Information Processing Letters 114 (2014) 639–642

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

# Estimating the number of connected components in sublinear time

Petra Berenbrink, Bruce Krayenhoff, Frederik Mallmann-Trenn [*]

*Simon Fraser University, Canada*

A R T I C L E   I N F O

*Article history:*
Received 18 February 2014
Received in revised form 15 May 2014
Accepted 16 May 2014
Available online 29 May 2014
Communicated by Ł. Kowalik

*Keywords:*
Approximation algorithms
Graph algorithms
Connected components
Sublinear time algorithms
Graph theory

A B S T R A C T

We present an algorithm that estimates the number of connected components of a graph over $n$ vertices within an additive error of $\varepsilon n$ in (sublinear) time $O(\frac{1}{\varepsilon^2}\log(\frac{1}{\varepsilon}))$. A connected component $C$ is the maximal subset of nodes of an undirected graph $G$ such that for any two nodes in $C$ there is path connecting. We consider graphs without multiple edges. The *Connected Component Problem* is well-known and amongst the first topics taught in graph theory. The number of connected components can be used to calculate the weight of the minimum spanning tree [1]. Moreover, the study of connected components finds its application in Connected-component labelling, which is used in computer vision. An algorithm runs in sublinear time if its running time is $o(n)$ for an input of size $n$. So far, the best known algorithm was provided by [1]. Their running time is $O(d\varepsilon^{-2}\log(\frac{d}{\varepsilon}))$ where $d$ is the average degree.

## 1. Introduction

Let $G = (V, E)$ be an undirected graph with $n$ nodes, without multiple edges. $G$ may have self-loops. A connected component $C$ is the maximal subset of nodes of an undirected graph $G$ such that for any two nodes in $C$ there is path connecting them. It is well-known that the problem of finding the number of connected components can be solved in time $O(|V|+|E|)$ using breadth-first search (see [2], for example). In this paper we present a sublinear-time algorithm which estimates the number of connected components of $G$ within an additive error of $\varepsilon n$ in time $O(\frac{1}{\varepsilon^2}\log(\frac{1}{\varepsilon}))$. So far, the best known sublinear-time algorithm was provided by [1]. Their running time is $O(d\varepsilon^{-2}\log(d/\varepsilon))$ where $d$ is the average degree. Note that our result is independent of the degree of the graph.

The *Connected Component Problem* is well-known and amongst the first topics taught in graph theory. The number of connected components can be used to calculate the weight of the minimum spanning tree [1]. Moreover, the study of connected components finds its application in Connected-component labelling, which is used in computer vision.

### 1.1. Related work

Calculating the number of connected components in $o(n)$ space or time has received a lot of attention: [3] give a deterministic algorithm to solve the *Connected Component Problem* in logarithmic space. The first sublinear time[1] approximation algorithm was provided by [1]. The authors present an algorithm that approximates the number of connected components with constant probability within a factor of $\varepsilon n$. The running time of their algorithm is $O(d\varepsilon^{-2}\log(\frac{d}{\varepsilon}))$ where $d$ is the average degree. As we explain in Section 2.1, the algorithm presented in this paper uses some of their techniques. The authors of [1] also provide a $\Omega(d\varepsilon^{-2})$ lower bound for estimating the number of connected components for graphs with multiple edges. We note that their lower bound does not apply to graphs

[*] Corresponding author.

[1] The required time is $o(n)$ for an input of size $n$.

http://dx.doi.org/10.1016/j.ipl.2014.05.008
0020-0190/© 2014 Elsevier B.V. All rights reserved.

without multiple edges and hence, their lower bound does not apply to the graph class considered in this paper.

The authors of [4] approximate the number of connected components within a multiplicative error in the metric setting, i.e., in graphs with edge weight satisfying the triangle inequality. Their algorithm runs in $\tilde{O}(\frac{n}{\varepsilon^8})$ time ($\tilde{O}$ is used to suppress logarithmic factors).

There are also results in the property testing model. The authors of [5] give a property-testing algorithm applicable to graphs of bounded degree $D$. Their algorithm accepts $G$ if $G$ is connected. If $G$ is $\varepsilon$-far from being connected (meaning that at least $\varepsilon Dn$ edges have to be added in order to make it connected), their algorithm rejects with a probability of at least $\frac{2}{3}$. The property tester presented has a running time of $O(\log^2(1/(\varepsilon D))/\varepsilon)$.

## 2. The algorithm

Our algorithm uses some ideas of [1]. In the following we first present their algorithm, then we sketch an easy algorithm which has the same approximation as our algorithm, but a running time of $O(\varepsilon^{-4})$. Finally, we present our algorithm in Section 2.3.

### 2.1. Chazelle's algorithm

The algorithm of [1] calculates first $r = O(\frac{1}{\varepsilon^2})$ estimators $\beta_1, \ldots, \beta_r$. Then the algorithm outputs

$$\hat{c} = \frac{n}{2r} \cdot \sum_{i=1}^{r} \beta_i,$$

which is defined as the average of these estimators. The authors show that $\hat{c}$ correctly estimates the number of connected components within an additive factor $\varepsilon n$, with probability of at least 3/4. Each $\beta_i$ is calculated in the following way: A start node $u_i$ is chosen uniformly at random. The algorithm performs a BFS (Breadth-First Search) starting at $u_i$. The BFS visits all neighbours of $u_i$ unless it has a degree greater than $W = 2/\varepsilon$. Afterwards, the algorithm proceeds in rounds. In every round a coin is flipped and if the outcome is tails, then the algorithm sets $\beta_i = 0$. If the outcome is heads, then the algorithm continues the BFS search by visiting (in the current round) at most as many edges as visited in all previous rounds combined. If all edges in the connected component have been explored, then the algorithm returns

$$\beta_i = \frac{d_{u_i} \cdot 2^{\#\text{ of coinflips}}}{\#\text{ visited edges}},$$

where $d_u$ denotes the degree of node $u$. If the connected components size exceeds a threshold of $O(d/\varepsilon)$ or the degree of one of the visited nodes exceeds the estimate of the average degree, then $\beta_i = 0$.

The overall estimator of the number of connected components is the average over all estimators $\beta_i$ scaled by $n$.

### 2.2. An easy algorithm

In order to establish some useful intuition, we briefly discuss an easier algorithm, inspired by [1] for achieving an $\varepsilon n$-approximation with probability at least 3/4 and with a running time of $O(\varepsilon^{-4})$. The algorithm does the following. Sample $r^* = O(\varepsilon^{-2})$ many starting nodes uniformly at random. Let $u_i$ be the node of sample $i$. Perform a BFS starting at $u_i$ until either the component is explored or $W^* = c'\varepsilon^{-1}$ nodes are visited for some fixed constant $c'$. We set $\beta_{u_i} = 1/c_{u_i}$ if the entire component was discovered where $c_{u_i}$ is the size of the connected component of $u_i$. Otherwise, i.e., if the connected component of $u$ was not discovered completely, we set $\beta_{u_i} = 0$. Finally, output $\hat{c}^* = n/r^* \sum_i \beta_i$ as an estimate of the number of connected components.

The total number of queries is $O(\varepsilon^{-4})$ since every BFS has $O(\varepsilon^{-2})$ many edges. By using the same ideas as in [1], one can see that $c - \frac{\varepsilon n}{2} < E[\hat{c}^*] \leq c$, where $c$ is the number connected components. The $\varepsilon n/2$ approximation, is due to the early stopping of explorations visiting $W^*$ many nodes, since the number of connected components having a size of more than $W^*$ is bounded by $\varepsilon n$. By observing that the variance of $\hat{c}^*$ is small and by applying Chebyshev's inequality, one can verify that the algorithm approximates $c$ within an additive error of $\varepsilon n$ with a probability of at least 3/4.

We now use a slightly more sophisticated algorithm to decrease the running time without affecting the error bounds.

### 2.3. Our algorithm

The core of our algorithm is similar to the algorithm of [1]. Our algorithm of calculates first $r = O(\frac{1}{\varepsilon^2})$ estimators $\beta_1, \ldots, \beta_r$ and outputs $\hat{c}$ as the average of these estimators. The difference between the two algorithms lies in the calculation of $\beta_i$. To decide when to stop the BFS walk for a node $v_i$ our algorithm uses a probability distribution inspired by [4]. Let $c_v$ denote the number of nodes in the connected component containing $v$. For every node $v_i$ we define a random variable $X_i$ and explore the component of $v_i$ for $\min\{X_i, W\}$ steps. $X_i$ is drawn according to the probability distribution $P(X \geq j) = \frac{1}{j^2}$. If the entire component is explored, our algorithm sets $\beta_i = c_{v_i}$. Otherwise, i.e., if our algorithm stops exploring the component, our algorithm sets $\beta_i = 0$.

The probability of visiting the whole component of node $v_i$ is $1/c_i^2$ if $c_i \leq W$ and 0 otherwise.

```
ALGORITHM CC:
  Set $r = O(\frac{1}{\varepsilon^2})$ and $W = \frac{2}{\varepsilon}$.
  for $i = 0$ to $r$ do
      Pick a start node $v_i$ uniformly at random.
      Choose $X_i$ according to the prob. distribution $P(X \geq j) = \frac{1}{j^2}$.
      Perform a BFS starting from $v_i$ until one of the following events occurs.
          Event 1) All nodes in $v_i$'s component are explored: Set $\beta_i = c_v$.
          Event 2) $\min\{X_i, W\} + 1$ nodes are explored: Set $\beta_i = 0$.
  end for
  Output $\hat{c} = \frac{n}{r} \sum_{i=1}^{r} \beta_i$ as the estimated number of connected components.
```