APPENDIX

A. DETAILS OF THE S-V PPA

Algorithm 1 presents the original S-V algorithm for the PRAM model, where "par($o \in S$) $\{op(o)\}$" means that the operation $op(o)$ runs in parallel for all objects o in set S. Pointers $D[.]$ are updated by examining edge links in parallel. Initially, for each vertex u, $D[u]$ is set as a neighbor $v < u$ if such a v exists (Line 2). Then, in each round, the pointers are updated by tree hooking (Line 5), star hooking (Line 7) and shortcutting (Line 8).

Each vertex u is also associated with a flag $star[u]$ indicating whether it is in a star. The algorithm terminates when all trees become stars, and each star corresponds to a CC. The value of $star[u]$ is required in two places in the while-loop: Line 3 and Line 7. Algorithm 2 shows the PRAM algorithm for computing $star[u]$ for all $u \in V$ according to the current pointer setting, based on the fact that a vertex u with $D[u] \neq D[D[u]]$ invalidates a tree from being a star (see vertex u in Figure 1(b)).

We now illustrate how to translate the steps of Algorithm 1 into Pregel.

Tree Hooking in Pregel. We compute Lines 4–5 of Algorithm 1 in four supersteps: (1)each vertex u sends request to $w = D[u]$ for $D[w]$; (2)each vertex w responds to u by sending $D[w]$; and meanwhile, each vertex v sends $D[v]$ to all v's neighbors; (3)each vertex u obtains $D[w]$ and $D[v]$ from incoming messages and evaluates the if-condition, and then an arbitrary v that satisfies the condition (if it exists) is chosen and $D[v]$ is sent to w; (4)each vertex w that receives messages sets $D[w] \leftarrow D[v]$ using an arbitrary message $D[v]$.

Star Hooking in Pregel. We compute Lines 6–7 of Algorithm 1 similarly. However, the condition "$D[u] \neq D[v]$" in Line 7 should be changed to "$D[v] < D[u]$".

Computing $star[u]$ in Pregel. We compute Algorithm 2 in five supersteps: (1)each vertex u sets $star[u] \leftarrow true$, and sends request to $w = D[u]$ for $D[w]$; (2)each vertex w responds by sending back $D[w]$; (3)each vertex v checks whether $D[u] = D[w]$; if not, it sets $star[u] \leftarrow false$ and notifies w and $D[w]$; (4)each vertex w that gets notified sets $star[w] \leftarrow false$. To process Line 5, in Superstep (3), we also make each vertex u send request to $w = D[u]$ for $star[w]$; in Superstep (4), we also make w respond by sending $star[w]$ (note that if w get notified, it must set $star[w]$ to be false first). Finally, in Superstep (5), each vertex u gets $star[w]$ and uses it to update $star[u]$.

Analysis. The other steps can be translated to Pregel similarly. To check the condition in Line 3 of Algorithm 1, we use an aggregator that computes the AND of $star[u]$ for all vertices u. The algorithm terminates if its value equals $true$.

The correctness of this PPA can be justified as follows. Since we always require $v_a < v_b$ when setting $D[v_a] \leftarrow v_b$ during hooking, the pointer values monotonically decrease, and thus $D[v] = color(v)$ for any vertex v when the algorithm terminates.

Since each step requires a constant number of supersteps, each round of the S-V algorithm uses a constant number of supersteps (14 in our implementation). Following an analysis similar to [18], the algorithm computes CCs in $O(\log n)$ rounds, and therefore we obtain a PPA for computing CCs. However, the algorithm is not a BPPA since a vertex w may become the parent of more than $d(w)$ vertices and hence receives/sends more than $d(w)$ messages at a superstep, though the overall number of messages is always bounded by $O(n)$ at each superstep.

Algorithm 1 The Shiloach-Vishkin Algorithm

```
1: par($u \in V$) {$D[u] \leftarrow u$}
2: par($(u, v) \in E$) {if($v < u$) $D[u] \leftarrow v$}
3: while($\exists u : star[u] = false$) {
4:     par($(u, v) \in E$)
5:         if($D[D[u]]=D[u]$ and $D[v]<D[u]$){ $D[D[u]] \leftarrow D[v]$}
6:     par($(u, v) \in E$)
7:         if($star[u]$ && $D[u] \neq D[v]$){ $D[D[u]] \leftarrow D[v]$}
8:     par($u \in V$) $D[u] \leftarrow D[D[u]]$
9: }
```

Algorithm 2 Computing $star[u]$

```
1: par($u \in V$) {
2:     $star[u] \leftarrow true$
3:     if($D[u] \neq D[D[u]]$){    $star[u] \leftarrow false$;
4:         $star[D[u]] \leftarrow false$; $star[D[D[u]]] \leftarrow false$}
5:     $star[u] \leftarrow star[D[u]]$
6: }
```

Extension for Computing Spanning Tree. The S-V algorithm can be modified to obtain an $O(\log n)$-superstep PPA that computes a spanning forest of a graph. The main idea is to mark an edge (u, v) as a tree-edge if a hooking operation is performed due to (u, v), which is straightforward for Line 2 of Algorithm 1. For Lines 4 and 6 that perform $D[D[u]] \leftarrow D[v]$, the last superstep is for $w = D[u]$ to pick an arbitrary message $D[v]$ sent by some v, and set $D[w] \leftarrow D[v]$. In this case, w needs to notify both u and v to mark their edge (u, v) as a tree-edge.

B. THEOREM PROOFS

LEMMA 1. *Let $V_{(i,j)} = \{v \in V : (min_f(v), min_b(v)) = (i, j)\}$. Then, (i)any SCC is a subset of some $V_{(i,j)}$, and (ii)$V_{(i,i)}$ is a SCC with color i.*

PROOF. We first prove (i). Given a SCC and a vertex v in the SCC, suppose that v has a label pair (i, j), we show that for any other vertex u in the SCC, u also has the same label pair (i, j). We only prove $min_f(u) = i$, and the proof of $min_b(u) = j$ is symmetric. Let us denote $v_1 \to v_2$ iff v_1 can reach v_2. Since $i \to v \to u$, $min_f(u) > i$ is impossible. Also, since $min_f(u) \to u \to v$, $min_f(u) < i$ is impossible. Therefore, we have $min_f(u) = i$.

We now prove (ii). First, $\forall v \in V_{(i,i)}$, $v \to i$ and $i \to v$, and thus $V_{(i,i)} \subseteq SCC(i)$. Second, if $V_{(i,i)}$ exists, then $i \in V_{(i,i)}$, and by (i), we have $SCC(i) \subseteq V_{(i,i)}$. Therefore, $V_{(i,i)} = SCC(i)$. □

THEOREM 1. *The min-label algorithm runs for at most $\mathcal{L}$ rounds.*

PROOF. We show the correctness of Theorem 1 by proving the following invariant: at the end of Round i, the length of any maximal path in the DAG is at most $\mathcal{L} - i$. We only need to show that in each round, a SCC is determined for any maximal path in the DAG. This is because, after the SCC is marked and the exterior edges are removed, (1) if the SCC is an endpoint of the DAG path (let us denote its length by ℓ), the path length becomes $\ell - 1$; (2) otherwise, the path is broken into two paths with length $< \ell - 1$. In either case, the new paths have length at most $\ell - 1$.

We now show that for an arbitrary maximal DAG path p, at least one SCC is found as $V_{(i,i)}$. First, let us assume i_1 is the smallest unmarked vertex in G, then $V_{(i_1,i_1)}$ is found as a SCC for any maximal path p_1 that contains $SCC(i_1)$. Now consider those maximal paths that does not contain $SCC(i_1)$, and let i_2 be the smallest vertex in the SCCs of these paths. Then $V_{(i_2,i_2)}$ is found as a SCC for any such path p_2 that contains $SCC(i_2)$. The reasoning continues for those maximal paths that do not contain $SCC(i_1)$ and $SCC(i_2)$ until all maximal paths are covered. $\square$

LEMMA 2. *Let* $V_{(S_f,S_b)} = \{v \in V : (Src_f(v), Src_b(v)) = (S_f, S_b)\}$. *Then, (i)any SCC is a subset of some* $V_{(S_f,S_b)}$, *and (ii)*$V_{(S_f,S_b)}$ *is a SCC if* $S_f \cap S_b \neq \emptyset$.

PROOF. We first prove (i). Given a SCC and a vertex v in the SCC, suppose that v has a label pair (S_f, S_b), we show that for any other u in the SCC, u also has the same label pair (S_f, S_b). We only prove $Src_f(u) = S_f$, and the proof of $Src_b(u) = S_b$ is symmetric. (1) For any vertex $s \in S_f$, we have $s \to v \to u$ and thus $s \in Src_f(u)$; therefore, $S_f \subseteq Src_f(u)$. (2) For any vertex $s \in Src_f(u)$, we have $s \to u \to v$ and thus $s \in Src_f(v) = S_f$; therefore, $Src_f(u) \subseteq S_f$. As a result, $Src_f(u) = S_f$.

We now prove (ii), Since $S_f \cap S_b \neq \emptyset$, $\exists u \in S_f \cap S_b$. First, $\forall v \in V_{(S_f,S_b)}$, $v \to u$ and $u \to v$, and thus $V_{(S_f,S_b)} \subseteq SCC(u)$. Second, if $V_{(S_f,S_b)}$ exists, then $u \in V_{(S_f,S_b)}$ and by (i), we have $SCC(u) \subseteq V_{(S_f,S_b)}$. Therefore, $V_{(S_f,S_b)} = SCC(u)$. $\square$

THEOREM 2. *If* $p_i < \frac{2}{k+1}$ *for all i, then* $\theta \leq (1 - 1/c)^k$. *Otherwise,* $\theta = \sum_{i=1}^{c} p_i(1-p_i)^k < 1 - 1/k$.

PROOF. We first prove the case when there exists a SCC SCC_i with $p_i \geq \frac{2}{k+1} > \frac{1}{k}$. Since we sample k vertices in total, in expectation at least one vertex in SCC_i is sampled. As a result, in expectation at least $n_i > n/k$ vertices are marked, or equivalently, $\theta < 1 - 1/k$.

We now prove the case when $p_i < \frac{2}{k+1}$ for all i. Consider the following optimization problem:

$$\begin{aligned} &\text{maximize} \quad && \theta(p_1, \ldots, p_c) = \sum_{i=1}^{c} p_i(1-p_i)^k \\ &\text{subject to} \quad && \sum_{i=1}^{c} p_i = 1, \ (p_i > 0) \end{aligned}$$

Using the method of Lagrange multipliers, we obtain the following Lagrange function:

$$L(p_1, \ldots, p_c, \lambda) = \sum_{i=1}^{c} p_i(1-p_i)^k + \lambda(\sum_{i=1}^{c} p_i - 1)$$

The stationary points can be obtained by solving the following equations:

$$\begin{aligned} \frac{\partial L}{\partial p_i} &= [1 - (k+1)p_i](1-p_i)^{k-1} + \lambda \triangleq 0 \\ \frac{\partial L}{\partial \lambda} &= \sum_{i=1}^{c} p_i - 1 \triangleq 0 \end{aligned}$$

Obviously, $p_i = 1/c$ and $\lambda = [(k+1)/c - 1](1 - 1/c)^{k-1}$ is a solution. To prove that θ is maximized at this point (i.e., $p_i = 1/c$ for all i), we need to show that θ is a concave function in the domain $\{(p_1, \ldots, p_c) \mid p_i > 0 \text{ for all } i \text{ and } \sum_{i=1}^{c} p_i = 1\}$. This is equivalent to showing that the hessian matrix of θ is negative definite.

We now compute the elements of the hessian matrix:

$$\begin{aligned} \frac{\partial^2 \theta}{\partial p_i \partial p_j} &= 0 \qquad \text{(when } i \neq j) \\ \frac{\partial^2 \theta}{\partial p_i^2} &= k[(k+1)p_i - 2](1-p_i)^{k-2} \end{aligned}$$

Round	Task	# of Steps	Comp. Time	Max Size
1	Opt 1	6	0.47 s	2,509
	MinLabel	13 + 13	9.13 s	
	GDecom	3	3.26 s	
2	Opt 1	24	2.66 s	8
	MinLabel	6 + 5	1.37 s	
	GDecom	3	0.31 s	
3	Opt 1	1	0.10 s	0
	MinLabel	6 + 4	0.53 s	
	GDecom	3	0.26 s	
Total			18.09 s	

Figure 19: Min-label performance on Pokec ($\tau = 0$)

Round	Task	# of Steps	Comp. Time	Max Size
1	Opt 1	5	0.38 s	124,164
	MinLabel	16 + 18	7.23 s	
	GDecom	3	3.51 s	
2	Opt 1	4	0.35 s	81
	MinLabel	12 + 9	1.79 s	
	GDecom	3	0.81 s	
3	Opt 1	2	0.27 s	35
	MinLabel	7 + 8	1.74 s	
	GDecom	3	0.31 s	
4	Opt 1	1	0.09 s	0
	MinLabel	5 + 5	1.17 s	
	GDecom	3	0.23 s	
Total			17.88 s	

Figure 20: Min-label performance on Flickr ($\tau = 0$)

Round	Task	# of Steps	Comp. Time	Max Size
1	Opt 1	17	1.77 s	0
	MinLabel	2 + 2	0.28 s	
	GDecom	3	0.29 s	
Total			2.34 s	

Figure 21: Min-label performance on Patent ($\tau = 0$)

Therefore, the hessian matrix is a diagonal matrix $diag(\frac{\partial^2\theta}{\partial p_1^2}, \cdots, \frac{\partial^2\theta}{\partial p_c^2})$. We now show that it is negative definite, which is based on the following property from linear algebra:

LEMMA 3. *Matrix* $M_{N\times N}$ *is negative definite iff for all* $r = 1, \cdots, N$, $(-1)^r \det({}_rM_r) > 0$, *where* ${}_sM_t$ *is the submatrix composed of the first t rows and s columns of* M.

The proof is completed by observing that

$$\begin{aligned} & (-1)^r \det({}_rM_r) \\ = \ & (-1)^r \det(diag(\frac{\partial^2\theta}{\partial p_1^2}, \cdots, \frac{\partial^2\theta}{\partial p_r^2})) \\ = \ & (-1)^r \prod_{i=1}^{r} \frac{\partial^2\theta}{\partial p_i^2} \\ = \ & k^r \cdot [\prod_{i=1}^{r}(2 - (k+1)p_i))] \cdot [\prod_{i=1}^{r}(1-p_i)]^{k-2} \\ > \ & 0 \qquad \text{(since } p_i < 2/(k+1) \text{ and } p_i < 1) \end{aligned}$$

$\square$

C. ADDITIONAL EXPERIMENTAL RESULTS

Here we present the experimental results of our SCC algorithms on the datasets *Pokec*, *Flickr* and *Patent*. We first consider the min-label algorithm. As shown in Figures 19, 20 and 21, the min-label

Round	Task	# of Steps	Comp. Time	Max Size
1	**Opt 1**	6	0.70 s	0
	MultiLabel	15	15.98 s	
	GDecom	3	5.01 s	
MapReduce			24 s	

Figure 22: Multi-label performance on Pokec ($\tau = 50,000$)

Round	Task	# of Steps	Comp. Time	Max Size
1	**Opt 1**	5	0.57 s	125,528
	MultiLabel	21	19.30 s	
	GDecom	3	4.13 s	
2	**Opt 1**	4	0.33 s	125,443
	MultiLabel	9	0.47 s	
	GDecom	3	0.26 s	
3	**Opt 1**	1	0.09 s	125,336
	MultiLabel	10	1.09 s	
	GDecom	3	0.27 s	
MapReduce			74 s	

Figure 23: Multi-label performance on Flickr ($\tau = 50,000$)

Round	Task	# of Steps	Comp. Time	Max Size
1	**Opt 1**	17	2.71 s	0
	MultiLabel	3	0.19 s	
	GDecom	3	0.36 s	
MapReduce			24 s	

Figure 24: Multi-label performance on Patent ($\tau = 50,000$)

algorithm with $\tau = 0$ finds all the SCCs of *Pokec*, *Flickr* and *Patent* in 3, 4 and 1 round(s), respectively. If we run the min-label algorithm with $\tau = 50,000$ for two rounds, and then run a MapReduce job to compute the SCCs of the marked subgraphs, the MapReduce job takes 24 seconds on *Pokec* and 25 seconds on *Flickr*.

We now report the performance of our multi-label algorithm. The performance of the multi-label algorithm on *Pokec*, *Flickr* and *Patent* are shown in Figures 22–24. From Figure 23, we can see that although Round 1 bounds the maximum unmarked subgraph size to a relatively small number, "Max Size" decreases slowly in the later rounds and we cannot afford to run till it gets smaller than $50,000$. However, the subgraphs are small enough to be assigned to different machines for local SCC computation using MapReduce. On the other hand, from Figures 22 and 24, we can see that the multi-label algorithm performs well on *Pokec* and *Patent*, since there is no subgraph with at least $50,000$ vertices after Round 1. Finally, we remark that the final round of MapReduce postprocessing is efficient on all the three graphs.