

# Testing $k$ -Connectivity and Bipartiteness in Bounded-degree/Sparse Graphs

We shall continue today to discuss results in the bounded-degree and sparse models, and if time permits will start talking about the dense model. We shall actually focus on the bounded-degree model so as to simplify the presentation.

## 1 Algorithmic Aspects of Testing $k$ -Connectivity

Recall that the connectivity (1-connectivity) testing algorithm is based on the observation that if a graph is far from being connected then it contains many small connected components. As mentioned in the last lecture, there is a generalization to  $k > 1$ . What can be shown is that if a graph is far from being  $k$ -connected then it contains *many* subsets  $C$  that are *small* and such that: (1)  $(C, \bar{C}) = \ell < k$ ; (2) for every  $C' \subset C$ ,  $(C', \bar{C}') > \ell$ . We say in this case the  $C$  is  $\ell$ -*extreme*.

As in the case of connectivity, we shall uniformly select a sufficient number of vertices and for each we shall try and detect whether it belongs to a small  $\ell$ -extreme set  $C$  for  $\ell < k$ . The algorithmic question is how to do this in time that depends only on the size of  $C$  and possibly  $d$  and  $k$ . There are special purpose algorithms for  $k = 2$  and  $k = 3$ , but I would like to discuss the general  $k > 3$ .

The problem is formalized as follows: Given a vertex  $v \in C$  where  $C$  is an  $\ell$ -extreme set for  $\ell < k$  and  $|C| \leq t$ , describe a (possibly randomized) procedure for finding  $C$  (with high probability), when given access to neighbor queries in the graph.

The suggested procedure is an iterative procedure: at each step it has a current subset of vertices  $S$  and it adds a single vertex to  $S$  until  $|S| = t$  or a cut of size less than  $k$  is detected. To this end the procedure maintains a cost that is assigned to every edge incident to vertices in  $S$ . Specifically, initially  $S = \{v\}$ . At each step, the procedure considers all edges in the cut  $(S, \bar{S})$ . If an edge was not yet assigned a cost, then it is assigned a cost uniformly at random from  $[0, 1]$  (implementation). Then the edge in  $(S, \bar{S})$  that has the minimum cost among all cut edges is selected. If this edge is  $(u, v)$  where  $u \in S$  and  $v \in \bar{S}$ , then  $v$  is added to  $S$ . The procedure is actually repeated  $\Theta(t^2)$  times. Our goal is to prove that a single iteration of the procedure succeeds in reaching  $S = C$  with probability at least  $t^{-2}$ . Before doing so observe that the total running time is upper bounded by  $O(t^2 \cdot t \cdot d \log(td)) = \tilde{O}(t^3 \cdot d)$ .

For our purposes it will be convenient to represent  $\bar{C}$  by a single vertex  $x$  that has  $\ell$  neighbors in  $C$ . Since, if the procedure traverses an edge in the cut  $(C, \bar{C})$  we view this as “losing”, we are not interested in any other information regarding  $\bar{C}$ . Let this new graph, over at most  $t + 1$  vertices, be denoted by  $G_C$ . Note that since  $C$  is an  $\ell$ -extreme set then every  $v \in C$  has degree greater than  $\ell$ . The first observation is that though our procedure assigns costs in an online manner, we can

think of the random costs being assigned ahead of time, and letting the algorithm “reveal” them as it goes along.

Consider any spanning tree  $T$  of the subgraph induced by  $C$  (this is the graph  $G_C$  minus the “outside” vertex  $x$ ). We say that  $T$  is *cheaper than the cut*  $(C, \overline{C})$  if all  $t - 1$  edges in  $T$  have costs that are lower than all costs of edges in the cut  $(C, \overline{C})$ .

**Claim 1** *Suppose that the subgraph induced by  $C$  has a spanning tree that is cheaper than the cut  $(C, \overline{C})$ . Then the search process succeeds in finding the cut  $(C, \overline{C})$ .*

**Proof:** We prove, by induction on the size of the current  $S$ , that  $S \subseteq C$ . Since the procedure stops when it find a cut of size less than  $k$ , it will stop when  $S = C$ . Initially,  $S = \{v\}$  so the base of the induction holds. Consider any step of the procedure. By the induction hypothesis, at the start of the step  $S \subseteq C$ . If  $S = C$  then we are done. Otherwise,  $S \subset C$ . But this means that there is at least one edge from the spanning tree in the current cut  $(S, \overline{S})$  (that is, an edge connecting  $v \in S$  to  $u \in C \setminus S$ ). But since all edges in  $(C, \overline{C})$  have a greater cost, one of the spanning tree edges must be selected, and the induction step holds. ■

**KARGER’S ALGORITHM.** It remains to prove that with probability  $\Omega(t^{-2})$ ,  $C$  has a spanning tree that is cheaper than the cut. To this end we consider a randomized algorithm for finding a minimum cut in a graph known as “Karger’s min-cut algorithm”. Note that while there are similarities between the local search procedure we described and Karger’s algorithm, they are somewhat different (discuss this more later), and here we use Karger’s algorithm only as an analysis techniques. (Prim vs. Kruskal?)

Karger’s algorithm works iteratively as follows. It starts from the original graph (in which it wants to find a min-cut) and at each step it modifies the graph and in particular decreases the number of vertices in the graph. An important point is that the intermediate graphs may have parallel edges (even if the original graph does not). The modification in each step is done by *contracting* two vertices that have at least one edge between them. After the contraction we have a single vertex instead of two, but we keep all edges to other vertices. That is, if we contract  $u$  and  $v$  into a vertex  $w$ , then for every edge  $(u, z)$  such that  $z \neq v$  we have an edge  $(w, z)$  and similarly for  $(v, z)$ ,  $z \neq u$ . The edges between  $u$  and  $v$  are discarded (i.e., we don’t keep any self-loops). The algorithm terminates when only two vertices remain: Each is the contraction of one side of a cut, and the number of edges between them is exactly the size of the cut.

The contraction is performed by selecting, uniformly at random, an edge in the current graph, and contracting its two endpoints. Recall that we have parallel edges, so the probability of contracting  $u$  and  $v$  depends on the number of edges between them. An equivalent way to describe the algorithm is that we first uniformly select a random ordering (permutation) of the edges (by picking the first, second and so on sequentially), and then, at each step we contract the next edge (that is still in the graph) according to this ordering. (To be convinced that we get exactly the same distribution, it might be helpful to think of Karger’s algorithm as selecting edges uniformly among all edges (i.e., including those that are “within a vertex”), but where each edge is selected only once, and if an “internal edge” is selected, then it does nothing.) To get a random ordering we can simply assign random costs in  $[0, 1]$  to the edges in the graph (which induces a random ordering of the edges).

Now, as a thought experiment, consider an execution of Karger’s min-cut algorithm on  $G_C$

(whose min-cut is  $(C, \{x\})$ ). If the algorithm succeeds in finding this cut (that is, it ends when  $C$  is contracted into a single vertex and no edge between  $C$  and  $x$  is contracted), then the edges it contracted along the way constitute a spanning tree of  $C$  and this spanning tree is cheaper than the cut. So the probability that Karger's algorithm succeeds is a lower bound on the probability that there exists a spanning tree that is cheaper than the cut, which is a lower bound on the success probability of the local search procedure. (The local search procedure also defines a spanning tree, but its process is similar to Prim, vs. Karger's, which is similar to Kruskal.)

Returning to the original description of Karger's algorithm, the simple key point is that, since  $C$  is an  $\ell$ -extreme set, at *every step* of the algorithm the degree of every vertex is at least  $\ell + 1$  (a vertex corresponds to a contracted subset  $C' \subset C$ ). Thus, at the start of the  $i$ th contraction step the current graph contains at least  $\frac{(n-(i-1)) \cdot (\ell+1) + \ell}{2}$  edges. Hence, the probability that no cut edge is contracted is at least

$$\prod_{i=1}^{t-1} \left( 1 - \frac{2\ell}{(t-(i-1)) \cdot (\ell+1) + \ell} \right) = \prod_{i=0}^{t-2} \left( 1 - \frac{2\ell}{(t-i) \cdot (\ell+1) + \ell} \right) \quad (1)$$

$$= \prod_{i=0}^{t-2} \frac{(t-i)(\ell+1) - \ell}{(t-i)(\ell+1) + \ell} \quad (2)$$

$$= \prod_{j=2}^t \frac{j - \ell/(\ell+1)}{j + \ell/(\ell+1)} \quad (3)$$

$$> \prod_{j=2}^t \frac{j-1}{j+1} > \frac{6}{t^2} \quad (4)$$

(If  $\ell$  is small, e.g.,  $\ell = 1$  then the probability is even higher.)

**A Comment.** Note that the local search procedure *does not* select to traverse at each step a random edge in the cut  $(S, \bar{S})$ . To illustrate why this would not be a good idea, consider the case in which  $k = 1$ ,  $C$  is a cycle, and there is one edge from a vertex  $v$  in  $C$  to  $x$ . If we executed the alternative algorithm then once  $v$  would be added to  $S$ , at each step the probability that the cut edge is traversed, would be  $1/3$ , and the probability this doesn't happen would be exponential in  $t$ . On the other hand, the way our procedure works, it succeeds with probability  $\frac{1}{t}$  because that is the probability that the cut edge gets the maximum cost (it has equal probability for every rank).

## 2 Testing Bipartiteness in Bounded-Degree Graphs

Up till now, the algorithms we have seen in the bounded-degree and sparse models work by performing various forms of local search. That is, the uniformly select vertices, and for each vertex selected they “search around the vertex” and views a number of vertices that is polynomial in  $1/\epsilon$ . A property in which different techniques are used is bipartiteness.

First recall the definition of bipartiteness and define what it means for a partition to be  $\epsilon$ -good ( $\epsilon$ -bad). The first result that was proved for bipartiteness in this model is that it is more costly than the other properties we have seen and mentioned. In particular, any algorithm for testing bipartiteness in this model must perform  $\Omega(\sqrt{n})$  queries (for constant  $\epsilon$ ). Namely, it was shown that in less than this number of queries an algorithm cannot distinguish between (of degree 3) that are bipartite, and graphs that are  $\Theta(1)$ -far from bipartite. We won’t get into the lower bound, but it will just give us context for the algorithm we shall see. The running time of the algorithm will be  $O(\sqrt{n} \cdot \text{poly}(\log n/\epsilon))$ .

The algorithm is based on performing *random walks*. In each step of the random walk on a graph of degree at most  $d$ , if the current vertex is  $v$ , then the walk continues to each of  $v$ ’s neighbors with equal probability  $\frac{1}{2d}$  and remains in place with probability  $1 - \frac{\text{deg}(v)}{2d}$ . An important property of random walks on connected graphs is that, no matter in what vertex we start, if we consider the distribution reached after  $t$  steps, then for sufficiently large  $t$ , the distribution is very close to uniform. The sufficient number of steps  $t$  depends on properties of the graph (and of course how close we need to be to uniform). In particular, when  $t$  is relatively small (i.e., logarithmic in  $n$ ), then we say that the graph is *rapidly mixing*. It is well known that an important family of graphs called *expander* graphs (in which subsets of vertices have relatively many neighbors outside the subset), are rapidly mixing.

We shall describe the algorithm, and give a sketch of the proof of its correctness for rapidly mixing graphs.

### Algorithm Test-Bipartite

- Repeat  $T = \Theta(\frac{1}{\epsilon})$  times:
  1. Uniformly select  $s$  in  $V$ .
  2. If `odd-cycle( $s$ )` returns found then reject.
- In case the algorithm did not reject in any one of the above iterations, it accepts.

`odd-cycle( $s$ )`

1. Let  $K \stackrel{\text{def}}{=} \text{poly}((\log n)/\epsilon) \cdot \sqrt{n}$ , and  $L \stackrel{\text{def}}{=} \text{poly}((\log n)/\epsilon)$ ;
2. Perform  $K$  random walks starting from  $s$ , each of length  $L$ ;
3. If some vertex  $v$  is reached (from  $s$ ) both on a prefix of a random walk corresponding to an even-length path and on a prefix of a walk corresponding to an odd-length path then return found. Otherwise, return not-found.

Note that while the number of steps performed in each walk is exactly the same ( $L$ ), the lengths of the paths they induce (i.e., removing steps in which the walk stays in place), vary. Hence, in particular, there are even-length walks and odd-length walks. Also note that if the graph is bipartite, then it is always accepted, and so we only need to show that if the graph is  $\epsilon$ -far from bipartite then it is rejected with probability at least  $2/3$ .

We assume the graph is rapidly mixing (with respect to  $L$ ). That is, from each starting vertex  $s$  in  $G$ , and for every  $v \in V$ , the probability that a random walk of length  $L = \text{poly}((\log n)/\epsilon)$  ends at  $v$  is at least  $\frac{1}{2n}$  and at most  $\frac{2}{n}$  — i.e., approximately the probability assigned by the stationary distribution. (Recall that this ideal case occurs when  $G$  is an expander). Let us fix a particular starting vertex  $s$ . For each vertex  $v$ , let  $p_v^0$  be the probability that a random walk (of length  $L$ ) starting from  $s$ , ends at  $v$  and corresponds to an even-length path. Define  $p_v^1$  analogously for odd-length paths. Then, by our assumption on  $G$ , for every  $v$ ,  $p_v^0 + p_v^1 \geq \frac{1}{2N}$ . We consider two cases regarding the sum  $\sum_{v \in V} p_v^0 \cdot p_v^1$  — In case the sum is (relatively) “small”, we show that there exists a partition  $(V_0, V_1)$  of  $V$  that is  $\epsilon$ -good, and so  $G$  is  $\epsilon$ -close to being bipartite. Otherwise (i.e., when the sum is not “small”), we show that  $\Pr[\text{odd-cycle}(s) = \text{found}]$  is constant. This implies that if  $G$  is  $\epsilon$ -far from being bipartite, then necessarily the sum is not “small” (or else we would get a contradiction by the first case). But this means that  $\Pr[\text{odd-cycle}(s) = \text{found}]$  is a constant for every  $s$ , so that if we select a constant number of starting vertices, with high probability for at least one we’ll detect a cycle.

Consider first the case in which  $\sum_{v \in V} p_v^0 \cdot p_v^1$  is smaller than  $\frac{\epsilon}{c \cdot n}$  for some suitable constant  $c > 1$  ( $c > 300$  should suffice). Let the partition  $(V_0, V_1)$  be defined as follows:  $V_0 = \{v : p_v^0 \geq p_v^1\}$  and  $V_1 = \{v : p_v^1 > p_v^0\}$ . Consider a particular vertex  $v \in V_0$ . By definition of  $V_0$  and our rapid-mixing assumption,  $p_v^0 \geq \frac{1}{4n}$ . Assume  $v$  has neighbors in  $V_0$ , and denote this set of neighbors by  $\Gamma_0(v)$ , and their number by  $d_0(v)$ . Then for each such neighbor  $u \in \Gamma_0(v)$ ,  $p_u^0 = p_u^0(L) \geq \frac{1}{4n}$  as well. As you shall show in the homework assignment, this implies that  $p_u^0(L-1) \geq \frac{1}{32n}$ . However, since there is a probability of  $\frac{1}{2d}$  of taking a transition from  $u$  to  $v$  in walks on  $G$ , we can infer that each neighbor  $u$  contributes  $\frac{1}{2d} \cdot \frac{1}{32n}$  to the probability  $p_v^1$ . In other words,

$$p_v^1 = \sum_{u \in \Gamma_0(v)} \frac{1}{2d} \cdot p_u^0(L-1) \geq d_0(v) \cdot \frac{1}{64dn} \quad (5)$$

Therefore, if we denote by  $d_1(v)$  the number of neighbors that a vertex  $v \in V_1$  has in  $V_1$ , then

$$\sum_{v \in V} p_v^0 \cdot p_v^1 \geq \sum_{v \in V_0} \frac{1}{4n} \cdot \frac{d_0(v)}{64dn} + \sum_{v \in V_1} \frac{1}{4n} \cdot \frac{d_1(v)}{64dn} \quad (6)$$

$$= \frac{1}{c'dn^2} \cdot \left( \sum_{v \in V_0} d_0(v) + \sum_{v \in V_1} d_1(v) \right) \quad (7)$$

Thus, if there were many (more than  $\epsilon dn$ ) violating edges with respect to  $(V_0, V_1)$ , then the sum  $\sum_{v \in V} p_v^0 \cdot p_v^1$  would be at least  $\frac{\epsilon}{c'n}$ , contradicting our case hypothesis (for  $c > c'$ ).

We now turn to the second case ( $\sum_{v \in V} p_v^0 \cdot p_v^1 \geq c \cdot \frac{\epsilon}{N}$ ). For every fixed pair  $i, j \in \{1, \dots, K\}$ , (recall that  $K = \Omega(\sqrt{n})$  is the number of walks taken from  $s$ ), consider the 0/1 random variable  $\eta_{i,j}$  that is 1 if and only if both the  $i$ th and the  $j$ th walk end at the same vertex  $v$  but correspond to paths with different parity. Then  $\Pr[\eta_{i,j} = 1] = \sum_{v \in V} 2 \cdot p_v^0 \cdot p_v^1$ , and so  $\text{Exp}[\eta_{i,j}] = \sum_{v \in V} 2 \cdot p_v^0 \cdot p_v^1$ . What we would like to have is a lower bound on  $\Pr[\sum_{i < j} \eta_{i,j} = 0]$ . Since there are  $K^2 = \Omega(n/\epsilon)$

such variables, the expected value of their sum is greater than 1. These random variables are not all independent from each other, nonetheless we can obtain a constant bound on the probability that the sum is 0 using Chebyshev's inequality (as you shall see in the second question in the homework).