Fast average case connectivity algorithms on Random Graphs

Saurav Shekhar (12640)

Undergraduate Project (CS498a) Report
Supervisor: Prof. Surender Baswana

Indian Institute of Technology Kanpur

Contents

Abstract

Random Graphs are represented using the edge set based ($G(n, m)$) or the edge probability based ($G(n, p)$) model. Erdös and Rényi published a collection of results on how different properties of a graph in $G(n, m)$ varied as the graph kept on adding edges (m increased) in 1961. We study the phase transitions a graph undergoes as p varies, particularly the emergence of a giant component around $p = \frac{1}{n}$. Using these results, Richard Karp and Robert Tarjan gave average case linear time algorithms for connectivity problems. We attempt to give a full analysis of Karp's connected components algorithm.

1 Preliminaries: The Random Graph model

We shall assume that all graphs are labelled. This is because most results on labelled graphs can be carried to unlabelled graphs without much difficulty. We assume a graph G to be of n vertices with vertex set $V = \{1, 2, \cdots n\}$ and m edges.

The two most most important models are $G(n, m)$ and $G(n, p)$. Proposed by Edgar Gilbert, $G(n, p)$ model is the most wisely used model in which every possible edge occurs with probability $0 < p < 1$. This gives a binomial distribution on the number of edges with parameter $N = \binom{n}{2}$ and p. $G(n, m)$ model was introduced by Erdös and Rényi in [ER59]. In this model all graphs with n labelled vertices and m edges are equally likely. Thus, the probability of an edge being present is $\frac{m}{\binom{n}{2}}$. In fact, properties of graphs in $G(n, m)$ model are very similar to that of graphs in $G(n, p)$ where $p = \frac{m}{\binom{n}{2}}$.

2 Some results on Random Graphs

In their seminal paper in 1961, Erdos and Renyi [ER61] derived some results on structure of a random graph as we keep on adding edges to it, particularly around $m = n/2$. In $G(n, p)$ notation, this corresponds to $p = \frac{1}{n}$. Results included:

- If $p < \frac{1}{n}$, all connected components of the graph will be of size $O(\log n)$ **whp**
- If $p > \frac{1}{n}$, the graph will have a giant component of size $O(n)$ **whp**
- If $p < \frac{(1-\epsilon)\ln n}{n}$, the graph will be disconnected **whp**
- If $p > \frac{(1+\epsilon)\ln n}{n}$, the graph will be connected **whp**

Thus, the random graph $G(n, p)$ undergoes remarkable phase transition around the edge probability $\frac{1}{n}$ in regarding to connected component size. There are a variety of proofs available for the results (see [Bol98] and [AS92]). Erdös and Rényi used counting. Later proofs used branching models. In 2012, Krivelevich and Sudakov [MK] gave a very simple proof of first two results. We will give a very brief sketch of the proof here.

The proof uses DFS (Depth First Search) algorithm for its argument. In DFS we maintain 3 sets of vertices, set S contains all the vertices who have been explored completely, T contains all the unvisited vertices and $U = V \setminus (S \cup T)$ is the set of vertices that are currently being explored. Vertices in U form a stack. There is an order of priority σ for choosing the vertices. We take $\sigma = 1, 2, 3, \ldots n$ for the proof.

Initially, $S = U = \phi$ and $T = V$. In each round, if the set U is non-empty, the algorithm picks the top vertex u from the stack, scans T according to σ and for each vertex $v \in T$, checks if edge $\{u, v\} \in E$. On finding the first neighbor v of u in T, algorithm removes u from T and pushes it on top of U. If u does not have a neighbor in T, we pop u from U and insert it into S. If U is empty, we choose the first vertex from T (based on σ) and push it into U. This goes on till $U \cup T = \phi$. In order to complete the exploration, after $T \cup U$ becomes empty (and the connected component structure has been revealed), we check all pairs of vertices in S which were not queried before.

Note that as long as U does not go empty, all the vertices pushed into U (and then to S) belong to the same connected component. Thus, from the time the first vertex of a connected component is pushed into (empty) U till U becomes empty again, all vertices are of the same connected component. Some relevant properties of this algorithm:

1. At each round either a vertex moves either from T to U, or from U to S
2. At any stage of the algorithm, there are no edges between current S and current T. Had there been such a edge $\{u, v\}$, we would've found v (u) when searching while u (v) was at the top of stack
3. The set U spans a path from the bottom of the stack to the top. This is because vertex v is pushed on top of u only if $\{u, v\} \in E$

Since the result of each edge query is independent of each other, we can map each query to a Bernoulli random variable with parameter p. Thus an edge exists if the corresponding random variable takes value 1, and 0 otherwise. Thus the probability of each edge being present in the graph is p and the graph obtained is clearly in $G(n, p)$. Each sequence $\bar{X} = (X_i)_{i=1}^{N}$ thus corresponds to a unique graph in $G(n, p)$. We can thus study the component structure using the properties of $\bar{X}$.

The proof uses two important insights, in case of small p, we will not get enough positive query results ($> k$ where $k = O(\log n)$) in a small interval kn so all components will be small in size ($O(\log n)$). And in case of p above threshold, after some time there will exist an interval of atleast a given length such that set U will not become empty during that interval. This will happen because in case U becomes empty, $|S||T|$ (edges not present) becomes large enough that it leads to a contradiction. It is during this period that the giant component is formed.

3 Introduction

Most of the algorithms we see for graph problems are the ones which give bounds on the worst case performance. However, worst case is rarely encountered in practice. Hence it is important to look for algorithms that work fast on the 'average' case. Previous work includes work of Angluin and Valiant [AV79], Karp [Kar79], and transitive closure algorithm of Schnorr [Sch78]. Rajeev Motwani [Mot94] gave algorithms for matching in graphs.

In 1980, Karp and Tarjan [KT80] gave $O(n)$ average time algorithms for finding connected components in a graph with n vertices. They had given a short analysis of the algorithm with some missing details. We will attempt to fill in the gaps and give a more detailed analysis for the same.

4 Algorithm Outline

The input to the algorithm is a graph G with n vertices and m edges. Following the Erdos Renyi model, all graphs with n vertices and m edges are equally likely. A randomly permuted list of all edges is provided. Also, we are provided with randomly permuted adjacency lists $\{\mathbf{adj}(v) | v \in V\}$ for each vertex.

Note that for finding the connected component structure, it suffices if we exhaust all edges in all except one component and explore enough edges in the remaining component so that we know it is connected. If the remaining component is large enough, we will have to look at fewer edges to find the components. Add this to the result by Erdos and Renyi (Theorem 9b in [ER61]) that for a random graph G with n vertices and n edges, there is a constant $\theta > 1/2$ such that **whp** there is a component in G with atleast θn vertices. We thus can use this giant component as the 'remaining' one and exhaust edges in other components.

The algorithm thus consists of 2 stages: a sampling stage in which we find the giant component by adding a random subset of edges part by part. And a cleanup stage, where elements not in the giant component are grouped into components. A priority order σ is assumed on the vertices which is used to start the dfs procedure for each unvisited vertex (for the proof, we assume σ to be the identity permutation).

5 Algorithm

Algorithm 1 Connected Components

```
procedure STAGE I($G$)
    Let $E_0 = \phi$
    while $E_0 \neq E$ do
        Let $D = \phi$
        if $|E - E_0| < n$ then
            $D = E - E_0$
        else
            $D \leftarrow n$ distinct randomly selected edges from $E - E_0$
        Use DFS to find connected components of $G_0 = (V, E_0)$
        if There exists a connected component in $G_0$ of size $> \theta n$ then
            STAGE II
            return
    return
procedure STAGE II($G$)
    Mark vertices of ‘Giant component‘ as ‘giant‘ and remaining as unmarked
    for $v \in \sigma$ do
        if $v$ is unmarked then
            Start DFS from vertex $v$
            if An edge leading to a giant vertex is found then
                Abort dfs and mark all vertices explored as ’giant’
            else
                Mark all vertices visited as a new component
```

Note that STAGE II uses all edges in E and may also transverse edges already transversed in STAGE I.

6 Average case runtime analysis

Here is Theorem 9b in [ER61]

Let $\varrho_{n,N}$ denote the size of the greatest component of $\Gamma_{n,N}$. If $N(n) \sim cn$ where $c > \frac{1}{2}$ we have for any $\eta > 0$

$$\lim_{n \to +\infty} \mathcal{P}\left(|\frac{\varrho_{n,N(n)}}{n} - G(c)| < \eta \right) = 1$$

where $G(c) = 1 - \frac{x(c)}{2c}$ *and* $x(c) = \sum_{k=1}^{\infty} \frac{k^{k-1}}{k!}(2ce^{-2c})^k$ *is the solution satisfying satisfying* $0 < x(c) < 1$ *of the equation* $x(c)e^{-x(c)} = 2ce^{-2c}$.

6.1 Stage I

let p_{in} be the probability that giant component exists in a random graph with n vertices and in edges. Note that $p_{in} > p_{(i-1)n}$ The probability that STAGE I takes i rounds is then

$$\begin{aligned} \mathcal{P}(i) &= (1 - p_n)(1 - p_{2n})(1 - p_{3n}) \dots (1 - p_{(i-1)n})p_{in} \\ &< (1 - p_n)(1 - p_n)(1 - p_n) \dots (1 - p_n)1 \\ &= (1 - p_n)^{i-1} \end{aligned}$$

Also, time taken in total i steps. In the j^{th} step, number of edges are nj. If we do a naive brute force each time, the total time taken is $O(\sum_{j=1}^{i} nj) = O(ni^2)$. We can improve it by 'remembering' the connected component data found in previous round. Since addition of edges will only lead to merging of components, we'll have to somehow support union of components. With disjoint set union, this can be done in $O(\alpha(n)j)$ time. Anyway, we get expected time complexity as

$$O\Big(\sum_{i=1}^{\infty} ni^2(1-p_n)^{i-1}\Big) = O\Big(\frac{n}{p_n^3}\Big) = O(n) \ as\ n \to \infty$$

6.2 Stage II

In a typical step in Stage II, we'll be examining some entry w in adjacency list $\mathbf{adj}(v)$, where v is not yet in the giant component. w is selected randomly among all the remaining entries. Now, observe that none of the edges explored yet lead to a giant component vertex. The number of giant component vertices is $\geq \theta n$, and we are running out of edges that do not lead to giant component while edges which lead to giant component remain as they are. Thus the probability of encountering a giant component vertex is increasing.

If all the vertices were equally likely, we could directly say that the probability of encountering a giant component vertex is $\geq \theta n/(n-1) \geq \theta$. However, it is not so. Consider a vertex whose adjacency list has been exhausted during the dfs by this time. We cannot have an edge from v to this vertex, this would violate the dfs property. For example, consider a vertex u occuring before v in the adjacency list of $parent(v)$. We cannot have an edge $\{v, u\}$ because if such an edge existed, we would've encountered v while exploring u and $parent(v)$ would not have been the parent of v.

Nonetheless, the probability bound is actually correct. Following lemma gives a lower bound on the probability of w being in the giant component.

Lemma 1. Let C_0 be the giant component vertex set in G_0 found during Stage I of the algorithm. Whenever an entry $w \in adj(v)$ is examined, the probability that $w \in C_0$, given that $\{v, w\} \notin E_0$ is $\geq \theta$.

Proof. We would show that among all possible sequences that can arise from the current DFS sequence, the ones where we choose a vertex belonging to the giant component have a higher probability than the ones where we don't.

In the second stage, consider the k^{th} examined edge. All edges examined at this point are the ones selected in stage I, and the ones selected in stage II by this time. The ones selected in stage II by this time can be represented by a sequence $\alpha = (v_1, w_1), \dots (v_{k-1}, w_{k-1})$. i^{th} edge examination selects w_i from $adj(v_i)$. If all of $adj(v_i)$ has been previously selected, $w_i = \mathbf{null}$. The sequence α is called *trace* of the algorithm.

Let $\hat{G} = (V, \hat{E})$ where $\hat{E} = E - E_0$ be the 'residual graph' at end of Stage I. $\hat{G}$ satisfies following 2 conditions:

1. $|\hat{E}| = m - |E_0|$
2. $\hat{E} \cap E_0 = \phi$

   Given $\hat{G}, E_0$ the probability that algorithm produces trace α is denoted by $p(\alpha|\hat{G}, E_0)$. α satisfies the following conditions:

3. If (v_i, w_i) occurs in α and $w_i \neq \mathbf{null}$, then $\{v_i, w_i\} \in E_0 \cup \hat{E}$
4. If (v_i, w_i) occurs in α and $w_i = \mathbf{null}$, then set of edges incident on v_i in $\hat{G}$ are

$$\{\{v_j, w_j\} | j < i, v_i \in \{v_j, w_j\}, \{v_j, w_j\} \notin E_0\}$$

Let S denote set of all graphs which satisfy (1) - (4). S is the probability space over all possible graphs(traces) over which stage II will run.

Consider the i^{th} examination. Once $(v_1, w_1), (v_2, w_2), \dots (v_{i-1}, w_{i-1})$ is determined, v_i will either be a new vertex (start of component, determined by σ), or it will be one of $w_1, w_2, \dots w_{i-1}$. Also, v_i will be uniquely determined by E_0 and $(v_1, w_1), (v_2, w_2), \dots (v_{i-1}, w_{i-1})$. Each permutation of edges incident on v_i is equally likely (some will be selected in Stage I , and others explored in Stage II). Let b_i edges incident on v_i be selected in E_0. If a_i vertices have already been selected from $\mathbf{adj}(v_i)$ and $d(v_i, G)$ denotes the degree of v_i in graph G,

$$\mathcal{P}(w_i \textit{ selected in ith query} | E_0, (v_1, w_1) \dots (v_{i-1}, w_{i-1})) = \frac{1}{d(v_i, G) - a_i}$$

$$\Rightarrow \mathcal{P}(\textit{ith edge in } \alpha = (v_i, w_i) | \hat{G}, E_0, (v_1, w_1) \dots (v_{i-1}, w_{i-1})) = \frac{1}{d(v_i, G) - a_i} = \frac{1}{d(v_i, \hat{G}) - a_i + b_i}$$

$$\Rightarrow p(\alpha | \hat{G}, E_0) = \prod_{i=1}^{k-1} \frac{1}{d(v_i, \hat{G}) - a_i + b_i} \tag{1}$$

Note that the function is decreasing with $d(v_i, \hat{G})$. Define function $f_{u,\alpha}$ by

$$f_{u,\alpha}(d(u, \hat{G})) = \prod_{i, v_i = u} \frac{1}{d(v_i, \hat{G}) - a_i + b_i}$$

If there exists no such i, then $f_{u,\alpha}(d(u, \hat{G}))$ will be 1. Also, none of the v_i is in C_0, thus we can now write

$$p(\alpha | \hat{G}, E_0) = \prod_{u \in V - C_0} f_{u,\alpha}(d(u, \hat{G})) \tag{2}$$

Denote by $p(\hat{G}|E_0)$ the probability that $\hat{G}$ is the residual graph given E_0 and by $p(\hat{G}|\alpha, E_0))$ the probability that $\hat{G}$ is the residual graph, given α is trace. Since all sets of edges are equally likely, $p(\hat{G}|E_0)$ is constant (equal to $\frac{1}{\binom{N-|E_0|}{m-|E_0|}}$). Using Baye's theorem,

$$p(\hat{G}|\alpha, E_0) = \frac{p(\alpha|\hat{G}, E_0) p(\hat{G}|E_0)}{p(\alpha|E_0)}$$

$$\Rightarrow p(\hat{G}|\alpha, E_0) = c \prod_{u \in V - C_0} f_{u,\alpha}(d(u, \hat{G})) \tag{3}$$

After α has occurred, we are choosing $w_k \in \mathbf{adj}(v_k)$.We call w_k eligible if $w_k \neq v_k$, $\mathbf{adj}(w_k)$ is not exhausted and $\{v_k, w_k\} \notin E_0$. Also, let x be a vertex in the giant component. We will now show that $\mathcal{P}(w_k = x | w_k \textit{ is eligible}) \geq 1/(n-1)$.

For a positive integer d, let $G_d = \{\hat{G} \in S | d(v_k, \hat{G}) = d\}$. Let $G_{d+} = G_d \cap \{\hat{G} = (V, \hat{E} | \{v_k, x\} \in \hat{E}\}$ and $G_{d-} = G_d \setminus G_{d+}$. Relation R on $G_{d+} \times G_{d-}$ is defined by $(H_1, H_2) \in R$ iff $H_1 \in G_{d+}, H_2 \in G_{d-}$ and H_2 is obtained from H_1 by deleting edge $\{v_k, x\}$ and adding some other edge $\{v_k, x'\}$ incident to v_k such that x' is eligible. Now

1. If $(H_1, H_2) \in R$ then in going from H_1 to H_2, we remove $\{v_k, x\}$ and add $\{v_k, x'\}$. If $x' \in C_0$ then $p(\hat{G}|\alpha, E_0)$ remains the same. However, if $x' \notin C_0$ then $d(x', \hat{G})$ increases, decreasing $p(\hat{G}|\alpha, E_0)$ (refer to 3). This implies, $p(H_1|\alpha, E_0) \geq p(H_2|\alpha, E_0)$.

2. Fixing H_1, we have $\leq (n-1-d)$ options for x' (x' must be eligible). Thus for fixed H_1 we have the atmost $n - 1 - dH_2$ where $(H_1, H_2) \in R$.

3. For fixed H_2, we can replace d of them with x. Thus, there are d graphs H_1 with $(H_1, H_2) \in R$.

If we sum over $p(H_1|\alpha, E_0) \geq p(H_2|\alpha, E_0)$ over all pairs $(H_1, H_2) \in R$, each H_1 would've $\leq (n-1-d)$ repititions. Each H_2 will've exactly d repititions. Thus

$$(n-1-d) \sum_{H_1 \in G_{d+}} p(H_1|\alpha, E_0) \geq d \sum_{H_2 \in G_{d-}} p(H_2|\alpha, E_0)$$

$$\Rightarrow \sum_{H \in G_d} p(H|\alpha, E_0) \geq \frac{d}{n-1} \sum_{\hat{G} \in G_d} p(\hat{G}|\alpha, E_0)$$

Thus given $d(v_k, \hat{G}) = d$ and α, the conditional probability that $\hat{G}$ contains $\{v_k, x\}$ is $\geq d/(n-1)$. The probability that $w_k = x$ is $\geq (\frac{1}{d})\frac{d}{n-1} = \frac{1}{n-1}$. This is true for any d and given $x \in C_0$. Summing over all x in C_0 gives me $\mathcal{P}(w_k \in C_0) \geq \theta n/(n-1) \geq \theta$ and hence the lemma is proved.

Note that for any vertex v, the number enteries $w \in \mathbf{adj}(v)$ examined such that $w \in C_0$ is ≤ 1 (we abort as soon as we get the giant component). Using lemma 1, we see that for $\{v, w\} \notin E_0$, probability that w is in C_0 is $\geq \theta$. Thus, the expected number of w examined where $\{v, w\} \notin E_0$ is $\leq 1/\theta$. Summing over all V, the expected number of vertices examined in $\hat{E}$ is $O(\frac{n}{\theta})$. Thus, the expected running time of Stage II is $O(n)$.

Therefore, running time of the given algorithm is $O(n)$.

7 Conclusion and Future Work

The technique of analysing algorithms on Random Graphs gives insight about average case preformance of various algorithms. We could use these techniques to analyse more algorithms. One such interesting problem is the one of incrementally maintaining a DFS tree in an undirected graph. Recently Baswana et al [BK14] published two algorithms for maintaining the DFS tree in an incremental graph, one with complexity $O(n^{3/2}\sqrt{m})$ and other with complexity $O(n^2)$. It has been shown that on random graphs the first algorithm takes time closer to $O(n^2\sqrt{\log n})$. We believe that further work is possible on this and it can be reduced to $O(n^2)$.

8 Acknowledgement

I would like to express my gratitude to Prof. Surender Baswana for introducing me to this wonderful field of random graphs. He has been a source of constant motivation and guidance. He has taught me many lessons (academic as well as non-academic) which I will take with me.

I would like to thank my friends for their constant support throughout the semester. I would like to thank Ayush Goel for his inputs in the project.

9 Bibliography

References

[AS92] N Alon and JH Spencer. The probabilistic method. *Wiley-Interscience, New York*, 1992.

[AV79] D. Angluin and L.G. Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. *Journal of Computer and System Sciences*, 18(2):155 – 193, 1979.

[BK14] Surender Baswana and Shahbaz Khan. Incremental algorithm for maintaining dfs tree for undirected graphs. In *Automata, Languages, and Programming*, pages 138–149. Springer Berlin Heidelberg, 2014.

[Bol98] Béla Bollobás. *Random graphs*. Springer, 1998.

[ER59] P ERDOS and A RENYI. On random graphs i. *Publ. Math. Debrecen*, 6:290–297, 1959.

[ER61] Paul Erdos and Alfréd Rényi. On the evolution of random graphs. *Bull. Inst. Internat. Statist*, 38(4):343–347, 1961.

[Kar79] Richard M. Karp. A patching algorithm for the nonsymmetric traveling-salesman problem. *SIAM J. Comput.*, 8(4):561–573, 1979.

[KT80] Richard M Karp and Robert Endre Tarjan. Linear expected-time algorithms for connectivity problems. In *Proceedings of the twelfth annual ACM symposium on Theory of computing*, pages 368–377. ACM, 1980.

[MK] Benny Sudakov Michael Krivelevich. The phase transition in random graphs - a simple proof.

[Mot94] Rajeev Motwani. Average-case analysis of algorithms for matchings and related problems. *J. ACM*, 41(6):1329–1356, 1994.

[Sch78] Claus-Peter Schnorr. An algorithm for transitive closure with linear expected time. *SIAM J. Comput.*, 7(2):127–133, 1978.