In-Core Computation of Geometric Centralities with HyperBall:
A Hundred Billion Nodes and Beyond

Paolo Boldi   Sebastiano Vigna

Dipartimento di Informatica, Università degli Studi di Milano, Italy

Abstract—Given a social network, which of its nodes are more central? This question was asked many times in sociology, psychology and computer science, and a whole plethora of *centrality measures* (a.k.a. *centrality indices*, or *rankings*) were proposed to account for the importance of the nodes of a network. In this paper, we approach the problem of computing *geometric centralities*, such as *closeness* [1] and *harmonic centrality* [2], on very large graphs; traditionally this task requires an all-pairs shortest-path computation in the exact case, or a number of breadth-first traversals for approximated computations, but these techniques yield very weak statistical guarantees on highly disconnected graphs. We rather assume that the graph is accessed in a *semi-streaming* fashion, that is, that adjacency lists are scanned almost sequentially, and that a very small amount of memory (in the order of a dozen bytes) per node is available in core memory. We leverage the newly discovered algorithms based on HyperLogLog counters [3], making it possible to approximate a number of geometric centralities at a very high speed and with high accuracy. While the application of similar algorithms for the approximation of closeness was attempted in the MapReduce [4] framework [5], our exploitation of HyperLogLog counters reduces exponentially the memory footprint, paving the way for in-core processing of networks with a hundred billion nodes using "just" 2 TiB of RAM. Moreover, the computations we describe are inherently parallelizable, and scale linearly with the number of available cores.

I. INTRODUCTION

In the last years, there has been an ever-increasing research activity in the study of real-world complex networks. These networks, typically generated directly or indirectly by human activity and interaction, appear in a large variety of contexts and often exhibit a surprisingly similar structure.

One of the most important notions that researchers have been trying to capture in such networks is "node centrality": ideally, every node (often representing an individual) has some degree of influence or importance within the social domain under consideration, and one expects such importance to be reflected in the structure of the social network. Centrality in fact has a long history in the context of social sciences: starting from the late 1940s [1] the problem of singling out influential individuals in a social group has been a holy grail that sociologists have been trying to capture for many decades.

Among the types of centrality that have been considered in the literature (see [6] for a good survey), many have to do with the distance to other nodes. If, for instance, the sum of distances to all other nodes is large, the node is *peripheral*, which is the starting point to define Bavelas's *closeness centrality* as the reciprocal of peripherality (i.e., the reciprocal of the distances to all other nodes).

Interestingly, many of these indices can be recast in terms of suitable calculations using the sizes of the balls of varying radius around a node. In a previous work [3] we presented HyperANF, a tool that can compute the distance distribution of very large graphs. HyperANF has been used, for instance, to show that Facebook has just four "degrees of separation" [7]. The goal of this paper is to extends the HyperANF approach to compute a number of centrality indices based on distances.

Beside large-scale experiment using the full ClueWeb09 graph (almost five billion nodes), we provide an empirical evaluation of the accuracy of our method through a comparison with the exact centrality values on a snapshot of Wikipedia (on larger graphs the exact computation would be infeasible). We also provide comparisons with a MapReduce-based [4] approach [5], showing that a careful combination of Hyper-LogLog counters, compression and succinct data structure can provide a speedup of two orders of magnitude, and in fact, comparing costs, more scalability. We also show how to extend our techniques to a class of weighted graphs with a tiny loss in space.

The Java software implementing the algorithms described in this paper is distributed as free software within the Web-Graph framework.[1] Moreover, all dataset we use are publicly available.

Using our Java tool we are able, for the first time, to approximate distance-based centrality indices on graphs with billions of nodes using a standard workstation.

II. NOTATION

In this paper, we use the following notation: $G = (V, E)$ is a directed graph with $n = |V|$ nodes and $m = |E|$ arcs; we write $x \to y$ as a shortcut for $(x, y) \in E$. The length of the shortest path from x to y is denoted by $d(x, y)$ and called the *distance* between x and y; we let $d(x, y) = \infty$ if there is no

---

[1]http://webgraph.di.unimi.it/

directed path from x to y. The nodes *reachable* from x are the nodes y such that $d(x,y) < \infty$. The nodes *coreachable* from x are the nodes y such that $d(y,x) < \infty$. We let G^T be the *transpose* of G (i.e., the graph obtained by reverting all arc directions in G). The ball of radius r around x is

$$\mathscr{B}_G(x,r) = \{\, y \mid d(x,y) \leq t \,\}.$$

III. Geometric centralities

We call *geometric* those centrality measures[2] whose basic assumption is that importance depends on some function of the distances. These are actually some of the oldest measures defined in the literature.

A. Closeness centrality

Closeness was introduced by Bavelas in the late forties [8]; the closeness of x is defined by

$$\frac{1}{\sum_y d(y,x)}. \qquad (1)$$

The intuition behind closeness is that nodes with a large sum of distances are *peripheral*. By reciprocating the sum, nodes with a smaller denominator obtain a larger centrality. We remark that for the above definition to make sense, the graph needs be strongly connected. Lacking that condition, some of the denominators will be ∞, resulting in a rank of zero for all nodes which cannot coreach the whole graph.

In fact, it was not probably in Bavelas's intentions to apply the measure to non-connected graphs, but nonetheless the measure is sometimes "patched" by simply not including pairs with infinite distance, that is,

$$\frac{1}{\sum_{d(y,x)<\infty} d(y,x)};$$

for the sake of completeness, one further assumes that nodes with an empty coreachable set have centrality 0 by definition. These apparently innocuous adjustments, however, introduce a strong bias toward nodes with a small coreachable set.

B. Lin's centrality

Nan Lin [9] tried to patch the definition of closeness for graphs with infinite distances by weighting closeness using the square of the number of coreachable nodes; his definition for the centrality of a node x with a nonempty coreachable set is

$$\frac{\left| \{ y \mid d(y,x) < \infty \} \right|^2}{\sum_{d(y,x)<\infty} d(y,x)}.$$

---

[2]Most centrality measures proposed in the literature were actually described only for undirected, connected graphs. Since the study of web graphs and online social networks has posed the problem of extending centrality concepts to networks that are directed, and possibly not strongly connected, in the rest of this paper we consider measures depending on the *incoming* arcs of a node, so distances will be taken from all nodes to a fixed node. If necessary, these measures can be called "negative", as opposed to the "positive" versions obtained by taking the transpose of the graph.

Nodes with an empty coreachable set have centrality 1 by definition.

The rationale behind this definitions is the following: first, we consider closeness not the inverse of a sum of distances, but rather the inverse of the *average* distance, which entails a first multiplication by the number of coreachable nodes. This change normalizes closeness across the graph. Now, however, we want nodes with a larger coreachable set to be more important, given that the average distance is the same, so we multiply again by the number of coreachable nodes.

Lin's index was somewhat surprisingly ignored in the following literature. Nonetheless, it seems to provide a reasonable solution for the problems caused by the definition of closeness.

C. Harmonic centrality

As we noticed, the main problem with closeness lies in the presence of pairs of unreachable nodes. In [2], we have proposed to replace the reciprocal of the sum of distances in the definition of closeness with the sum of reciprocals of distances. Conceptually, this corresponds to replacing the reciprocal of a denormalized average of distances with the the reciprocal of a denormalized *harmonic* mean of distances, analogously to what Marchiori and Latora proposed to do with the notion of average distance [10]. The harmonic mean has the useful property of handling ∞ cleanly (assuming, of course, that $\infty^{-1} = 0$).

We thus obtain the *harmonic centrality* of x:

$$\sum_{y \neq x} \frac{1}{d(y,x)} = \sum_{d(y,x)<\infty, y \neq x} \frac{1}{d(y,x)}. \qquad (2)$$

The difference with (1) might seem minor, but actually it is a radical change. Harmonic centrality is strongly correlated to closeness centrality in simple networks, but naturally also accounts for nodes y that cannot reach x. Thus, it can be fruitfully applied to graphs that are not strongly connected.

IV. HyperBall

In this section, we present *HyperBall*, a general framework for computations that depend on the number of nodes at distance at most t or exactly t from a node. HyperBall uses the same dynamic programming scheme of algorithms that approximate neighborhood functions, such as ANF [11] or HyperANF [3], but instead of aggregating at each step the information about all nodes into a single output value (the neighbourhood function at t) HyperBall makes it possible to perform a different set of operations (for example, depending on the centrality to be computed). We have tried to make the treatment self-contained, albeit a few details will be only sketched here, when they can be deduced from the description of HyperANF [3].

A. HyperLogLog counters

HyperLogLog counters, as described in [12] (which is based on [13]), are used to count approximately the number of distinct elements in a stream. For the purposes of the present paper, we need to recall briefly their behaviour. Essentially, these probabilistic counters are a sort of *approximate set representation* to which, however, we are only allowed to pose questions about the (approximate) size of the set.

Let $\mathscr{D}$ be a fixed domain and $h : \mathscr{D} \to 2^{\infty}$ be a fixed hash function mapping each element of $\mathscr{D}$ into an infinite binary sequence. For a given $x \in 2^{\infty}$, let $h_t(x)$ denote the sequence made by the leftmost t bits of $h(x)$, and $h^t(x)$ be the sequence of remaining bits of x; h_t is identified with its corresponding integer value in the range $\{0, 1, \ldots, 2^t - 1\}$. Moreover, given a binary sequence w, we let $\rho^+(w)$ be the number of leading zeroes in w plus one (e.g., $\rho^+(00101) = 3$). Unless otherwise specified, all logarithms are in base 2.

Algorithm 1 The Hyperloglog counter as described in [12]: it allows one to count (approximately) the number of distinct elements in a stream. α_p is a constant whose value depends on p and is provided in [12]. Some technical details have been simplified.

```
0    h : 𝒟 → 2^∞, a hash function from the domain of items
1    M[−] the counter, an array of p = 2^b registers
2       (indexed from 0) and set to −∞
3
4    function add(M: counter, x: item)
5    begin
6       i ← h_b(x);
7       M[i] ← max{ M[i], ρ⁺(h^b(x)) }
8    end; // function add
9
10   function size(M: counter)
11   begin
12       Z ← ( Σ_{j=0}^{p−1} 2^{−M[j]} )^{−1};
13       return E = α_p p² Z
14   end; // function size
15
16   foreach item x seen in the stream begin
17       add(M,x)
18   end;
19   print size(M)
```

The value E printed by Algorithm 1 is [12][Theorem 1] an asymptotically almost[3] unbiased estimator for the number n of distinct elements in the stream; for $n \to \infty$, the *relative standard deviation* (that is, the ratio between the standard deviation of E and n) is at most $\beta_p / \sqrt{p} \le 1.06 / \sqrt{p}$, where

---

[3]For the purposes of this paper, in the following we will consider in practice the estimator as it if was unbiased, as suggested in [12].

β_p is a suitable constant. Moreover, even if the size of the registers (and of the hash function) used by the algorithm is unbounded, one can limit it to $\log \log(n/p) + \omega(n)$ bits obtaining almost certainly the same output ($\omega(n)$ is a function going to infinity arbitrarily slowly); overall, the algorithm requires $(1 + o(1)) \cdot p \log \log(n/p)$ bits of space (this is the reason why these counters are called HyperLogLog). Here and in the rest of the paper we tacitly assume that $p \ge 16$ and that registers are made of $\lceil \log \log n \rceil$ bits.

B. Estimating balls

The basic idea used by algorithms such as ANF [11] and HyperANF [3] is that that $\mathscr{B}_G(x, r)$, the ball of radius r around node x, satisfies

$$\mathscr{B}_G(x, 0) = \{ x \}$$
$$\mathscr{B}_G(x, r+1) = \bigcup_{x \to y} \mathscr{B}_G(y, r) \cup \{ x \}.$$

We can thus compute $\mathscr{B}_G(x, r)$ iteratively using sequential scans of the graph (i.e., scans in which we go in turn through the successor list of each node). One obvious drawback of this solution is that during the scan we need to access randomly the sets $\mathscr{B}_G(x, r-1)$ (the sets $\mathscr{B}_G(x, r)$ can be just saved on disk on an *update file* and reloaded later). For this to be possible, we need to store the (approximated) balls in a data structure that can be fit in the core memory: here is where probabilistic counters come into play; to be able to use them, though, we need to endow counters with a primitive for the union. Union can be implemented provided that the counter associated with the stream of data AB can be computed from the counters associated with A and B; in the case of HyperLogLog counters, this is easily seen to correspond to maximising the two counters, register by register.

Algorithm 2, named *HyperBall*, describes our strategy to compute centralities. We keep track of one HyperLogLog counter for each node; at the t-th iteration of the main loop, the counter $c[v]$ is in the same state as if it would have been fed with $\mathscr{B}_G(v, t)$, and so its expected value is $|\mathscr{B}_G(v, t)|$. During the execution of the loop, when we have finished examining node v the counter a is in the same state as if it would have been fed with $\mathscr{B}_G(v, t+1)$, and so its value will be $|\mathscr{B}_G(v, t+1)|$ in expectation.

This means, in particular, that it is possible to compute an approximation of

$$|\{ y \mid d(x, y) = t \}|$$

(the number of nodes at distance t from x) by evaluating

$$|\mathscr{B}_G(v, t+1)| - |\mathscr{B}_G(v, t)|.$$

The computation would be exact if the algorithm had actually kept track of the set $\mathscr{B}_G(x, t)$ for each node, something that is obviously not possible; using probabilistic counters makes this feasible, at the cost of tolerating some approximation in the computation of cardinalities.

The idea of using differences between ball sizes to estimate the number of nodes at distance t appeared also in [14], where it was used with a different kind of counter (Martin–Flajolet) to estimate the 90% percentile of the distribution of distances from each node. An analogous technique, always exploiting Martin–Flajolet counters, was adopted in [5] to approximate closeness. In both cases the implementations were geared towards MapReduce [4]. A more sophisticated approach, which can be implemented using breadth-first visits or dynamic programming, uses *all-distances sketches* [15]: it provides better error bounds, but it requires also significantly more memory.

Algorithm 2 HyperBall in pseudocode. The algorithm uses, for each node $v \in n$, an initially empty HyperLogLog counter $c[v]$. The function union$(-, -)$ maximises two counters register by register. At line 19, one has the estimate of $|\mathscr{B}_G(v, t)|$ from $c[v]$ and the estimate of $|\mathscr{B}_G(v, t + 1)|$ from a.

```
0    c[−], an array of n HyperLogLog counters
1
2    function union(M: counter, N: counter)
3       foreach i < p begin
4          M[i] ← max(M[i], N[i])
5       end
6    end; // function union
7
8    foreach v ∈ n begin
9       add(c[v], v)
10   end;
11   t ← 0;
12   do begin
13      foreach v ∈ n begin
14         a ← c[v];
15         foreach v → w begin
16            a ← union(c[w], a)
17         end;
18         write ⟨v, a⟩ to disk
19         do something with a and c[v]
20      end;
21      Read the pairs ⟨v, a⟩ and update the array c[−]
22      t ← t + 1
23   until no counter changes its value.
```

HyperBall is run until all counters stabilise (e.g., the last iteration must leave all counters unchanged). As shown in [3], any alternative termination condition may lead to arbitrarily large mistakes on pathological graphs.

V. ESTIMATING CENTRALITIES

It should be clear that exactly three ingredients for each node x are necessary to compute closeness, harmonic, and Lin's centrality:

- the sum of the distances to x;
- the sum of the reciprocals of the distances to x;
- the size of the coreachable set of x.

The last quantity is simply the value of each counter $c[v]$ in HyperBall at the end of the computation on G^T. The other quantities can be easily computed in a cumulative fashion nothing that

$$\sum_y d(y, x) = \sum_{t > 0} t |\{ y \mid d(y, x) = t \}|$$
$$= \sum_{t > 0} t \big(|\mathscr{B}_{G^T}(x, t)| - |\mathscr{B}_{G^T}(x, t - 1)|\big),$$

and

$$\sum_{y \neq x} \frac{1}{d(y, x)} = \sum_{t > 0} \frac{1}{t} |\{ y \mid d(y, x) = t \}|$$
$$= \sum_{t > 0} \frac{1}{t} \big(|\mathscr{B}_{G^T}(x, t)| - |\mathscr{B}_{G^T}(x, t - 1)|\big).$$

We can thus obtain estimators for the first two ingredients by storing a single floating point value per node, and cumulating the values for each node during the execution of HyperBall. Note that we have to run the algorithm on the *transpose* of G, since we need to estimate the distances *to* x, rather than *from* x.

If we accept the minimum possible precision (16 registers per HyperLogLog counter), the core memory necessary for running HyperBall is just 16 bytes per node (assuming $n \leq 2^{64}$), plus four booleans per node to keep track of modifications, and ancillary data structures that are orders of magnitude smaller. A machine with 2 TiB of core memory could thus compute centralities on networks with more than a hundred billion nodes, prompting the title of this paper.

Note that even if we use a small number of registers per HyperLogLog counter, by executing HyperBall multiple times we can increase the confidence in the computed value for each estimator, leading to increasingly better approximations.

As in the case of the average distance [3], the theoretical bounds are quite ugly, but actually the derived values we compute are very precise, as shown by the concentration of the values associated several runs. Multiple runs in this case are very useful, as they make it possible to compute the empirical standard deviation.

A. Representing and scanning the graph

In the previous section we have estimated the core memory usage of HyperBall without taking the graph size into account. However, representing and accessing the graph is a nontrivial problem, in particular during the last phases of the computation, where we can keep track of the few nodes that are modifying their counter, and propagate new values only when necessary.

Here we exploit two techniques: *compression*, to represent the graph as a bit stream in a small amount of disk space, so

that we are able to access it from disk efficiently using memory mapping; and *succint data structures*, to access quickly the bitstream in a random fashion.

In particular, for compression we use the WebGraph framework [16], which is a set of state-of-the-art algorithms and codes to compress web and social graphs. WebGraph represents a graph as a bitstream, with a further 64-bit pointer for each node if random access is necessary. To store the pointers in memory, we use a succinct encoding based on a broadword implementation [17] of the Elias-Fano representation of monotone sequences [18]. This way, the cost of a pointer is logarithmic in the average length per node of the bitstream, and in real-world graphs this means about one byte of core memory per node, which is an order of magnitude less than the memory used by HyperBall.

B. Error bounds

The estimate $\hat{\mathscr{B}}_G(x,t)$ for $|\mathscr{B}_G(x,t)|$ obtained by HyperBall follow the bounds given in Section IV-A. Nonetheless, as soon as we consider the differences $\hat{\mathscr{B}}_G(x, t+1) - \hat{\mathscr{B}}_G(x,t)$, the bounds on the error become quite ugly. A similar problem occurs when estimating the distance distribution and its statistics: by taking the difference between points of the cumulative distribution, the bound on the relative standard deviation is lost [3].

Note that in part this is an intrinsic problem: HyperBall essentially runs in quasi-linear expected time $O(pm\log n)$ [15], and due to known bounds on the approximation the diameter [19] it is unlikely that it can provide in all cases a good approximation of the differences (which would imply a good approximation of the eccentricity of each node, and in the end a good approximation of the diameter).

Nonetheless, for a number of reasons the estimates of the differences on real-world graphs turn out to be very good. First of all, for very small numbers the HyperLogLog counters compute a different estimator (not shown in Algorithm 1) that is much more accurate. Second, on social and web graphs (and in general, for small-world graphs) the function $|\mathscr{B}_G(x,t)|$ grows very quickly for small values of t, so the magnitude of the difference is not far from the magnitude of the ball size, which makes the relative error on the ball size small with respect to the difference. Third, once most of the nodes in the reachable set are contained in $\mathscr{B}_G(x,t)$, the error of the HyperLogLog counter tends to stabilise, so the bound on the relative standard deviation "transfers" to the differences.

We thus expect (and observe) that the estimation of the size of the nodes at distance t to be quite accurate, in spite of the difficulty of proving a theoretical error bound.

From a practical viewpoint, the simplest way of controlling the error is generating multiple samples, and computing the empirical standard deviation. This is, for example, the way in which the results for the "degrees of separation" in [7] were reported. By generating several samples, we can restrict the confidence interval for the computed values.

In Section VIII we report experiments on a relatively small graph on which centralities could be computed exactly to show that the precision obtained on the final values is very close to the theoretical prediction for a single counter.

VI. Computing with weights on the nodes

It is very natural, in a number of contexts, to have *weights* on the nodes that represent their importance. Centrality measures should then be redefined taking into account weights in the obvious way: the sum of distances should become

$$\sum_y w(y)d(y,x),$$

the sum of inverse distances should become

$$\sum_y \frac{w(y)}{d(y,x)},$$

and the size of the coreachable set should become

$$\sum_{d(y,x)<\infty} w(y).$$

There is no direct way to incorporate weights in the dynamic programming algorithm, but weights can be easily simulated if they are integers. Suppose that the weighting function is $w : V \to \{1, \dots, W\}$, and assume that each node $x \in V$ is associated with a set $\mathscr{R}(x) = \{x_1, \dots, x_{w(x)}\}$ of replicas of the node (with the proviso that distinct nodes have disjoint replicas).

Then the *weighted ball of radius r around x* can be defined recursively as:

$$\mathscr{W}_G(x,0) = \mathscr{R}(x)$$
$$\mathscr{W}_G(x,r+1) = \mathscr{R}(x) \cup \bigcup_{x\to y} \mathscr{W}_G(y,r).$$

It is easy to see that

$$|\mathscr{W}_G(x,r+1)| - |\mathscr{W}_G(x,r)| = \sum_{y:d(x,y)=r} w(y).$$

Attention must be paid, of course, to the sizing of the counters in this case. Instead of $\log\log n$ bits, counters with

$$\log\log \sum_x w(x) \le \log\log(Wn) = \log(\log n + \log W)$$

bits will have to be used. We note, however, that since the increase factor $\sum_x w(x)/n$ passes through two logarithms, it is unlikely that more than 6 or at most 7 bits will be ever necessary.

VII. Computing with discount functions

If we look at harmonic centrality from a more elementary perspective, we can see that when measuring the centrality of a node we start by considering its (in)degree, that is, how many neighbours it has at distance one. Unsatisfied by this raw measure, we continue and take into consideration nodes at distance two. However, their number is not as important as the degree, so before adding it to the degree we *discount*

its importance it by $1/2$. The process continues with nodes at distance three, discounted by $1/3$ until all coreachable nodes have been considered.

The essence of this process is that we are counting nodes at larger and larger distances from the target, discounting their number based on their distance. One can generalize this idea to a *family* of centrality measures. The idea, similar to the definition of *discounted cumulative gain* in information retrieval [20], is that with each coreachable node we gain some importance. However, the importance given by the node is *discounted* by a quantity depending on the distance that, in the case of harmonic centrality, is the reciprocal $1/d$. Another reasonable choice is a *logarithmic* discount $1/\log(d+1)$, which attenuates even more slowly the importance of far nodes, or a *quadratic* discount $1/d^2$. More generally, the centrality of x based on a non-increasing discount function $f : \mathbf{N} \to \mathbf{R}$ is

$$\sum_{d(y,x)<\infty, y \neq x} f(d(y,x)).$$

It can be approximated by HyperBall nothing that

$$\sum_{d(y,x)<\infty, y \neq x} f(d(y,x)) = \sum_{t>0} f(t) |\{ y \mid d(y,x) = t \}|$$
$$= \sum_{t>0} f(t) \big(|\mathscr{B}_{G^T}(x,t)| - |\mathscr{B}_{G^T}(x,t-1)| \big).$$

We are proposing relatively mild discount functions, in contract with the *exponential* decay used, for example, in Katz's index [21]. This is perfectly reasonable, since Katz's index is based on *paths*, which are usually infinite. Discount-based centralities are necessarily given by finite summations, so there is no need for a rapid decay. Actually, by choosing a constant discount function we would estimate the importance of each node just by the number of nodes it can coreach (i.e., in the undirected case, by the size of its connected component).

Combining this observation and that of Section VI, we conclude that HyperBall can compute a class of centralities that could be called *discounted-gain centralities*:[4]

$$\sum_{d(y,x)<\infty, y \neq x} w(y) f(d(y,x)).$$

VIII. EXPERIMENTS

We decided to perform three kinds of experiments:

- A small-scale experiment on the same graphs for which explicit timings are reported in [5], to compare the absolute speed of a MapReduced-based approach using the Hadoop open-source implementation and of an in-core approach. Note that the graphs involved are extremely unrealistic (e.g., they have all diameter 2 and are orders of magnitude denser than typical web or social graphs). This experiment was run using $p = 64$ registers per HyperLogLog counter, corresponding to a relative standard deviation of 13.18%, which is slightly better

---

[4]These are called *spatially decaying* in [22].

TABLE I. COMPARATIVE TIMINGS PER ITERATION BETWEEN THE HADOOP IMPLEMENTATION DESCRIBED IN [5] RUNNING ON 50 MACHINES AND HYPERBALL ON A MACBOOK PRO LAPTOP (2.6 GHz INTEL I7, 8 GIB RAM, 8 CORES) AND ON A 32-CORE, 64 GIB RAM WORKSTATION USING 2.3 GHz AMD OPTERON 6276 PROCESSORS. TIMINGS FOR THE HADOOP IMPLEMENTATION WERE DEDUCTED FROM FIGURE 4(B) OF [5]. NOTE THAT THE BETTER PROCESSOR AND THE SSD DISK OF THE MACBOOK PRO MAKE IT ALMOST TWICE FASTER (PER CORE) THAN THE WORKSTATION.

Size (nodes/arcs)	Hadoop [5]	MacBook	32 cores
20 K / 40 M	250 s	2 s	1 s
59 K / 282 M	1750 s	10 s	4 s
177 K / 1977 M	2875 s	70 s	23 s

than the one used in [5] (13.78%, as communicated by the authors), to make a comparison of the execution times possible.

- A medium-size experiment to verify the convergence properties of our computations. For this purpose, we had to restrict ourselves to a graph for which exact values could be computed using n breadth-first visits. We focused on a public snapshot of Wikipedia[5]. This graph consists of $4\,206\,785$ nodes and $101\,355\,853$ arcs (with average degree 24 and the largest strongly connected component spanning about 89% of the nodes). We performed 100 computations using $p = 4096$ registers per counters, corresponding to a theoretical relative standard deviation of 1.62% for each computation. The exact computation of the centralities required a few days using 40 cores.

- A large-scale experiment using the largest ClueWeb09[6] graph; ClueWeb09 is, at the time of this writing, the largest web graph publicly available, one order of magnitude larger that previous efforts in terms of nodes. It contains $4\,780\,950\,903$ nodes and $7\,939\,647\,896$ arcs. The purpose of this experiment was to show our methods in action on a very large dataset.[7]

In Table I we report the timings for an iteration on the same set of Kronecker graphs used in [5]. A standard workstation with 32 cores using HyperBall is at least 150 times faster than a Hadoop-based implementation using using 50 machines; even a MacBook Pro with 8 cores is at least 50 times faster.

In Figure 1 we report the results of the second set of experiments, which fully confirm our empirical observations on the behaviour of the difference estimator: on average, the

---

[5]Available at `http://law.di.unimi.it/`

[6]A dataset gathered in 2009 within the U.S. National Science Foundation's Cluster Exploratory (CluE) program. The ClueWeb12 graph will be even larger, but it is presently still under construction. See http://lemurproject.org/clueweb09/

[7]We remark that due to the way in which the graph has been collected (e.g., probably starting from a large seed) the graph is actually significantly less dense than a web graph obtained by breadth-first sampling or similar techniques. Moreover, the graph contains the whole set of discovered nodes, even if only about 1.2 billion pages were actually crawled. As a result, many statistics are off scale: the harmonic diameter [10], [23] is ≈ 15131 (typical values for breadth-first web snapshots are ≈ 20) and the giant component is just 0.6% of the whole graph.

Closeness centrality · **Lin centrality** · **Harmonic centrality** · **# coreachable nodes**
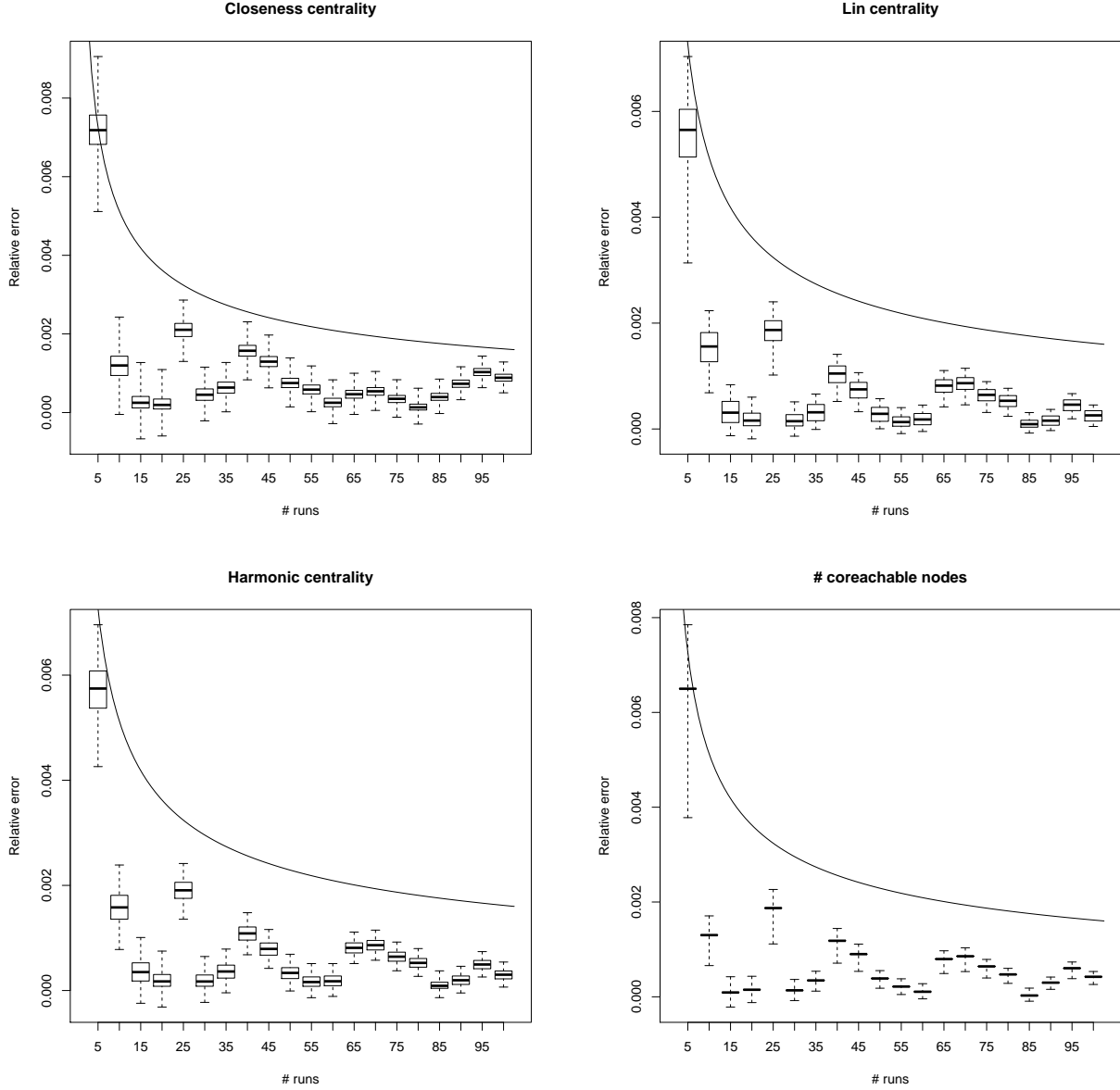
Fig. 1. Relative errors in the computation of centrality measures on Wikipedia: we averaged the values computed in 5, 10, 15, ..., 100 runs and computed the relative error with respect to the real value (the latter were obtained by running an exact implementation). The boxes represent the 1st (lower edge), 2nd (i.e., the median; midline) and 3rd (upper edge) quartile; the whiskers correspond to an interval of length 2σ around the mean. For comparison, each plot contains the curve of the theoretical relative standard deviation for each single HyperLogLog counter over the given number of samples.

relative error on the computed centrality indices is very close to the theoretical prediction for each single HyperLogLog counter, and, in fact, almost always significantly smaller.

It is interesting to observe that the estimation on the number of coreachable nodes (depending on the value of a single counter at the end of the computation) is extremely more concentrated. This is due both to the lack of differences, which reduces the error, and to the fact that most nodes (89%) lie in the giant strongly connected component, so their coreachable set is identical, and this induces a collapse of the quartiles of

the error on the median value.

On the same dataset, Table II reports figures showing that increasing the number of cores leaves essentially unmodified the time per arc per core (i.e., linear scalability). The only significant (30%) increase happen at 32 cores, and it is likely to be caused by the nonlinear cost of caching.

Finally, we ran HyperBall on ClueWeb09 using a workstation with 40 Intel Xeon E7-4870 at 2.40 GHz and 1 TiB of RAM (with the same hardware, we could have analysed a graph with 50 billion nodes using $p = 16$). We report

TABLE II.   TIME PER ARC PER CORE OF A HYPERBALL ITERATION, TESTED ON THE WIKIPEDIA GRAPH WITH $p = 4096$.

cores	Time per arc per core
1	906 ns
2	933 ns
4	967 ns
8	1018 ns
16	1093 ns
32	1389 ns

TABLE III.   TIMINGS FOR A FULL 40-CORE COMPUTATION (≈ 200 ITERATIONS) ON CLUEWEB09 USING A DIFFERENT NUMBER p OF REGISTERS PER HYPERLOGLOG COUNTER. THE AMOUNT OF MEMORY DOES NOT INCLUDE 7.2 GiB OF SUCCINCT DATA STRUCTURES THAT STORE POINTERS TO THE MEMORY-MAPPED ON-DISK BITSTREAMS REPRESENTING THE GRAPH AND ITS TRANSPOSE.

p	Memory	Overall time	Per iteration (avg.)
16	73 GiB	96 m	27 s
64	234 GiB	141 m	40 s
256	875 GiB	422 m	120 s

the results in Table III. We performed three experiments with different levels of precision, and in the one with the highest precision we fully utilized the in-core memory: the timings show that increasing the precision scales even better than linearly, which is to be expected, because the cost of scanning the graph is constant whereas the cost of computing with greater precision grows linearly with the number of registers per HyperLogLog counter. Thus, for a fixed desired precision a greater amount of in-core memory translates into higher speed.

IX.   CONCLUSIONS AND FUTURE WORK

We have described HyperBall, a framework for in-core approximate computation of centralities based on the number of (possibly weighted) nodes at distance exactly t or at most t from each node x of a graph. With 2 TiB of memory, HyperBall makes it possible to compute accurately and quickly harmonic centrality for graphs up to a hundred billion nodes. We obtain our results with a mix of approximate set representations (by HyperLogLog counters), efficient compressed graph handling, and succinct data structures to represent pointers (that make it possible to access quickly the memory-mapped graph representation).

We provide experiments on a 4.8 billion node dataset, which should be contrasted with previous literature: the largest dataset in [5] contains 25 million nodes, and the dataset of [14] contains 1.4 billion nodes. Moreover, both papers provide timings only for a small, $\approx 177\,000$-nodes graph, whereas we report timings for all our datasets.

REFERENCES

[1] A. Bavelas, "A mathematical model for group structures," *Human Organization*, vol. 7, pp. 16–30, 1948.

[2] P. Boldi and S. Vigna, "Axioms for centrality," *CoRR*, vol. abs/1308.2140, 2013.

[3] P. Boldi, M. Rosa, and S. Vigna, "HyperANF: Approximating the neighbourhood function of very large graphs on a budget," in *Proceedings of the 20th international conference on World Wide Web*, S. Srinivasan, K. Ramamritham, A. Kumar, M. P. Ravindra, E. Bertino, and R. Kumar, Eds.   ACM, 2011, pp. 625–634.

[4] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *OSDI '04: Sixth Symposium on Operating System Design and Implementation*, 2004, pp. 137–150.

[5] U. Kang, S. Papadimitriou, J. Sun, and H. Tong, "Centralities in large networks: Algorithms and observations," in *Proceedings of the Eleventh SIAM International Conference on Data Mining*.   SIAM / Omnipress, 2011, pp. 119–130.

[6] S. P. Borgatti, "Centrality and network flow," *Social Networks*, vol. 27, no. 1, pp. 55–71, 2005.

[7] L. Backstrom, P. Boldi, M. Rosa, J. Ugander, and S. Vigna, "Four degrees of separation," in *ACM Web Science 2012: Conference Proceedings*.   ACM Press, 2012, pp. 45–54, best paper award.

[8] A. Bavelas, "Communication patterns in task-oriented groups." *Journal of the Acoustical Society of America*, 1950.

[9] N. Lin, *Foundations of Social Research*.   New York: McGraw-Hill, 1976.

[10] M. Marchiori and V. Latora, "Harmony in the small-world," *Physica A: Statistical Mechanics and its Applications*, vol. 285, no. 3-4, pp. 539 – 546, 2000.

[11] C. R. Palmer, P. B. Gibbons, and C. Faloutsos, "Anf: a fast and scalable tool for data mining in massive graphs," in *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*.   New York, NY, USA: ACM, 2002, pp. 81–90.

[12] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm," in *Proceedings of the 13th conference on analysis of algorithm (AofA 07)*, 2007, pp. 127–146.

[13] M. Durand and P. Flajolet, "Loglog counting of large cardinalities (extended abstract)," in *Algorithms - ESA 2003, 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003, Proceedings*, ser. Lecture Notes in Computer Science, G. D. Battista and U. Zwick, Eds., vol. 2832.   Springer, 2003, pp. 605–617.

[14] U. Kang, C. E. Tsourakakis, A. P. Appel, C. Faloutsos, and J. Leskovec, "HADI: Mining radii of large graphs," *ACM Transactions on Knowledge Discovery from Data*, vol. 5, no. 2, pp. 8:1–8:24, 2011.

[15] E. Cohen, "All-distances sketches, revisited: Scalable estimation of the distance distribution and centralities in massive graphs," *CoRR*, vol. abs/1306.3284, 2013.

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

[17] S. Vigna, "Broadword implementation of rank/select queries," in *Experimental Algorithms. 7th International Workshop, WEA 2008*, ser. Lecture Notes in Computer Science, C. C. McGeoch, Ed., no. 5038.   Springer–Verlag, 2008, pp. 154–168.

[18] P. Elias, "Efficient storage and retrieval by content and address of static files," *J. Assoc. Comput. Mach.*, vol. 21, no. 2, pp. 246–260, 1974.

[19] L. Roditty and V. V. Williams, "Fast approximation algorithms for the diameter and radius of sparse graphs," in *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, D. Boneh, T. Roughgarden, and J. Feigenbaum, Eds.   ACM, 2013, pp. 515–524.

[20] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of ir techniques," *ACM Trans. Inf. Syst.*, vol. 20, no. 4, pp. 422–446, 2002.

[21] L. Katz, "A new status index derived from sociometric analysis," *Psychometrika*, vol. 18, no. 1, pp. 39–43, 1953.

[22] E. Cohen and H. Kaplan, "Spatially-decaying aggregation over a network," *Journal of Computer and System Sciences*, vol. 73, no. 3, pp. 265–288, 2007.

[23] P. Boldi and S. Vigna, "Four degrees of separation, really," in *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*.   IEEE Computer Society, 2012, pp. 1222–1227.