

# Heaps Are Better than Buckets: Parallel Shortest Paths on Unbalanced Graphs<sup>\*</sup>

Ulrich Meyer

Max-Planck-Institut für Informatik  
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany  
[www.uli-meyer.de](http://www.uli-meyer.de)

**Abstract.** We propose a new parallel algorithm for the single-source shortest-path problem (SSSP). Its heap data structure is particularly advantageous on graphs with a moderate number of high degree nodes. On arbitrary directed graphs with  $n$  nodes,  $m$  edges and independent random edge weights uniformly distributed in the range  $[0, 1]$  and maximum shortest path weight  $\mathcal{L}$  the PRAM version of our algorithm runs in  $\mathcal{O}(\log^2 n \cdot \min\{2^i \cdot \mathcal{L} \cdot \log n + |V_i|\})$  average-case time using  $\mathcal{O}(n \cdot \log n + m)$  operations where  $|V_i|$  is the number of graph vertices with degree at least  $2^i$ . For power-law graph models of the Internet or call graphs this results in the first work-efficient  $o(n^{1/4})$  average-case time algorithm.

## 1 Introduction

The *single-source shortest-path problem* (SSSP) is a fundamental and well-studied combinatorial optimization problem with many practical and theoretical applications. However, the fast and efficient SSSP computation still constitutes a major bottleneck in parallel computing.

Let  $G = (V, E)$  be a directed graph with  $|V| = n$  nodes and  $|E| = m$  edges, let  $s$  be a distinguished vertex of the graph, and  $c$  be a function assigning a non-negative real-valued *weight* to each edge of  $G$ . The objective of the SSSP is to compute, for each vertex  $v$  reachable from  $s$ , the weight of a minimum-weight (“shortest distance”) path from  $s$  to  $v$ , denoted by  $\text{dist}(s, v)$ , abbreviated  $\text{dist}(v)$ ; the weight of a path is the sum of the weights of its edges. We are particularly interested in graphs with unbalanced node degrees, i.e., the maximum node degree  $d$  is by orders of magnitude bigger than the average node degree in  $G$ .

The parallel random access machine (PRAM) [10] is one of the most widely studied abstract models of a parallel computer. A PRAM consists of  $P$  independent processors (processing units, PUs) and a shared memory, which these processors can synchronously access in unit time. We assume the *arbitrary* CRCW (concurrent read concurrent write) PRAM, i.e., in case of conflicting write accesses to the same memory cell, an adversary can choose which access is successful.

---

<sup>\*</sup> Partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

A fast and efficient parallel algorithm minimizes both *time* and *work* (product of time and number of processors). Ideally, the work bound matches the complexity of the best (known) sequential algorithm. Dijkstra’s sequential approach [8] with Fibonacci heaps [11] solves SSSP on arbitrary directed graphs with non-negative edge weights in  $\mathcal{O}(n \log n + m)$  time. It maintains a partition of  $V$  into *settled*, *queued*, and *unreached* nodes, and for each node  $v$  a *tentative distance*  $\text{tent}(v)$ ; In each iteration, the queued node  $v$  with smallest tentative distance is removed from the queue, and all edges  $(v, w)$  are *relaxed*, i.e.,  $\text{tent}(w)$  is set to  $\min\{\text{tent}(w), \text{tent}(v) + c(v, w)\}$ . It is well known that  $\text{tent}(v) = \text{dist}(v)$ , when  $v$  is selected from the queue, hence  $v$  is settled and will never re-enter the queue. Therefore, Dijkstra’s approach is a so called *label-setting* method.

*Label-correcting* variants may remove nodes from the queue for which  $\text{tent}(v) > \text{dist}(v)$  and hence have to *re-insert* those nodes until they are finally settled. Linear average-case time for *directed* SSSP can be achieved with a sequential label-correcting approach [17], too. Label-correcting SSSP algorithms are natural candidates for parallelization: using a number of sequential priority queues, several nodes are removed concurrently in one round. A key problem is the efficient selection of a large “provably good” node set for removal.

**Previous Work on Parallel SSSP.** So far there is no parallel  $\mathcal{O}(n \cdot \log n + m)$  work PRAM SSSP algorithm with sublinear running time for arbitrary digraphs with non-negative edge weights. The  $\mathcal{O}(n \cdot \log n + m)$  work solution by Driscoll et. al. [9] has running time  $\mathcal{O}(n \cdot \log n)$ . An  $\mathcal{O}(n)$  time algorithm requiring  $\mathcal{O}(m \cdot \log n)$  work was presented by Brodal et. al. [5]. Faster algorithms require more work, e.g., the approach by Han et. al. [12] needs  $\mathcal{O}(\log^2 n)$  time and  $\mathcal{O}(n^3 \cdot (\log \log n / \log n)^{1/3})$  work. The algorithm of Klein and Subramanian [14] takes  $\mathcal{O}(\sqrt{n} \cdot \log \mathcal{L} \cdot \log n \cdot \log^* n)$  time and  $\mathcal{O}(\sqrt{n} \cdot m \cdot \log \mathcal{L} \cdot \log n)$  work where  $\mathcal{L}$  is the maximum shortest path weight. Similar results have been obtained by Cohen [6] and Shi and Spencer [20].

Further work-efficient SSSP algorithms exist for random graphs [4] where each of the  $n^2$  possible edges is present with a certain probability. Under the assumption of independent random edge weights uniformly distributed in the interval  $[0, 1]$  the fastest work-efficient label-correcting approach for random graphs [18,19] requires  $\mathcal{O}(\log^2 n)$  time and linear work on average; additionally,  $\mathcal{O}(d \cdot \mathcal{L} \cdot \log n + \log^2 n)$  time and  $\mathcal{O}(n + m + d \cdot \mathcal{L} \cdot \log n)$  work on average is achieved for arbitrary graphs with random edge weights where  $d$  denotes the maximum node degree in the graph and  $\mathcal{L}$  denotes the maximum weight of a shortest path to a node reachable from  $s$ , i.e.,  $\mathcal{L} = \max_{v \in G, \text{dist}(v) < \infty} \text{dist}(v)$ . The algorithms fail for large  $d$ . Crauser et. al. [7] gave general criteria that divide Dijkstra’s label-setting algorithm into a number of phases, such that the operations within a phase can be done in parallel. The efficiency of the criteria was shown for random graphs where sublinear average-case running time can be obtained.

**New Results.** We propose a new parallel label-correcting SSSP approach that is particularly suited for graphs with unbalanced node degrees and an-

alyze its average-case performance on arbitrary directed graphs: Let  $|V_i|$  denote the number of graph vertices with degree at least  $2^i$ . Assuming independent random edge weights uniformly distributed in  $[0, 1]$ , our algorithm runs in  $\mathcal{O}(\log^2 n \cdot \min_i \{2^i \cdot \mathcal{L} \cdot \log n + |V_i|\})$  time on average using  $\mathcal{O}(n \cdot \log n + m)$  operations. This significantly extends the class of inputs for which parallel SSSP can be solved in sublinear average-case time using only  $\mathcal{O}(n \cdot \log n + m)$  work. In particular, for some random power-law graph classes which are widely considered to be appropriate models of the Internet or telephone call graphs [1,16] our algorithm is the first to achieve  $o(n^{1/4})$  average-case time while still remaining work efficient. Furthermore, we sketch how to extend the average-case analysis of the parallel label-setting approach with heaps [7] from random graphs to arbitrary graphs and compare its performance with our new algorithm.

## 2 Preliminaries

The parallel SSSP algorithm of [18], called  $\Delta$ -stepping, and its improved version [19] are label-correcting approaches that work in phases: if  $M$  denotes the smallest tentative distance in the queue data structure  $Q$  at the beginning of a phase, then they remove all nodes  $v$  with tentative distance  $\text{tent}(v) < M + \Delta$  in parallel. The parameter  $\Delta$  is called the *step-width*. Those nodes which are removed with non-final distance values are eventually *re-inserted* into  $Q$ . In order to bound the number of re-insertions and hence the total work,  $\Delta$  must not be chosen too big. On the other hand, taking  $\Delta$  too small can result in many phases, i.e., poor running times:

**Lemma 1** ([18]) *Under the assumption of independent random edge weights uniformly drawn from  $[0, 1]$ , the average-case number of re-insertions for an arbitrary node  $v_0$  in the  $\Delta$ -stepping algorithm is bounded by  $\mathcal{O}(1)$  provided that  $\Delta \leq 1/d$  for maximum node degree  $d$ .*

**Proof:** If a node  $v_0$  was removed from  $Q$  for the first time in phase  $i$ , then a re-insertion of  $v_0$  in phase  $i + j$  can be mapped on a simple path  $\langle v_j, \dots, v_0 \rangle$  such that  $v_k$  was removed in phase  $i + j - k$  and the tentative distance of  $v_k$  was improved by relaxing the edge  $(v_{k+1}, v_k)$ . In any case, the total weight of the path  $\langle v_j, \dots, v_0 \rangle$  must be smaller than the step-width. If  $d$  denotes the maximum degree in the graph then there are at most  $d^l$  simple paths of  $l$  edges into  $v_0$ . For  $l$  independent random edge weights uniformly distributed in  $[0, 1]$ , the probability that their sum is at most  $\Delta \leq 1$  is bound by  $\Delta^l/l!$ . Hence, the average-case number of re-insertions for  $v_0$  can be bounded by  $\sum_1^\infty (d \cdot \Delta)^l/l! = \mathcal{O}(1)$  for  $\Delta \leq 1/d$ . □

Parallel node-removal is realized by using a random mapping  $\pi$  of node indices to a number of  $P$  sequential priority queues  $Q_j$ . Load-balancing for node removals is guaranteed with high probability (whp)<sup>1</sup> by the random distribution

<sup>1</sup> with high probability (whp) means that the probability for some event is at least  $1 - n^{-\beta}$  for any constant  $\beta > 0$ .

provided that sufficiently many nodes are removed in parallel. Load-balancing for relaxations is done as follows: all relaxations of a phase are first grouped according to their target nodes (semi-sorting with integer keys) and for each node only the relaxation resulting in the smallest tentative distance is forwarded to its priority queue in charge [19].

The sequential priority structure for processor  $PU_j$  can be implemented by linear arrays  $B_j$  of buckets such a queued node  $v$  is kept in  $B_{\pi(v)}[i]$  for tentative distances in the range  $[i \cdot \Delta, (i + 1) \cdot \Delta)$ . Let  $k$  denote the biggest bucket index such that  $B_j[0], \dots, B_j[k - 1]$  are empty for all  $j$ . Then a phase removes all nodes from  $B_0[k], \dots, B_{P-1}[k]$ . After  $\mathcal{O}(\log n)$  phases the smallest tentative distance in the queue has increased by at least  $\Delta$  whp. This is a simple consequence of the observation that in a graph with maximum degree  $d$  and random edge weights there are no simple paths of  $\Omega(\log n)$  edges and total weight at most  $1/d$  whp [18]. Hence, each bucket is expanded during at most  $\mathcal{O}(\log n)$  subsequent phases until it finally remains empty whp. Finding the next bucket index for deletion is done in a sequential fashion, i.e., testing  $k$  buckets takes  $k$  phases. Thus, for maximum shortest path weight  $\mathcal{L}$ , at least  $\mathcal{L} \cdot d$  phases and at most  $\mathcal{O}(\mathcal{L} \cdot d \cdot \log n)$  phases are required whp.

Based on the  $\Delta$ -stepping algorithm sketched above, a sequential SSSP algorithm with average-case running time  $\mathcal{O}(n + m)$  was developed [17]. Starting with buckets of width  $\Delta = 1$ , it builds a bucket hierarchy in the following way: before the algorithm removes all nodes from the current bucket  $B_{\text{cur}}$  of width  $\Delta_{\text{cur}}$  it checks the maximum node degree  $d^*$  in  $B_{\text{cur}}$ . If  $\Delta_{\text{cur}} > 1/d^*$  then it splits  $B_{\text{cur}}$  into smaller buckets of size  $2^{-\lceil \log_2 d^* \rceil}$  each and continues with the leftmost non-empty bucket among those just generated. Thus, on the one hand, the average-case number of re-insertions and re-relaxations can be bounded by  $\mathcal{O}(n + m)$  since nodes with high degree are exclusively expanded from buckets with sufficiently small widths; on the other hand, the number of buckets is bounded by  $\mathcal{O}(n + m)$ , as well, independent of the maximum shortest path weight  $\mathcal{L}$ . However, even though the nodes within a bucket can still be expanded in parallel,  $\Omega(\mathcal{L} + \max_{v \in V} \text{degree}(v))$  phases are required just to visit all buckets.

### 3 Degree Heaps

In the following we will use the idea of an adaptive step-width for an alternative data structure that avoids scanning lots of small distance intervals at little extra cost. Our new label-correcting algorithm (called Parallel Degree SSSP) uses a number of sequential priority queues and a simple method to compute an appropriate current step width. Opposite to the algorithm of [17] changing the step-width does not require restructuring of the priority queues themselves.

For a graph with  $n$  nodes we define a *sequential Degree Heap*  $D$  to be a collection of  $h = \lceil \log_2 n \rceil$  relaxed heaps  $D_1, \dots, D_h$  such that  $D_i$  is in charge of tentative distances for nodes having in-degree in  $[2^{i-1} + 1, 2^i]$ . A relaxed heap allows insertions and decrease\_key operations in worst-case constant time, deletions of the minimum in worst-case logarithmic time [9]. There is no heap for

nodes with degree zero as they are never reached in the SSSP algorithm anyway. Let  $M_i$  be the smallest tentative distance in  $D_i$  ( $M_i = \infty$  for empty  $D_i$ ) and let  $M = \min_i M_i$ . Then we compute

$$\Delta_{\max} := \min_i \{\max\{2^{-i-1}, M_i - M\}\}. \tag{1}$$

Subsequently, for each  $D_i$ , the SSSP algorithm removes all nodes  $v \in D_i$  satisfying  $\text{tent}(v) < M + \Delta_{\max}$ .

**Property 1** *Up to a multiplicative factor of two,  $\Delta_{\max}$  is maximal such that for any queued node  $v$  with  $\text{tent}(v) < M + \Delta_{\max}$ , we have  $\text{degree}(v) \leq 1/(2 \cdot \Delta_{\max})$ .*

**Proof:** Consider the index  $i^*$  that minimizes (1), i.e.,  $\Delta_{\max} = \max\{2^{-i^*-1}, M_{i^*} - M\} < \infty$ . If  $M_{i^*} - M > 2^{-i^*-1}$ , then  $\Delta_{\max}$  was just chosen small enough in order not to remove any node from  $D_{i^*}$  for the current phase. Taking another step-width  $\Delta \geq 2 \cdot \Delta_{\max} > 2^{-i^*}$  there is a node  $v \in D_{i^*}$  having degree at least  $2^{i^*-1} + 1$  and  $\text{tent}(v) < M + \Delta$ , hence  $\text{degree}(v) > 1/(2 \cdot \Delta)$ . Similarly, if  $M_{i^*} - M \leq 2^{-i^*-1}$ , then there is already a node  $v \in D_{i^*}$  having degree at least  $2^{i^*-1} + 1$  and  $\text{tent}(v) < M + 2^{-i^*-1}$ . Enlarging  $\Delta_{\max}$  by a factor of two leads to the same kind of contradiction.  $\square$

**Corollary 1** *Nodes of degree  $d$  are removed using step-width at most  $2^{-\lceil \log_2 d \rceil - 1}$ .*

Parallel Degree Heaps are obtained by having a sequential Degree Heap  $D^j$  for each processor  $PU_j$ . Again, a random mapping  $\pi$  is used to distribute the nodes over the sequential Degree Heaps, i.e., node  $v$  with in-degree  $d$ ,  $2^{i-1} + 1 \leq d \leq 2^i$ , is kept in the  $i$ -th heap of  $PU_{\pi(v)}$ ,  $D_i^{\pi(v)}$ . The step-width computation is adapted in the obvious way by setting  $M_i = \min_j M_i^j$  where  $M_i^j$  denotes the smallest tentative distance in the  $i$ -th heap of  $PU_j$ . The minimum of a relaxed heap can be determined in  $\mathcal{O}(\log n)$  time, hence each PU can compute its local minima in  $\mathcal{O}(\log^2 n)$  time. After that, the global minima  $M_i$  and  $M$  can be computed by standard pipelined tree-reductions in  $\mathcal{O}(\log n)$  time. Finally, the new largest possible step-width is computed as the minimum of  $\lceil \log_2 n \rceil$  expressions.

**Lemma 2** *Each step-width computation for Parallel Degree Heaps with  $P \leq n$  processors can be performed in  $\mathcal{O}(\log^2 n)$  time and  $\mathcal{O}(P \cdot \log^2 n)$  work.*

## 4 Average-Case Analysis of Parallel Degree Heaps SSSP

We exploit the correlation between the maximum degree among the nodes currently deleted in a phase and the applied step-width to show:

**Lemma 3** *Using Parallel Degree Heaps SSSP for graphs with random edge weights uniformly drawn from  $[0, 1]$ , each node is re-inserted at most  $\mathcal{O}(1)$  times on the average.*

**Proof:** Consider an arbitrary node  $v_0$  with in-degree  $d_0$ . Let  $M$  be the minimum of all tentative distances for queued nodes in the Degree Heaps when  $v_0$  is removed for the first time. By then,  $\text{tent}(v_0) \leq M + 2^{-\lceil \log_2 d_0 \rceil - 1}$  (Corollary 1). As in the proof of Lemma 1, re-insertions of  $v_0$  can be mapped to appropriate paths  $P = \langle v_j, \dots, v_0 \rangle$  of total weight at most  $2^{-\lceil \log_2 d_0 \rceil - 1}$  such that the nodes  $v_j, \dots, v_0$  are subsequently removed from the Degree Heaps and the relaxation of their edges lead to an improvement for  $\text{tent}(v_0)$ .

Let  $d_i$  be the in-degree of node  $v_i$ . We can confine our analysis to *degree-weight balanced* paths  $\langle v_j, \dots, v_0 \rangle$  where  $c(v_{i+1}, v_i) \leq 2^{-\lceil \log_2 d_i \rceil - 1}$ . In order to see this, let  $k$  be the smallest index such that  $c(v_{k+1}, v_k) > 2^{-\lceil \log_2 d_k \rceil - 1}$  for the path  $P$  above. The value of  $\text{tent}(v_k)$  is already correct up to at most  $2^{-\lceil \log_2 d_k \rceil - 1}$  when  $v_k$  is removed for the first time. Either,  $\text{dist}(v_{k+1}) < \text{dist}(v_k) - 2^{-\lceil \log_2 d_k \rceil - 1}$ , then  $v_{k+1}$  must have been settled before  $v_k$  was removed, and the edge  $(v_{k+1}, v_k)$  will never be re-relaxed again to re-insert  $v_k$  (and  $v_0$  in the end). Or  $\text{dist}(v_{k+1}) \geq \text{dist}(v_k) - 2^{-\lceil \log_2 d_k \rceil - 1}$ , but then after the first removal of  $v_k$  no improvement of  $\text{tent}(v_k)$  can be obtained via a relaxation of  $(v_{k+1}, v_k)$ . Hence, re-insertions for  $v_k$  that could trigger re-insertions of  $v_0$  require edges into  $v_k$  of weight less than  $2^{-\lceil \log_2 d_k \rceil - 1}$ .

Therefore, in order to bound the number of re-insertions for node  $v_0$  it is sufficient to consider all sub-paths  $\langle v_l, \dots, v_0 \rangle$  into  $v_0$  that are degree-weight balanced and have total weight at most  $2^{-\lceil \log_2 d_0 \rceil - 1}$ . The expected number of edges with weight at most  $\min\{2^{-\lceil \log_2 d_i \rceil - 1}, 2^{-\lceil \log_2 d_0 \rceil - 1}\}$  into node  $v_i$  is bounded by  $1/2$ . Therefore, using elementary results of branching processes [2,13] the expected number of relevant degree-weight balanced paths having  $l$  edges can be bounded by  $(1/2)^l$ . Thus, the expected number of re-insertions for  $v_0$  can be bounded by  $\mathcal{O}(\sum_{l \geq 1} (1/2)^l) = \mathcal{O}(1)$ . □

**Theorem 1** *For graphs with random edge weights uniformly drawn from  $[0, 1]$ , Parallel Degree Heap SSSP needs  $r = \mathcal{O}(\min_i \{2^i \cdot \mathcal{L} \cdot \log n + |V_i|\})$  phases on the average where  $\mathcal{L}$  denotes the maximum shortest path weight and  $|V_i|$  is the number of graph vertices with in-degree at least  $2^i$ . On a CRCW PRAM it can be implemented in  $\mathcal{O}(r \cdot \log^2 n)$  time and  $\mathcal{O}(n \cdot \log n + m)$  work on the average.*

**Proof:** We fix some arbitrary integer  $x \geq 0$  and consider *B-phases* having step-width bigger or equal  $\Delta_x := 2^{-x-1}$  and *S-phases* having step-width smaller than  $\Delta_x$ . By Lemma 3, each node is re-inserted  $\mathcal{O}(1)$  times on average, and by Property 1, *S-phases* do only occur when a node of in-degree at least  $2^x + 1$  is deleted. Hence, the expected number of *S-phases* is bounded by  $\mathcal{O}(|V_x|)$ . After  $\mathcal{O}(\log n)$  *B-phases* the smallest tentative distance among all queued nodes,  $M$ , has increased by  $\Delta_x$  whp ([18], intermediate *S-phases* can only increase  $M$ ). Therefore, the total number of *B-phases* is bounded by  $\mathcal{O}(\frac{\mathcal{L} \cdot \log n}{\Delta_x}) = \mathcal{O}(2^x \cdot \mathcal{L} \cdot \log n)$  whp. Altogether we need  $\mathcal{O}(2^x \cdot \mathcal{L} \cdot \log n + |V_x|)$  phases on average. Since we are free to choose  $x$ , the average-case bound for all phases can be improved to  $\mathcal{O}(\min_i \{2^i \cdot \mathcal{L} \cdot \log n + |V_i|\})$ .

For the PRAM algorithm we use the load-balancing approach with semi-sorting from [19] where buckets are replaced by Degree Heaps: if  $r$  phases are

needed on the average then up to  $\Theta(\frac{n}{r \cdot \log n})$  sequential Degree Heaps can be used in a load-balanced way. Hence, determining a new step-width takes  $\mathcal{O}(\log^2 n)$  time and  $\mathcal{O}(\frac{n \cdot \log n}{r})$  work for each phase by Lemma 2. Since node deletions in a Sequential Degree Heap require  $\mathcal{O}(\log n)$  time (as compared to  $\mathcal{O}(1)$  time for buckets), the algorithm needs altogether  $\mathcal{O}(r \cdot \log^2 n)$  time and  $\mathcal{O}(n \cdot \log n + m)$  work on the average.  $\square$

## 5 Parallel Label-Setting on Arbitrary Graphs

In this section we sketch how to improve the analysis of a previous parallel label-setting algorithm [7] that is also based on heaps. It applies an adaptive node-removal criterion, too: let  $T = \min\{\text{tent}(u) + c(u, z) : u \text{ is queued and } (u, z) \in E\}$ . The OUT-approach removes all queued nodes  $v$  with  $\text{tent}(v) \leq T$  in one phase since these nodes cannot lead to further distance reductions.

The analysis for *random* graphs and random edge weights given in [7] is based on two observations: (1) random graphs are expanders, i.e., the priority queue is well-filled during most phases. (2) For  $q$  queued nodes, each of which having expected degree  $d$ , there is a constant probability that the  $\sqrt{q/d}$  queued nodes with smallest tentative distance can be concurrently removed in one phase: Let  $v_1, v_2, \dots, v_q$  be the queued nodes in order of increasing tentative distances, and let  $T'$  be the value of  $T$  in the previous phase. The distance labels  $\text{tent}(v_i)$  are random variables in  $[T', T' + 1]$ . Their values are independent and their distributions are biased towards smaller values since they constitute the minimum of potentially many incoming path weights. The value of  $\text{tent}(v_r)$  is therefore less than  $r/q$  with constant probability for arbitrary  $r$ ,  $1 \leq r \leq q$ . The number of edges out of  $v_1, \dots, v_r$  is  $\mathcal{O}(d \cdot r)$  with constant probability. Opposite to the tentative distances of queued nodes, the edge weights are not biased towards smaller values. Therefore, the shortest of these edges has length about  $\frac{1}{rd}$ . We remove  $v_1, \dots, v_r$  from the queue if  $\text{tent}(v_r)$  is smaller than the length of the shortest edge out of  $v_1, \dots, v_r$ . This is the case (with constant probability) if  $r/q \leq \frac{1}{rd}$  or  $r \leq \sqrt{q/d}$ .

An improved analysis for arbitrary graphs with random edge weights can be based on “active edges” where all edges of a queued node are active. By the same argument as before, for  $k$  active edges, there is a constant probability that  $\Omega(\sqrt{k})$  of them are concurrently removed in a single phase. Additionally, we use that  $T - T' \geq \sqrt{1/k}$  with constant probability. Note that the smallest tentative distance kept in the queue,  $M$ , increases by the same amount as  $T$ . Therefore, looking at a number of consecutive phases, either  $M$  will significantly increase or many edges will be removed. Altogether, for maximum shortest path weight  $\mathcal{L}$  there are at most  $\mathcal{O}(\sqrt{\mathcal{L} \cdot m})$  phases that remove at least  $\Omega(\sqrt{m/\mathcal{L}})$  edges each. For any of the other phases, there is a constant probability that  $T$  is increased by  $\Omega(\sqrt{\mathcal{L}/m})$ . However, there cannot be more than  $\mathcal{O}(\sqrt{\mathcal{L} \cdot m})$  such phases because  $M$  will not exceed  $\mathcal{L}$ . Thus, on the average there are  $\mathcal{O}(\sqrt{\mathcal{L} \cdot m})$  phases in total.

Using the OUT-approach together with the load-balancing scheme of [19], SSSP can be solved in  $\mathcal{O}(\sqrt{\mathcal{L} \cdot m} \cdot \log^3 n)$  average-case time and  $\mathcal{O}(n \cdot \log n + m)$  work. Hence, label-correcting SSSP with Parallel Degree Heaps is faster than the OUT-approach unless the graph contains  $\Omega(\sqrt{\mathcal{L} \cdot m})$  nodes of in-degree  $\Omega(\sqrt{m/\mathcal{L}})$ . In that case, the average-case time of both algorithms is bounded by  $\mathcal{O}(\sqrt{\mathcal{L} \cdot m} \cdot \log^3 n)$ . In fact, if  $\mathcal{L} \cdot m$  is very big, both algorithms require at most  $n$  phases since each phase settles at least one node whereas the  $\Delta$ -stepping may take  $\Omega(n^2)$  phases. On the other hand, disregarding logarithmic factors, the OUT-approach (and therefore also the Parallel Degree Heap SSSP) is faster than the  $\Delta$ -stepping whenever  $d^2 \cdot \mathcal{L} > m$ .

## 6 Performance on Power Law Graphs

Many massive graphs such as the WWW graph and telephone call graphs share universal characteristics which can be described by the so-called “power law”: the number of nodes,  $y$ , of a given in-degree  $x$  is proportional to  $x^{-\beta}$  for some constant  $\beta > 0$ . For most massive graphs,  $\beta > 2$ . Independently, Kumar et. al. [15] and Babarasi et. al. [3] reported  $\beta \approx 2.1$  for the in-degrees of the WWW graph, and the same value was estimated for telephone call graphs [1].

Graph models where the targets of edges are drawn *randomly* and the *expected* number of nodes for a certain node degree follow a power-law are widely considered to be appropriate models of real massive graphs like the WWW. For  $\beta > 2$ , the diameter of such graphs was shown to be  $\mathcal{O}(\log n)$  whp [16]. Furthermore, for  $\beta \geq 2$ , the expected maximum node degree is  $\Theta(n^{1/\beta})$  and  $m = \mathcal{O}(n)$ . Hence, assuming independent random edge weights uniformly distributed in  $[0, 1]$ , SSSP on WWW-like graphs with  $\Delta$ -stepping requires about  $\mathcal{O}(n^{1/2.1} \cdot \log^2 n) = \mathcal{O}(n^{0.48})$  time and linear work on average. The average-case execution time of the OUT-approach is about the same,  $\mathcal{O}(n^{0.5} \cdot \text{polylog}(n))$ .

The expected number of nodes having in-degree at least  $d^*$  is bounded by  $\mathcal{O}(n \cdot \sum_{x \geq d^*} x^{-\beta})$  which for constant  $\beta \geq 2$  and arbitrary  $d^* \geq \beta$  can be approximated by  $\mathcal{O}(n \cdot \int_{d^*}^{\infty} x^{-\beta} dx) = \mathcal{O}(n \cdot d^{*\beta-1})$ . Thus, taking  $\beta = 2.1$  and  $d^* = n^{10/41}$  one expects  $\mathcal{O}(n \cdot n^{-3.1 \cdot 10/41}) = \mathcal{O}(n^{10/41})$  nodes of in-degree at least  $d^*$ . Therefore, with Parallel Degree Heaps, the average-case time for SSSP on WWW-like graphs drops to  $\mathcal{O}(\log^2 n \cdot (n^{10/41} \cdot \log n \cdot \log n + n^{10/41})) = o(n^{1/4})$  using  $\mathcal{O}(n \cdot \log n + m)$  work.

## 7 Conclusions

We have given a new parallel label-correcting SSSP approach together with a data structure that efficiently supports the required operations. The provable average-case performance depends on the node degree distribution. Improved running times while retaining work-efficiency could be shown for practically important inputs with unbalanced node degrees like the WWW graph. However, for the future it would be desirable to solve the SSSP on Web-like graphs in polylogarithmic time and  $\mathcal{O}(n \cdot \log n + m)$  work. Furthermore, any work-efficient



algorithm with sublinear running time that is independent of the diameter would be of great interest.

## References

1. W. Aiello, F. Chung, and L. Lu. A random graph model for massive graphs. In *Proc. 32nd Annual ACM Symposium on Theory of Computing*, pages 171–180. ACM, 2000. [345](#), [350](#)
2. K. B. Athreya and P. Ney. *Branching Processes*. Springer, 1972. [348](#)
3. A. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999. [350](#)
4. B. Bollobás. *Random Graphs*. Academic Press, 1985. [344](#)
5. G. S. Brodal, J. L. Träff, and C. D. Zaroliagis. A parallel priority queue with constant time operations. *Journal of Parallel and Distributed Computing*, 49(1):4–21, 1998. [344](#)
6. E. Cohen. Using selective path-doubling for parallel shortest-path computations. *Journal of Algorithms*, 22(1):30–56, January 1997. [344](#)
7. A. Crauser, K. Mehlhorn, U. Meyer, and P. Sanders. A parallelization of Dijkstra’s shortest path algorithm. In *23rd Symp. on Mathematical Foundations of Computer Science*, volume 1450 of *LNCS*, pages 722–731. Springer, 1998. [344](#), [345](#), [349](#)
8. E. W. Dijkstra. A note on two problems in connexion with graphs. *Num. Math.*, 1:269–271, 1959. [344](#)
9. J. R. Driscoll, H. N. Gabow, R. Shrairman, and R. E. Tarjan. Relaxed heaps: An alternative to fibonacci heaps with applications to parallel computation. *Communications of the ACM*, 31, 1988. [344](#), [346](#)
10. S. Fortune and J. Wyllie. Parallelism in random access memories. In *Proc. 10th Symp. on the Theory of Computing*, pages 114–118. ACM, 1978. [343](#)
11. M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, 1987. [344](#)
12. Y. Han, V. Pan, and J. Reif. Efficient parallel algorithms for computing all pair shortest paths in directed graphs. *Algorithmica*, 17(4):399–415, 1997. [344](#)
13. T. Harris. *The Theory of Branching Processes*. Springer, 1963. [348](#)
14. P. Klein and S. Subramanian. A randomized parallel algorithm for single-source shortest paths. *Journal of Algorithms*, 25(2):205–220, November 1997. [344](#)
15. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the web for emerging cyber-communities. In *Proc. 8th International World-Wide Web Conference*, 1999. [350](#)
16. L. Lu. The diameter of random massive graphs. In *Proc. 12th Annual Symposium on Discrete Algorithms*, pages 912–921. ACM–SIAM, 2001. [345](#), [350](#)
17. U. Meyer. Single-source shortest-paths on arbitrary directed graphs in linear average-case time. In *Proc. 12th Annual Symposium on Discrete Algorithms*, pages 797–806. ACM–SIAM, 2001. [344](#), [346](#)
18. U. Meyer and P. Sanders.  $\Delta$ -stepping: A parallel shortest path algorithm. In *6th European Symposium on Algorithms (ESA)*, volume 1461 of *LNCS*, pages 393–404. Springer, 1998. [344](#), [345](#), [346](#), [348](#)
19. U. Meyer and P. Sanders. Parallel shortest path for arbitrary graphs. In *Proc. Euro-Par 2000 Parallel Processing*, volume 1900 of *LNCS*, pages 461–470. Springer, 2000. [344](#), [345](#), [346](#), [348](#), [350](#)
20. H. Shi and T. H. Spencer. Time–work tradeoffs of the single-source shortest paths problem. *Journal of Algorithms*, 30(1):19–32, 1999. [344](#)