[red[u]]]}{outdeg[red[u]]} * weight[u] + \frac{1-d-weight[u]}{n}$
4: **end for**
5: **for all** $u \in V$ such that $rep[u] \neq u$ in parallel **do**
6: $pr[u] = pr[rep[u]]$
7: **end for**
8: **end procedure**

4. EXPERIMENTAL RESULTS

In this section we describe our experimental platform, the dataset we use, and analyze the results of our algorithm.

4.1 Platform

We use an Intel i7 980x processor with 8 GB memory as our experimental platform. The 980x is based on the Intel Westmere micro-architecture and has six cores with each core running at 3.4 GHz. With active SMT (hyper-threading), the i7 980x can support twelve logical threads. The memory hierarchy of the i7 980x has a three level cache system with an L1 cache of size 64 KB per core, an L2 cache of size 256 KB per core, and a shared L3 cache of size 12 MB.

4.2 Dataset

We experiment on four classes of real-world graph datasets namely web graphs, social networks, collaboration networks, and road networks. The graphs are part of standard datasets [1, 2], and important characteristics of the graphs are listed in Table 1. In Table 1, the column SCC's indicates the number of strongly connected components in the graph, and the column Levels indicates the number of levels in the block graph of the corresponding graph.

4.3 Results

We first present the overall speedup results and a study of the time spent across the phases of Algorithm 2. We then proceed to analyze each of the four graph classes.

4.3.1 *Overall Speedup*

We calculate the speedup of Algorithm 2 as the ratio of the time taken by Algorithm 1 to that of Algorithm 2. Both the algorithms are run in a multi-threaded manner with 12 threads on the machine described in Section 4.1. Each experiment is run multiple times and the average speedup is used in reporting.

The results of the overall speedup are shown in Figure 5. In Figure 5, the phrase "baseline" refers to time taken by Algorithm 1. On the X-axis of Figure 5, the graphs are arranged according to their class as listed in Table 1. On an average, we get a speedup of 77% on web graphs, 28% on social graphs, 13% on collaboration networks, and 8% on road networks.

4.3.2 *Profile across Phases*

To understand the differences in the speedup that our algorithm achieves on various graphs, we first start by studying the relative time spent by our algorithm in the preprocessing and the actual computation phases. The post-processing time is not shown as it is noted to be negligible and under 1% of the total time. The reason behind this is that all our techniques are able to simultaneously compute the pagerank of the nodes removed in preprocessing.

The preprocessing time is on an average 25% on web graphs, 11% on social networks, 17% on collaboration networks and 30% on road networks. About half of this spent on finding the SCCs and a topological order among SCCs. To identify identical nodes, the time spent is usually under 2%, except for web graphs where it takes nearly 12%. Identifying identical nodes in web graphs takes more time as there is usually a large percentage of identical nodes. We observe that in road networks, both the SCC and Topological Ordering and the preprocessing time in the Chain nodes optimization step is 14%. This is because these graphs have a lot of levels as shown in Table 1.

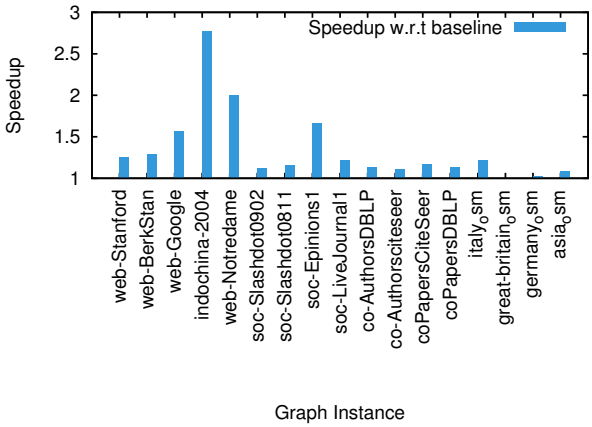


Figure 5: Figure shows the speedup of algorithm on graphs listed in Table 1.

In general, the high processing time of our algorithm can be attributed to the fact that all the preprocessing algorithms are running as sequential algorithms. The effectiveness of our technique can be seen by comparing the time taken by our actual computation with the computation time taken by Algorithm 1. This ratio is on an average 2.4 on web graphs, 1.45 on social networks, 1.3 on collaboration networks, and 1.43 on road networks.

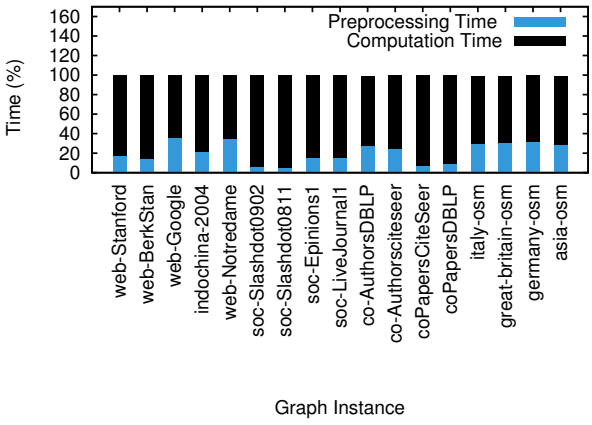


Figure 6: Profile of time taken across various steps of Algorithm 2.

4.3.3 *Analysis of Graph Classes*

In this section, we discuss in detail the effect of our techniques on the graph classes from our dataset.

Web Graphs.

We experimented with five web graphs. We note that these graphs have a large number (15% on average) of cross edges, i.e., edges that have end points in two SCCs. These graphs also have a large number of SCCs indicating that our SCC and Topological Ordering optimization should actually

Graph name	Source	$\|V\|$	$\|E\|$	SCCs	Levels
Web Graphs					
web-Stanford	[21]	281903	2312497	29914	141
web-BerkStan	[21]	685230	7600595	109406	114
indochina-2004	[8, 9]	7414866	194109311	1749052	524
web-Google	[21]	916428	5105039	412479	34
web-Notredame	[3]	325729	1497134	203609	18
Social Networks					
soc-Slashdot0811	[21]	77360	905468	6724	4
soc-Slashdot0902	[21]	82168	948464	10559	3
soc-Epinions1	[23]	75888	508837	42185	10
soc-LiveJournal1	[5, 21]	4847571	68993773	971232	24
Collaboration Networks					
co-AuthorsDBLP	[16]	299067	977676	45242	7
co-Authorsciteseer	[16]	227320	814134	30322	7
coPapersCiteSeer	[16]	434102	16036720	6372	7
coPapersDBLP	[16]	540486	15245729	10244	6
Road Networks					
italy_osm	[2]	6686493	7013978	1470097	2692
great-britain_osm	[2]	7733822	8156517	2444901	725
germany_osm	[2]	11548845	12369181	2466406	534
asia_osm	[2]	11950757	12711603	3511783	10943

Table 1: List of graphs that we use in our experiments.

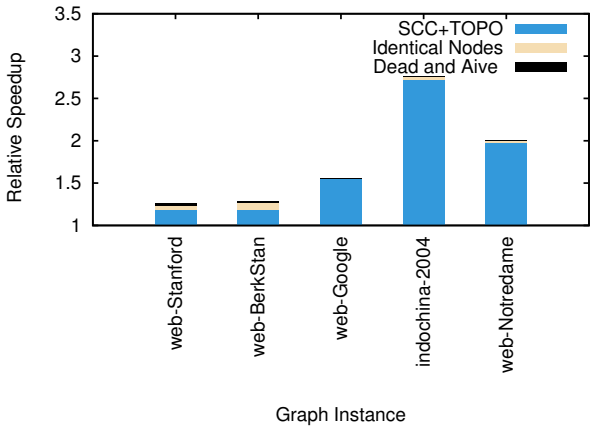


Figure 7: Relative Speedup of optimizations on Web graphs.

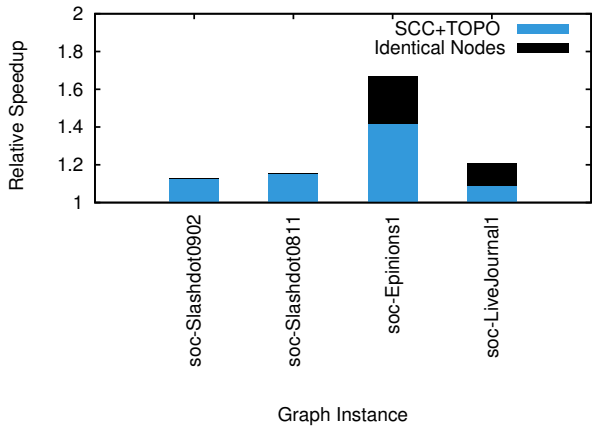


Figure 8: Relative Speedup of optimizations on social networks.

improve the performance of our algorithm by a good factor. Further, we notice that most of the web graphs do not have long directed paths. Therefore, we do not apply the chain optimization step on web graphs. On the other hand, these graphs contain a significant portion (18% on average) of identical nodes. So, we apply this optimization.

The impact of each of the optimizations applied on web graphs is shown in Figure 7. The dead and live optimization works only for the graphs web-Stanford, web-BerkStan and Indochina-2004. On an average, we get a 3% speedup with this optimization on these graphs.

Social Networks.

In the class of social networks, we consider four graphs. These graphs on average have about 5% of cross edges but the SCCs themselves are arranged in fewer levels. These graphs also do not have long directed paths, so this optimization is not considered for social networks. Some of these graphs may not also have a good number of identical nodes, so this optimization can be applied only in specific cases. Figure 8 shows the impact of the optimizations applied on social networks.

Collaboration Networks.

We now consider collaboration networks. These graphs are an example of graphs that do not benefit from our optimizations for several reasons.

These graphs have very few cross edges, very few levels, and do not have long directed paths. These graphs however benefit from keeping nodes dead. The overall speedup on these graphs is quite small as several of our optimizations are not applicable in this class of graphs. Figure 9 shows the

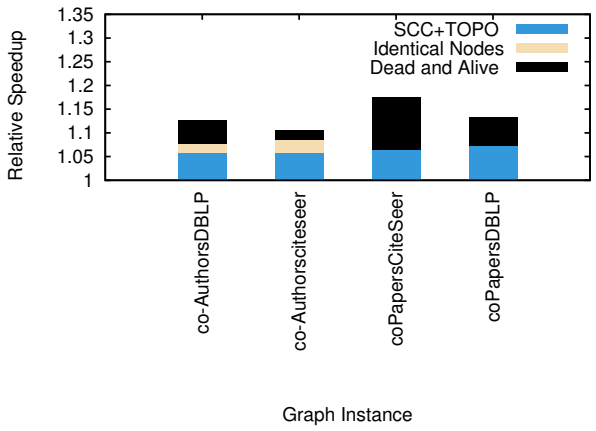


Figure 9: Relative Speedup of optimizations on Collaboration Networks.

relative speedup of techniques on collaboration networks.

Road Networks.

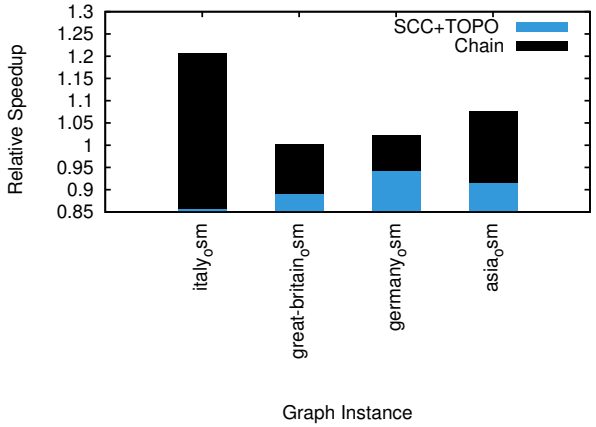


Figure 10: Relative Speedup of optimizations on road networks.

Finally, we move to analyzing road networks. These networks by virtue of their application domain, have a large number of levels and also a large number of SCCs. This means that our SCC and Topological Ordering optimization incurs a huge preprocessing overhead. In fact, just applying this optimization on these graphs actually results in a 10% average slowdown compared to the baseline algorithm.

These graphs also have a small number of identical nodes. So, the advantage gained by identifying the identical nodes will be offset by the preprocessing time required. However, these graphs have a large number of long directed paths. So, the chain node optimization technique works well on these graphs. Figure 10 shows the relative speedup of techniques on collaboration network graphs.

5. CONCLUSIONS AND FUTURE WORK

In this work, we proposed the STIC-D framework to optimize the time taken to compute pagerank in graphs. The techniques in the framework proposed are based on exploiting the structures found in real-world graphs and are useful in reducing the pagerank computation time. The framework can also decide dynamically whether applying a technique is beneficial or not.

In future, we would like to provide for parallel versions of our preprocessing techniques that will improve the performance of our framework.

6. REFERENCES

[1] Stanford network analysis platform dataset. http://www.cise.uïňĆ.edu/research/sparse/matrices/SNAP.

[2] The University of Florida Sparse Matrix Collection. http://www.cise.ufl.edu/research/sparse/matrices/.

[3] Albert, R., Jeong, H., and Barabási, A.-L. Internet: Diameter of the world-wide web. *Nature 401*, 6749 (1999), 130–131.

[4] Arasu, A., Novak, J., Tomkins, A., and Tomlin, J. Pagerank computation and the structure of the web: Experiments and algorithms. In *Proceedings of the Eleventh International World Wide Web Conference, Poster Track* (2002), pp. 107–117.

[5] Backstrom, L., Huttenlocher, D., Kleinberg, J., and Lan, X. Group formation in large social networks: membership, growth, and evolution. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* (2006), ACM, pp. 44–54.

[6] Banerjee, D. S., Kumar, A., Chaitanya, M., Sharma, S., and Kothapalli, K. Work efficient parallel algorithms for large graph exploration on emerging heterogeneous architectures. *JPDC 76* (2015), 81–93.

[7] Banerjee, D. S., Sharma, S., and Kothapalli, K. Work efficient parallel algorithms for large graph exploration. In *HiPC* (2013).

[8] Boldi, P., Rosa, M., Santini, M., and Vigna, S. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th international conference on World Wide Web* (2011), ACM Press.

[9] Boldi, P., and Vigna, S. The WebGraph framework I: Compression techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)* (Manhattan, USA, 2004), ACM Press, pp. 595–601.

[10] Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., and Wiener, J. Graph structure in the web. *Computer Networks 33*, 1-6 (2000), 309–320.

[11] Broder, A. Z., Lempel, R., Maghoul, F., and Pedersen, J. O. Efficient pagerank approximation via graph aggregation. In *Proceedings of the 13th international conference on World Wide Web* (2004), pp. 484–485.

[12] Cong, G., and Bader, D. A. An experimental study of parallel biconnected components algorithms on symmetric multiprocessors (smps). In *Proc. IEEE IPDPS* (2005).

[13] Cormen, Leiserson, Rivest, and Stein. *Introduction To Algorithms*. Prentice Hall.

[14] Duong, N. T., Nguyen, Q. A. P., Nguyen, A. T., and Nguyen, H.-D. Parallel pagerank computation

using gpus. In *Proceedings of the Third Symposium on Information and Communication Technology* (2012), SoICT '12, pp. 223–230.

[15] Eiron, N., McCurley, K. S., and Tomlin, J. A. Ranking the web frontier. In *Proceedings of the 13th international conference on World Wide Web* (2004), pp. 309–318.

[16] Geisberger, R., Sanders, P., and Schultes, D. Better approximation of betweenness centrality. In *ALENEX* (2008), SIAM, pp. 90–100.

[17] Gleich, D. F. PageRank beyond the web. *arXiv cs.SI* (2014), 1407.5107. Accepted for publication in SIAM Review.

[18] Hong, S., Rodia, N. C., and Olukotun, K. On Fast Parallel Detection of Strongly Connected Components (SCC) in Small-World Graphs. In *Proc. of SC'13* (2013).

[19] Kamvar, S., Haveliwala, T., Manning, C., and Golub, G. Exploiting the block structure of the web for computing pagerank. Tech. Rep. 2003-17, Stanford University, 2003.

[20] Kohlschutter, C., Chirita, P.-A., and Nejdl, W. Efficient parallel computation of pagerank. In *Proc. of the 28th European Conference on IR Research* (2006), pp. 241–252.

[21] Leskovec, J., Lang, K., Dasgupta, A., and Mahoney, M. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters.

[22] Page, L., Brin, S., Motwani, R., and Winograd, T. The pagerank citation ranking: bringing order to the web.

[23] Richardson, M., Agrawal, R., and Domingos, P. Trust management for the semantic web. In *The Semantic Web-ISWC 2003*. Springer, 2003, pp. 351–368.

[24] Sarma, A. D., Molla, A. R., Pandurangan, G., and Upfal, E. Fast distributed pagerank computation. In *Proc. of ICDCN* (2013), pp. 11–26.

[25] SarÄśyuce, A. E., .Saüle, E., Kaya, K., and Çatalyürek, U. V. Shattering and compressing networks for betweenness centrality. In *SIAM Data Mining* (2013).

[26] Slota, K. G., and Madduri, K. Simple parallel biconnectivity algorithms for multicore platforms. In *HiPC* (2014), pp. 1–10.

[27] Wang, Y., and DeWitt, D. J. Computing pagerank in a distributed internet search engine system. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases* (2004), pp. 420–431.

[28] Wu, J., and Aberer, K. Using siterank for decentralized computation of web document ranking. In *Proc. of the Third Workshop on Adaptive Hypermedia and Adaptive Web-Based Systems* (2004), pp. 265–274.