Constant-time connectivity tests
Ph­il­ipp­ Klaus­ Krause
Se­p­tem­ber 26, 2022

Abstract
We present im­ple­men­ta­tions of con­stant­time algo­rithms for con­nect­iv­ity tests and re­lated prob­lems. Some are im­ple­men­ta­tions of slight­ly im­proved vari­ants of pre­vi­ously known algo­rithms; for oth­er prob­lems we pre­sent new algo­rithms with sub­stan­tially bet­ter run­time than pre­vi­ous algo­rithms: es­tim­ates of the dis­tance to and tol­er­ant tests­ers for con­nect­iv­ity, 2­edge­con­nect­iv­ity, 3­edge­con­nect­iv­ity, eu­ler­i­an­i­ty.

1 Introduction
Prop­erty test­ing is con­cerned with ex­tremely fast (con­stant­time or oth­er sub­li­near) algo­rithms for ap­prox­i­mate de­ci­sion­mak­ing. While the run­time of con­stant­time algo­rithms does not de­pend on the size of the in­put, the run­time de­pend­ing on para­me­ters, such as the av­er­age de­gree of the in­put graph, or on the max­i­mum al­lowed er­ror of the out­put are still the sub­ject of re­search.

We im­ple­ment­ed con­stant­time graph algo­rithms for test­ing con­nect­iv­ity, 2­edge­con­nect­iv­ity and 3­edge­con­nect­iv­ity for sparse graphs. We im­ple­ment­ed an es­tim­ate of the num­ber of con­nected com­po­nents, es­tim­ates of the dis­tance to con­nect­iv­ity, 2­edge­con­nect­iv­ity and eu­ler­i­an­i­ty, tol­er­ant tests­ers for con­nect­iv­ity, 2­edge­con­nect­iv­ity and eu­ler­i­an­i­ty. We also pre­sent an algo­rithm for es­tim­at­ing the dis­tance to 3­edge­con­nect­iv­ity and a tol­er­ant tester for 3­edge­con­nect­iv­ity that have not been im­ple­ment­ed yet. For con­nect­iv­ity and eu­ler­i­an­i­ty our ap­proach has bet­ter worst­case run­time than pre­vi­ous ap­proaches (and the same ex­pect­ed run­time). For 2­edge­con­nect­iv­ity and 3­edge­con­nect­iv­ity, our ap­proaches have bet­ter run­time than the pre­vi­ously known algo­rithm for gen­eral k­edge­con­nect­iv­ity. These ad­va­tages car­ry over to the tol­er­ant tests­ers.

Full, com­pile­able C source of our im­ple­men­ta­tion can be found as free soft­ware [18] at http://zshg.sourceforge.net

2 Preliminaries
Let G = (V, E) be a graph. For a set of nodes U ⊆ V, the de­gree d(U) is the num­ber of edges with ex­actly one en­d­point in U. For a sin­gle node v ∈ V we also de­fine the de­gree d(v) := d({v}). The num­ber Δ(G) := max{d(v) | v ∈ V} is the max­i­mum de­gree of G. The num­ber

\[ d(G) := \frac{1}{|V|} \sum_{v \in V} d(v) = 2 \frac{|E|}{|V|} \]
is the average degree of $G$.

In the bounded-degree model, a graph $G = (V, E), \Delta \geq \Delta(G)$ is $\epsilon$-far from having a property $\mathcal{P}$, if it cannot be transformed into a graph $G' \in \mathcal{P}, \Delta(G') \leq \Delta$ by at most $\epsilon \Delta |V|$ edge modifications. Complexities are given as functions of the maximum degree $\Delta$.

In the sparse graph model (also called unbounded-degree model), a graph $G = (V, E)$ is $\epsilon$-far from having a property $\mathcal{P}$, if it cannot be transformed into a graph $G' \in \mathcal{P}$ by at most $\epsilon |E|$ edge modifications (i.e. edge insertions and edge deletions). Complexities are usually given as functions of the average degree $d$.

When estimating distance to a property, often the distance is given in terms of edge modifications relative to the number of nodes ($\delta$): $\delta |V| = \epsilon |E|$.

The incidence-lists model consists of the bounded-degree model and the unbounded-degree model. It corresponds to an implementation of the graph data structure using adjacency lists or incidence lists.

In the incidence lists model the available queries are degree and $i$-th neighbour.

Other models include the dense graph model and the combined model (also called the general graph model). The dense graph model corresponds to an implementation of the graph data structure using an adjacency matrix. The available query is adjacency. The combined model allows all queries from the incidence lists and the dense graph models.

All models allow uniform random sampling of nodes.

A graph is connected if it has at least one node and for any two nodes $u, v$ in the graph, there is a path from $u$ to $v$. A connected component of a graph is a maximal connected subgraph. A graph is called $k$-edge-connected, if any graph obtained from it by removing less than $k$ edges is connected. An edge in a graph is a bridge, if all paths connecting its endpoints contain the edge. A $k$-class in a graph $g$ is a $k$-edge connected subset of nodes of a graph $G$ (note that for $k > 2$ a $k$-class in $G$ is not necessarily a $k$-edge-connected subgraph of $G$). A $k$-set in a graph is a $k$-edge-connected subgraph that is connected to the rest of the graph by less than $k$ edges. Every node in a graph is contained in at least one $k$-set. The 1-sets are the connected components. A minimal $k$-set is called a $k$-class-leaf (the terminology is inspired by the use of the word “leaf” for 2-class-leaves [8] and by the creation of a auxiliary graph, that has one node per $k$-class in the original graph. For $k = 2$, this auxiliary graph is the bridge tree. The $k$-class-leaves correspond to nodes of degree 1 in the auxiliary graph.

A set of nodes $U \subseteq V$ is $\ell$-extreme, if $d(U) = \ell$ and each subset of $U$ has degree larger than $\ell$. Every $\ell$-extreme set is an $\ell'$-class for some $\ell'$ and any two extreme sets in a graph are either disjoint or one is a subset of the other [13]. For $U, W \subseteq V$ we use the notation $U \subset W$ to denote that $U \subseteq W$ and $U$ is extreme and there is no extreme set $U'$, such that $U \subseteq U' \subseteq W$.

A graph is eulerian if it contains a cycle that traverses each edge of the graph exactly once.

3 Related Work

For connectivity, there is a relatively simple algorithm with small constants [8] in the bounded-degree model, that also works well in the sparse graph model [15], where it has slightly higher complexity. With a more careful analysis, one
| Property          | Model | Complexity                                      |
|------------------|-------|------------------------------------------------|
| Connectivity     | $\Delta$ | $O\left(\frac{\log^2 \left(\frac{1}{\delta}\right) \epsilon}{\delta^2}\right)$ |
| Connectivity     | $d$    | $O\left(\frac{\log \left(\frac{1}{\epsilon^d}\right)}{\epsilon^d}\right)$ |
| Connectivity     | $d$    | $O\left(\frac{\log \left(\frac{1}{\epsilon^d}\right)}{\epsilon^d}\right)$ |
| Connectivity (here) | $d$ | $O\left(- \log(1-p) \cdot \frac{\log \left(\frac{1}{\epsilon^d}\right)}{\epsilon^d}\right)$ |
| 2-edge-connectivity | $\Delta$ | $O\left(\frac{\log^2 \left(\frac{1}{\epsilon}\right) \epsilon}{\epsilon^2}\right)$ |
| 2-edge-connectivity (here) | $d$ | $O\left(- \log(1-p) \cdot \frac{\log \left(\frac{1}{\epsilon^d}\right)}{\epsilon^d}\right)$ |
| 3-edge-connectivity | $\Delta$ | $O\left(\frac{\log^2 \left(\frac{1}{\epsilon}\right) \epsilon}{\epsilon^2}\right)$ |
| 3-edge-connectivity (here) | $d$ | $O\left(- \log(1-p) \cdot \frac{\log \left(\frac{1}{\epsilon^d}\right)}{\epsilon^d}\right)$ |
| $k$-edge-connectivity | $\Delta$ | $O\left(\frac{k^3 \log \left(\frac{1}{\epsilon^d}\right) \epsilon^3}{\epsilon^2 \Delta^2}\right)$ |
| $k$-edge-connectivity | $d$ | $O\left(\frac{k^4}{\epsilon^d}\right)$ |
| Eulerianity      | $\Delta$ | $O\left(\frac{\log^2 \left(\frac{1}{\epsilon}\right) \epsilon}{\epsilon^2}\right)$ |
| Eulerianity      | $d$    | $O\left(\frac{\log \left(\frac{1}{\epsilon^d}\right)}{\epsilon^d}\right)$ |
| Eulerianity (here) | $d$ | $O\left(- \log(1-p) \cdot \frac{\log \left(\frac{1}{\epsilon^d}\right)}{\epsilon^d}\right)$ |

Figure 1: Comparison of connectivity tests, $\Delta$: bounded-degree model, $d$: sparse graph model.

...can achieve a slightly better complexity bound [14]. Based on the existing approaches, we came up with a connectivity test that uses fewer queries (but has the same asymptotic complexity).

The algorithm for connectivity testing can be generalized to $k$-edge-connectivity testing [8]. These generalizations are more involved than the algorithm for connectivity, but still implementable, which also holds for their sparse graph versions [14]. A graph is eulerian when it is connected and has no nodes of odd degree [7]. Algorithms for connectivity often can be modified to get algorithms testing for eulerianity.

Tolerant testers can be created from algorithms that estimate the distance to having the property (as we do in Section 6). There are various approaches to estimating the distance to connectivity. A recent one [4] has an expected runtime of $O\left(\frac{\log(\delta)}{\delta^2}\right)$, though the worst-case complexity is substantially higher at $O\left(\frac{\log(\delta)}{\delta^2}\right)$. An earlier approach [12] can estimate the distance to $k$-edge-connectivity for any $k$. For $k = 2$ and $k = 3$ we can estimate the distance with lower asymptotic complexity than the general approach.
| Property                  | Model | Complexity                                                      |
|-------------------------|-------|-----------------------------------------------------------------|
| Connectivity [5]         | $d$   | $O\left(\frac{d \log(\frac{d}{\delta})}{\delta^2}\right)$    |
| Connectivity [12]        | $d$   | $O\left(\frac{1}{\delta^2}\right) = O\left(\frac{1}{\pi}\right)$ |
| Connectivity [11]        | $d$   | $O\left(\frac{1}{\pi}\right)$, exp.: $O\left(\frac{\log(\frac{1}{\delta})}{\delta^2}\right)$ |
| Connectivity (here)      | $d$   | $O\left(\frac{\log(\frac{1}{\delta})}{(1-p)\delta^2}\right)$ |
| 2-edge-connectivity (here)| $d$   | $O\left(\frac{\log(\frac{1}{\delta})}{(1-p)\delta^2}\right)$ |
| 3-edge-connectivity (here)| $d$   | $O\left(\frac{\log(\frac{1}{\delta})}{(1-p)\delta^2}\right)$ |
| $k$-edge-connectivity [12]| $d$   | $O\left(\frac{k^6 \log(\frac{1}{\delta})}{\pi^6 \delta^6}\right) = O\left(\frac{k^6 \log(\frac{1}{\delta})}{\pi^6 \delta^6}\right)$ |
| Eulerianity [12]        | $d$   | $O\left(\frac{1}{\pi^2}\right) = O\left(\frac{1}{\pi^2}\right)$ |
| Eulerianity (here)       | $d$   | $O\left(\frac{\log(\frac{1}{\delta})}{(1-p)\delta^2}\right)$ |

Figure 2: Comparison of estimates of distance to connectivity, $d$: sparse graph model.

There was an attempt to construct tolerant testers for $k$-edge-connectivity via local reconstructors [4]. The claimed complexity was substantially higher than previous approaches and exponential in $k$; there are multiple flaws in the proof, that cannot be fixed easily.

4 Connectivity Test

We implemented a connectivity (and 2- and 3-edge connectivity) test for the sparse graph model. It is similar to existing connectivity test algorithms [8, 15, 14], which also carries over to the proofs. Our approach (Figure 3) has a few minor changes to reduce the query complexity by a constant factor. Also, instead of having a fixed probability of $\frac{2}{3}$ for rejecting graphs that are $\epsilon$-far from being connected, we allow a parameter $p$; compared to just running the algorithm repeatedly, this lowers runtime and number of queries when the acceptable probability $1 - p$ of an unconnected graph that is $\epsilon$-far from being connected being considered connected is not a power of $\frac{1}{3}$. The implementations of the algorithms rand_index(), which uniformly selects a random node, and $k$-set$(s, 2^i)$, which return the number of nodes in the minimal $k$-set containing $s$ connected to the rest of the graph by less than $k$ edges are not shown. $1$-set is a connectivity test (which can be implemented as a depth-first search). $2$-set() and $3$-set can be implemented using variants of algorithms for identifying 2-class-leaves and 3-class leaves [8].

**Lemma 1.** The algorithm in Figure 3 accepts all $k$-edge-connected graphs and rejects graphs that are $\epsilon$-far from being $k$-edge-connected with probability at least $p$. 4
bool zshg_k(e, p, d, n)
{
    q := -log_e(1 - p);
    if (n == 0) // Empty graph: unconnected
        return (false);
    if (d < 1) // Low degree: connected if single node only
        return (n == 1);
    ℓ := log_2 \left( \frac{8(\frac{k}{2})}{cd} \right);
    if (n ≤ \frac{8q(\frac{k}{2})\ell}{cd}) // Small graph: do exact check
        return (k_set(0, n) == n);
    for (i := 1; i ≤ ℓ; i := i + 1)
    {
        m_i := \frac{8q(\frac{k}{2})\ell}{2^{cd}};
        for (c := 0; c ≤ m_i; c := c + 1)
        {
            s := rand_index(n);
            if (k_set(s, 2^i) < 2^i)
                return (false);
        }
    }
    return (true);
}

Figure 3: k-edge-connectivity test algorithm in C-like pseudocode (the implementation resides in zshg_impl() in zshg.c)

Proof. Clearly, the algorithm never rejects a k-edge-connected graph. If a graph is ϵ-far from being k-edge-connected, it has more than \frac{en}{\frac{k}{2}} k-class-leaves (otherwise one could add \epsilon mk edges to connect all k-class-leaves by adding \left\lceil \frac{k}{2} \right\rceil cycles through them, resulting in a k-edge-connected graph). Then the graph has at least \frac{en}{\frac{k}{2}} k-class-leaves consisting of less than \frac{4(\frac{k}{2})}{cd} nodes each. Let B_i be the set of all such k-class-leaves of size at least 2^{i-1}, but less than 2^i. Let ℓ := \log_2 \left( \frac{8(\frac{k}{2})}{cd} \right). Since \sum_{i=1}^\ell |B_i| ≥ \frac{en}{\frac{k}{2}}, there exists an i, 1 ≤ i ≤ ℓ, such that |B_i| ≥ \frac{en}{\frac{k}{2}}. The number of nodes residing in k-class-leaves of size at least 2^{i-1}, but less than 2^i is at least 2^{i-1}|B_i| ≥ \frac{2^{i-1}en}{\frac{k}{2}}. The probability of selecting such a node by uniform sampling is \frac{2^{i-1}|B_i|}{n} ≥ \frac{2^{i-1}en}{n\frac{k}{2}} = \frac{2^{i-3}cd}{\frac{k}{2}}. The probability of not
selecting any such node in iteration $i$ of the algorithm bounds the probability $p_{\text{fail}}$ of not rejecting an unconnected graph that is $\epsilon$-far from being connected. Using $1 + x \leq e^x$, which follows from Bernoulli’s inequality [17, 2], we get

$$p_{\text{fail}} \leq \left(1 - \frac{2^{i-3} \epsilon d}{k / 2^i \ell}ight)^{m_i} \leq e^{-\left(\frac{2^{i-3} \epsilon d m_i}{k / 2^i \ell}\right)} \leq e^{-\left(\frac{8q \ell}{ed} \frac{s}{2^i \epsilon d}\right)} = e^{-q} = e^{\log_e(1-p)} = 1 - p.$$

$q. e. d.$

**Lemma 2.** For $k = 1, 2$ the query complexity of the algorithm in Figure 3 is $O\left(\frac{\log(1-p) \log\left(\frac{1}{\epsilon d}\right)}{e^d}\right)$. For $k = 3$ it is $O\left(\frac{\log(1-p) \log\left(\frac{1}{\epsilon d}\right)}{e^d}\right)$.

**Proof.** In each iteration of the algorithm, $m_i$ nodes are selected. For $k = 1$, for each selected node, in $k_{\text{set}}()$ a depth-first-search for up to $2^i$ nodes is done (dfs() in zshg.c), resulting in less than $2^i \cdot 2^i$ queries for each search. The query complexity is thus less than

$$\sum_{i=1}^{\ell} m_i \cdot 2^i \cdot 2^i = \sum_{i=1}^{\ell} \frac{8q \ell}{2^i \epsilon d} \cdot 2^i \cdot 2^i \leq \frac{8q \ell}{ed} 2^{\ell+1} = \frac{8q \log_2\left(\frac{s}{\epsilon d}\right)}{2 \epsilon d} = 128 \frac{q \log_2\left(\frac{s}{\epsilon d}\right)}{\epsilon^2 d^2} = 128 \frac{-\log_e(1-p) \cdot \log_2\left(\frac{s}{\epsilon d}\right)}{\epsilon^2 d^2} \in O\left(\frac{-\log_e(1-p) \cdot \log\left(\frac{1}{\epsilon d}\right)}{\epsilon^2 d^2}\right).$$

For $k = 2$, zshg uses a variant of an algorithm for 2-class-leaves [8] (comp2() in zshg2.c). This algorithm essentially does two depth-first-searches for up to $2^i$ nodes, with the second search somewhat restricted, for a total of at most $(2^i)^2$ queries. This results in a call to $2_{\text{set}}$ needing up to 4 times as many queries as $1_{\text{set}}$, and thus a total factor of about 4 in the number of queries done by zshg2 vs. zshg1. For $k = 3$, we use a variant of an algorithm for 3-class-leaves [8]. This algorithm essentially does one depth-first-search for up to $2^i$ nodes followed by an invocation of $2_{\text{set}}()$ on each node discovered. We get an additional factor of about $\frac{16}{\epsilon d^2}$ in the number of queries by zshg3 vs. zshg1. $q. e. d.$

From our connectivity test, we obtain a test for eulerianity based on the following:

**Lemma 3.** Let $G = (V, E)$ be a graph that is $\epsilon$-far from eulerianity. Then, it is $\frac{\epsilon}{2}$-far from connectivity or it has more than $\epsilon dn$ nodes of odd degree.

**Proof.** Assume that the graph is $\frac{\epsilon}{2}$-close to connectivity (i.e. it has at most $\frac{\epsilon}{2}dn$ connected components) and at most $\frac{\epsilon}{2}dn$ nodes of odd degree. We show that we can make the graph Eulerian by adding $\epsilon dn$ edges, a contradiction. We first add $\frac{\epsilon}{2}dn$ edges connecting nodes of odd degree to obtain a graph in which all nodes have even degree. We then add $\frac{\epsilon}{2}dn$ edges, choosing one node out of each connected component in the original graph and adding a cycle through these. The resulting graph is connected and has only nodes of even degree and is thus eulerian [7, 11, 9]. $q. e. d.$
Theorem 1. There are testers for 1- and 2-edge-connectivity and for euclidianity with complexity $O\left(\frac{\log(1-p) \log(\frac{1}{p})}{c^2 d^2}\right)$, a tester for 3-edge-connectivity with complexity $O\left(\frac{\log(1-p) \log(\frac{1}{p})}{c^3 d^3}\right)$.

Proof. This follows from Lemmata 1, 2, 3. q. e. d.

5 Distance to Connectivity Estimate

Our algorithm for estimating the number of connected components is based on an earlier algorithm [1], which had worst-case query complexity $O(\delta^{-4})$ and expected query complexity $O(\delta^{-2} \log(\delta^{-1}))$. We reduce the worst-case query complexity to $O(\delta^{-2} \log(\delta^{-1}))$ and allow the probability $p$ of the error being outside the error bound to be specified instead of using a fixed value of $\frac{\delta}{4}$.

```c
float 1_set (\delta, \rho, n) {
    \rho = \frac{2 \sqrt{1 - \rho}}{\delta^2};
    a = 0;
    \ell = \rho \left(\log_e (\frac{2}{\rho}) + \frac{9}{2}\right);
    for (i = 0; i < \rho; i++)
        { s = rand_index (n); x = rand_range (\frac{s}{2}); b = 1_set (s, x + 1, &\ell); if (b \leq x)
            a += b; }
    return (\rho n + \frac{\rho n}{4});
}
```

Figure 4: 1-set counting algorithm in C-like pseudocode (implementation: zshg_components() in zshg.c)

Lemma 4. For $\delta \leq 1$, with probability at least $p$, the return value of the algorithm in Figure 4 is the number of connected components in the graph up to an error of at most $\rho n$. 
Proof. Let \( X_i \) be the value of \( x \) in iteration \( i \). Let \( c^* \) be the number of connected components. Let \( c^* \) be the number of connected components of size at most \( \frac{\delta}{2} \). Let \( C \) be the return value of the algorithm in Figure 4. Let \( \hat{c} \) be the return value when ignoring the query bound \( \ell \). Let \( B_i \) be the value of \( b \) in iteration \( i \) if \( b < X_i \) and 0 otherwise. Let \( C_s \) be the size of the connected components containing the node \( s \).

\[
E(B_i) = \frac{1}{n} \sum_{s \in V} C_s P(X_i \geq C_s) = \frac{1}{n} \sum_{s \in V, C_s \leq \frac{\delta}{2}} \frac{1}{C_s} \leq 1.
\]

\[
E(B_i^2) = \frac{1}{n} \sum_{s \in V} C_s^2 P(X_i \geq C_s) = \frac{1}{n} \sum_{s \in V, C_s \leq \frac{\delta}{2}} 1 \leq 1.
\]

\[
Var(B_i) = E(B_i^2) - E(B_i)^2 \leq E(B_i^2) \leq 1.
\]

\[
\hat{c} = \frac{a}{r} \sum_{i=0}^{r-1} + \frac{\delta n}{4}.
\]

\[
E(\hat{c}) = \frac{r E(B_i)n}{r} + \frac{\delta n}{4} = c^* + \frac{\delta n}{4}.
\]

\[
Var(\hat{c}) = \frac{n^2}{r} Var(B_i) \leq \frac{n^2}{r}.
\]

The number of connected components bigger than \( \frac{\delta n}{2} \) is at most \( \frac{\delta n}{2} \). This allows us to bound the probability of the estimate being far off [19] (still ignoring the query bound \( \ell \)).

\[
c - \frac{\delta n}{4} < c^* + \frac{\delta n}{4} = E(\hat{c}) = c^* + \frac{\delta n}{4} \leq c + \frac{\delta n}{4}.
\]

\[
P\left( |\hat{c} - E(\hat{c})| > \frac{3\delta n}{4} \right) < \frac{Var(\hat{c})}{(\frac{3\delta n}{4})^2} \leq \frac{8}{9} (1 - p).
\]

Let \( Q_i \) be the number of queries made by a call to 1-set(s, \( X_i \)) for a uniformly chosen node \( s \) (i.e. we consider the behaviour of the algorithm as if the limit \( \ell \) wasn’t there). The procedure 1-set can be implemented as a depth-first-search, resulting in at most \( \frac{\delta n}{2} \) queries for 1-set(s, \( x \)).
\[
E(Q_i) \leq \sum_{j=1}^{\frac{r-1}{2}} \left( \frac{1}{j^2} - \frac{1}{(j+1)^2} \right) \frac{j(j+1)}{2} + \frac{1}{(\frac{r}{2})^2} \left( \frac{\frac{r}{2}}{2} \right) = \\
\sum_{j=1}^{\frac{r-1}{2}} \left( \frac{2j+1}{2j(j+1)} \right) + \frac{2 + \delta}{4} \leq \log_{e} \left( \frac{2}{\delta} \right) + \frac{1}{2}.
\]

\[
E(Q_i^2) \leq \sum_{j=1}^{\frac{r-1}{2}} \left( \frac{1}{j^2} - \frac{1}{(j+1)^2} \right) \frac{(j(j+1))^2}{2} + \frac{1}{(\frac{r}{2})^2} \left( \frac{\frac{r}{2}}{2} \right)^2 = \\
\sum_{j=1}^{\frac{r-1}{2}} \left( \frac{(j+1)^2}{4} - \frac{j^2}{4} \right) + \frac{(\frac{r}{2} + 1)^2}{4} = \frac{1}{2} \sum_{j=1}^{\frac{r-1}{2}} j + \frac{1}{4} \left( \frac{2}{\delta} - 1 \right) + \frac{(\frac{r}{2} + 1)^2}{4} = \frac{2}{\delta^2} + \frac{1}{\delta}.
\]

\[
Var(Q_i) = E(Q_i^2) - E(Q_i)^2 \leq E(Q_i^2) \leq \frac{2}{\delta^2} + \frac{1}{\delta}.
\]

This allows us to bound the probability of hitting the query bound \( \ell \).

\[
P \left( \sum_{i=0}^{r-1} Q_i > \ell \right) \leq P \left( \sum_{i=0}^{r-1} Q_i > r \left( E(Q_0) + 4 \right) \right) < \frac{Var(Q_0)}{16r} < \frac{1}{9} (1 - p).
\]

We can thus bound the probability of the algorithm returning a result outside the error bounds.

\[
P \left( |C - c| > \delta n \right) \leq P \left( \sum_{i=0}^{r-1} Q_i > \ell \text{ or } |\tilde{c} - E(\tilde{c})| > \frac{3\delta n}{4} \right) \leq \\
P \left( \sum_{i=0}^{r-1} Q_i > \ell \right) + P \left( |\tilde{c} - E(\tilde{c})| > \frac{3\delta n}{4} \right) < (1 - p) .
\]

q. e. d.

**Lemma 5.** The algorithm in Figure 4 has query complexity \( O(\frac{1}{(1 - p)^2} \log_e (\frac{1}{\delta})) \).

*Proof.* This follows directly from the choice of \( \ell \). q. e. d.

While our algorithm for estimating the distance to connectivity did directly reuse the function \( \ell \) set from the connectivity test, this doesn’t work for general \( k \)-edge-connectivity, since for \( k \geq 2 \), not every \( k \)-set is a \( k \)-class-leaf. For \( k = 2 \) we use:

**Lemma 6** (Eswaran and Tarjan [4]). Let \( G = (V, E) \) be a graph. Let \( c_1 \) be the number of 2-class-leaves that are 1-class-leaves in \( G \). Let \( c_2 \) be the number of 2-class-leaves that are not 1-class-leaves in \( G \). If \( |V| > 2 \) and \( c_1 + c_2 > 1 \), the minimal number of edge modifications necessary to make \( G \) 2-edge-connected is \( \left\lceil \frac{c_2}{2} \right\rceil + c_1 \).
Using classic terminology, $c_1$ is the number of connected components in $G$ that are 2-edge-connected and $c_2$ is the number of 2-edge-connected components in $G$ that are connected to the rest of the graph by a single bridge. To estimate the distance to 2-edge-connectivity we want to estimate the numbers $c_1$ and $c_2$.

We use the algorithm in Figure 5 to find out if a node $s$ is in a 2-edge-connected component of size up to $r$, if this is also a connected component, and its size. If $s$ is in a 2-set of size up to $r+1$, that is not a 1-set, the first depth-first search will have a bridge connecting this 2-set to the rest of the graph in its search tree. Thus, after the second search we know if $s$ is in a 2-set of size up to $r$. If we are in such a set, we do a third search to find out if this 2-set is 2-edge-connected and thus a 2-class-leaf. For every node it visits (except for the last one), the first search will issue queries to find more nodes. At each node, at most one query is made that finds a new node or tells us that there are no more neighbours. At each node, we could also make queries that give us edges to nodes that we already visited. The number of these queries is bounded by the nodes already visited. Thus, in the first search there at most $\sum_{i=0}^{r} (1 + (i-1)) = \frac{r(r+1)}{2}$ queries. The second search is similar, but it will have to avoid certain edges, resulting in some additional queries; the number of these edges is $r$, we thus get a bound of $\frac{r(r+1)}{2} + r$ queries. The third search also needs to avoid edges to nodes not found by the second search. But since the second search resulted in a 2-set, there is only one such edge. The number of queries in the second search is thus bounded by $\sum_{i=0}^{r-1} (1 + (i-1)) + (r-1) + 1 = \frac{r(r-1)}{2} + r$. This gives us a bound of at most $\frac{2r(r+1)}{2}$ queries made by the algorithm in Figure 5.

**Lemma 7.** For $\delta \leq 1$, with probability at least $p$, from the return value of the algorithm in Figure 6 we get the minimum number of edge modifications necessary to make the input graph 2-edge-connected up to an error of at most $\frac{1}{2} + \frac{\delta}{2}n$.

**Proof.** Let $c_1^*$ be the number of 2-class-leaves that are 1-class-leaves in $G$ and of size at most $\frac{1}{2}$. Let $c_2^*$ be the number of 2-class-leaves that are not 1-class-leaves in $G$ and of size at most $\frac{1}{2}$. The algorithm in Figure 6 gives us an estimate $(\hat{C}_1, \hat{C}_2)$ for $(c_1^*, c_2^*)$ that we will later use to estimate the distance to 2-edge-connectivity. Let $(\hat{c}_1, \hat{c}_2)$ be the return value of the algorithm in Figure 6 when ignoring the query bound $\ell$. Let $B_1,i$ be the value of $b_1$ in iteration $i$, let $B_2,i$ be the value of $b_2$ in iteration $i$. Let $C_{2,i}$ be the size of the 2-class-leaf that contains $s$. Let $\hat{c} := \frac{\hat{c}_1}{2} + \frac{1}{2} + \hat{c}_1 + \frac{\delta}{2}$ be our estimate of the distance. Similarly to the calculation for the distance to connectivity estimate, we get:
int x int 1_2_leaf(s, r)
{
  1 Do first depth-first search from s for up to r+1 nodes.
  Let n₁ be the number of nodes found.

  2 Do second depth-first search from s for up to r+1 nodes,
  never traversing an edge of the search tree of 1 in the same direction.
  Let n₂ be the number of nodes found.

  if (n₁ == n₂ == r+1) // Not in a 2-set of size at most r
    return (0, 0);

  3 Do third depth-first search from s for up to r nodes,
  only considering nodes found in 2,
  never traversing an edge of the search tree of 2 in the same direction.
  Let n₃ be the number of nodes found.

  if (n₂ == n₃ == n₁) // In a 2-class-leaf of size n₃ that is a 1-class-leaf
    return (n₃, 0);
  else if (n₂ == n₃) // In a 2-class-leaf of size n₃ that is not a 1-class-leaf
    return (0, n₃);
  else // In a 2-set of size n₂, that is not a 2-class-leaf
    return (0, 0);
}

Figure 5: 1_2_leaf in pseudocode (implementation: zshg_component2() in zshg2.c,c)
float 1_2_leaves(δ, p, n)
{
    r := \frac{7}{2(1-p)\delta^2};
    a_1 := a_2 := 0;
    ℓ := r(3\log_e(\frac{3}{\delta}) + 6);
    for (i := 0; i < r; i := i + 1)
    {
        s := rand_index(n);
        x := rand_range(\frac{\delta}{2});
        (b_1, b_2) := 1_2_leaf(s, x, &ℓ);
        if (b_1 \neq 0)
            a_1 := a_1 + b_1;
        if (b_2 \neq 0)
            a_2 := a_2 + b_2;
    }
    return \left(\frac{a_1 n}{r}, \frac{a_2 n}{r}\right);
}

The number of 2-class-leaves bigger than \frac{3}{2} is at most \frac{\delta n}{2}. Ignoring the query bound for now, we bound the probability of the estimate being far off [19]:

$$\left\lceil \frac{c_2}{2} \right\rceil + c_1 - \frac{1}{2} - \frac{\delta n}{4} \leq E(\hat{c}) \leq \left\lceil \frac{c_2}{2} \right\rceil + c_1 + \frac{1}{2} + \frac{\delta n}{4}.$$  

$$P\left(\left| \hat{c} - E(\hat{c}) \right| > \frac{3\delta n}{4} \right) < \frac{Var(\hat{c})}{\left(\frac{3\delta n}{4}\right)^2} \leq \frac{28}{9r^2} = \frac{8}{9}(1 - p).$$

Let Q_i be the number of queries made by a call to 1_2_leaves in the algorithm in Figure 6.

Figure 6: Counting algorithm for 2-edge-connectivity in C-like pseudocode (implementation: zshg2_component() in zshg2.c.c)
\[
E(Q_i) \leq \sum_{j=1}^{\frac{3}{2}} \left( \frac{1}{j} - \frac{1}{(j+1)^2} \right) \frac{3j(j+1)}{2} + \frac{1}{(\frac{3}{2})^2} \frac{3^2(\frac{3}{2}) + 1}{2} = \\
= \sum_{j=1}^{\frac{3}{2}} \frac{6j + 3}{3j(j+1)} + \frac{6 + 3\delta}{16} < 3 \log_e \left( \frac{2}{\delta} \right) .
\]

\[
E(Q_i^2) \leq \sum_{j=1}^{\frac{3}{2}} \left( \frac{1}{j^2} - \frac{1}{(j+1)^2} \right) \left( \frac{3j(j+1)}{2} \right)^2 + \frac{1}{(\frac{3}{2})^2} \left( \frac{3^2(\frac{3}{2}) + 1}{2} \right)^2 = \\
= \sum_{j=1}^{\frac{3}{2} - 1} \left( \frac{9(j+1)^2}{4} - \frac{9j^2}{4} \right) + \frac{9(\frac{3}{2}) + 1}{2} = \frac{9}{3^2} + \frac{9}{2\delta} .
\]

\[
Var(Q_i) = E(Q_i^2) - E(Q_i)^2 \leq \frac{9}{3^2} + \frac{9}{2\delta} .
\]

Which we use to bound the probability of hitting the query bound [19].

\[
P \left( \sum_{i=0}^{r-1} Q_i > \ell \right) \leq P \left( \sum_{i=0}^{r-1} Q_i > r(E(Q_0) + 6) \right) < \frac{Var(Q_0)}{36r} < \frac{1}{9}(1-p) .
\]

\[
P \left( |\hat{C} - E(\hat{c})| > \frac{3\delta n}{4} \right) \leq P \left( |\hat{c} - E(\hat{c})| > \frac{3\delta n}{4} \right) + P \left( \sum_{i=0}^{r-1} Q_i > \ell \right) < 1 - p .
\]

q. e. d.

Lemma 8. The algorithm in Figure 6 has query complexity \(O(\frac{1}{(1-p)^2} \log_e \left( \frac{1}{\delta} \right))\).

Proof. This follows directly from the choice of \(\ell\). q. e. d.

For general \(k\), we use a generalization of Lemma 6 above.

The edge-demand of a set of nodes \(U \subseteq V\) is

\[
\Phi_k(U) := \max \left\{ 0, k - d(U), \sum_{W \subseteq U} \Phi_k(W) \right\} , \Phi_k(V) := \sum_{W \subseteq V} \Phi_k(W) .
\]

Lemma 9 (Naor et alii [13] for connected graphs, generalized by Marko [12]). Let \(G = (V, E)\) be a graph and \(k \in \mathbb{N}\). The distance of \(G\) to \(k\)-edge-connectivity is

\[
\frac{1}{|E|} \left[ \sum_{U} \Phi_k(U) \right] .
\]

To estimate \(\Phi_k(V)\) we can estimate the number of extreme sets \(U\), for which \(\Phi_k(U) \neq \sum_{W \subseteq U} \Phi(W)\), and their \(\Phi_k(U)\) and then get the estimate of \(\Phi_k(V)\) as the weighted sum.
For \( k = 3 \), there are 4 kinds of such \( U \): 3-class-leaves that are connected components (for these, \( \Phi_k = 3 \)), 3-class-leaves that are connected to the rest of the graph by a single bridge (for these, \( \Phi_k = 2 \)), 3-class-leaves that are connected to the rest of the graph by exactly two edges and are not inside the fourth kind (for these, \( \Phi_k = 1 \)), connected components that are 2-edge-connected and contain exactly 2 3-edge-leaves (for these, \( \Phi_k = 3 \)). Sets of the fourth kind contribute 3 each to \( \Phi_3 \), but for the algorithm it is simpler to make them contribute 1 and instead ignore the condition on sets of the third kind not being in the fourth kind. Let \( C_0 \) be the number of of 3-class-leaves that are connected components, let \( C_1 \) be the number of of 3-class-leaves that are connected to the rest of the graph by a single bridge, let \( C_2 \) be the number of 3-class-leaves that are connected to the rest of the graph by exactly two edges, let \( C_3 \) be the number of connected components that are 2-edge-connected and contain exactly 2 3-edge-leaves.

\[
\Phi_3(V) = 3C_0 + 2C_1 + 1C_2 + 1C_3 - k.
\]

To find out if a node \( s \) is in any such set of size at most \( r \), we use the algorithm in Figure 8. It returns the size of the containing sets: Figure 7 illustrates the possible cases: If \( s \) is in a 3-set larger than \( r \), the algorithm returns \((0, 0, 0, 0)\) to contribute nothing to any estimate of \( C_0 \),... If \( s \) is in a 3-set \( m \) of size at most \( r \) that is also a connected component, the algorithm returns \((|m|, 0, 0, 0)\) to contribute to the estimate of \( C_0 \) only. If \( s \) is in a 3-set \( m \) of size at most \( r \) that is connected to the rest of the graph by a bridge, the algorithm returns \((0, |m|, 0, 0)\) to contribute to the estimate of \( C_1 \) only. If \( s \) is in a 3-set \( m \) that is connected to another 3-set \( n \) by two edges, with \( m \cup n \) being a connected component of size at most \( r \), the algorithm returns \((0, 0, |m|, |m+n|)\) to contribute to the estimates of both \( C_2 \) and \( C_3 \). Otherwise, \( s \) is in a 3-set \( m \) of size at most \( r \) that is connected to the rest of the graph by two edges, the algorithm returns \((0, 0, |m|, 0)\) to contribute to the estimate of \( C_3 \) only. To decide if a node is in a set of the fourth kind, the algorithm uses the plant graph [16] of the connected component, which can be computed in time quadratic in the number of nodes in the connected component [20]. This results in total query complexity \( O(r^3) \)

![Figure 7: Cases for the algorithm in Figure 8](image)

We can use this algorithm to construct an algorithm for estimating the distance to 3-edge-connectivity similar to how we did so for 2-edge-connectivity above.

From our estimate of the number of components, we also obtain an estimate for the distance to eulerianity (Figure 9). The algorithm \texttt{1_sets_even} is a variant of \texttt{1_sets} that only counts components in which all nodes have even
\begin{verbatim}
int \times int \times int \times int 1_2_3_leaf(s, r)
{
    0 Do a depth-first search from \(s\) for up to \(r+1\) nodes.
        Let \(n_0\) be the number of nodes found. Let \(m_0\) be the set of nodes found.
    1 Do a depth-first search from \(s\) for up to \(r+1\) nodes,
        never traversing an edge of the search tree of 1 in the same direction.
        Let \(n_1\) be the number of nodes found.
    if(\(n_0 \leq r \land n_0 == n_1\)) // \(s\) is in a 2-connected connected component of size at most \(r\)
    {
        Compute the plan graph \(\Gamma\) of the subgraph induced by \(m_0\).
        if (\(\Gamma\) has exactly 2 nodes of degree 1 and no cycles)
            \(n_2 := n_0\);
        else
            \(n_2 := 0\);
    }
    else
        \(n_2 := 0\);

    2 For every edge traversed in the first depth-first search,
        invoke 2_set searching for up to \(r+1\) nodes in the graph with that edge omitted.
        Let \(m'\) be a smallest set found by 2_set.
    if (\(|m'| > r\)) // Not in a 3-set of size at most \(r\)
        return \((0,0,0,n_2)\);
    3 Do a depth-first search from \(s\) for up to \(|m'|\) nodes in the subgraph induced by \(m'\).
    4 Do depth-first search from \(s\) for up to \(|m'|\) nodes in the subgraph induced by \(m'\),
        never traversing an edge of the search tree of 3 in the same direction.
    5 For every edge traversed in the previous depth-first search,
        invoke 2_set searching for up to \(|m'|\) nodes in the subgraph induced by \(m'\) with that edge omitted.
        Let \(m\) be a smallest set found by 2_set.
    if (\(|m| \neq |m'|\)) // In a 3-set of size \(|m|\) that is not a 3-class-leaf
        return \((0,0,0,n_2)\);
    if(d(m) == 0) // In a connected component that is a 3-class-leaf.
        return \(|m|,0,0,n_2\);
    else if (d(m) == 1) // In a 3-class-leaf connected to the rest by a single bridge
        return \((0,|m|,0,n_2)\);
    else // d(m) == 2 // In a 3-class-leaf connected to the rest by two edges
        return \((0,0,|m|,n_2)\);
}
\end{verbatim}

Figure 8: 1_2_3_leaf in pseudocode

**Lemma 10.** With probability at least \(p\), value returned by the algorithm in Figure 9 is the distance of the graph to eulerianity with an error of at most \(\epsilon\).
```c
float euler_distance(ε, p, n) {
    δ := ε / 2;
    c := 1_sets_even(δ, 1-p);
    m := 2 / δ²(1+p);
    a := 0;

    for (i := 0; i < m; i := i + 1)
        if (d(s) odd)
            a := a + 1;

    u := am / m;
    e := c + u / 2;

    return (e / m);
}
```

Figure 9: Algorithm for estimating the distance to eulerianity in C-like pseudocode (the implementation resides in zshg_euler_distance() in zshg.c.c)

Proof. A graph is eulerian if it is connected and all nodes have even degree [7, 11, 9]. Let C be the number of connected components, in which all nodes have even degree. Let U be the number of nodes of odd degree. Then the minimum number of edge modifications necessary to make the graph connected is C + U / 2: In every component, the number of nodes of odd degree is even [10], so one can add a cycle that goes through all components (using preexisting edges within components that contain nodes of odd degree), requiring only that number of edges. It is not possible to do with fewer edge modifications: At every component that contains only nodes of even degree, we have to add at least 2 incident edges to a node, and at every node of odd degree in the graph, we have to add or remove an edge. Let q be the probability of the algorithm returning a value outside the error bound.

\[
E(c) = C, E(a_i) = U / n, E(a_i^2) = U / n, \text{Var}(u) \leq U / n.
\]

\[
E(u) = U, \text{Var}(u) = \frac{n^2}{m} \text{Var}(a_i) \leq \frac{n}{m} U \leq \frac{n^2}{m}.
\]

\[
P(|u - U| > εdn) < \frac{\text{Var}(u)}{ε^2 d^2 m} \leq \frac{1}{me^2 d^2} = \frac{1 - p}{2}.
\]

\[
q \leq P(|c - C| > \frac{εdn}{2}) + P(|u - U| > εdn) \leq 1 - \frac{1 - p}{2} + 1 - \frac{1 + p}{2} = 1 - p.
\]

q. e. d.

Lemma 11. The algorithm in Figure 9 has query complexity \(O\left(\frac{1}{(1-p)^2} \log_c \left(\frac{1}{3}\right)\right)\).
Proof. For 1 sets even we get that complexity like in Lemma 5 and the number of further queries is $m \in O \left( \frac{1}{p^2(1-p)} \right) \subseteq O \left( \frac{1}{(1-p)^2} \log_e \left( \frac{1}{\delta} \right) \right)$. q. e. d.

Theorem 2. There are estimates for the distance to connectivity, 2-edge-connectivity and eulerianity with complexity $O \left( \frac{1}{(1-p)^2} \log_e \left( \frac{1}{\delta} \right) \right)$. There is an estimate for the distance to 3-edge-connectivity with complexity $O \left( \frac{1}{(1-p)^2} \log_e \left( \frac{1}{\delta} \right) \right)$.

Proof. For the distance to connectivity this follows from Lemmata 4, 5. For the distance to 2-edge-connectivity, this follows from Lemmata 6, 7, 8. For the distance to eulerianity, this follows from Lemmata 10, 11. For the distance to 3-edge-connectivity, this can be proven using Lemma 9 and the algorithm in Figure 8. q. e. d.

6 Tolerant Connectivity Test

```c
bool zshg_tolerant(\epsilon_1, \epsilon_2, p, d, n)
{
    \delta := \frac{\epsilon_2 - \epsilon_1}{4} d;
    c := 1 sets (\delta, p, n) - 1;
    return (e \leq (\epsilon_1 + \epsilon_2)dn) / 4;
}
```

Figure 10: Tolerant connectivity testing algorithm in C-like pseudocode (the implementation resides in zshg_tolerant() in zshg.c.c)

Lemma 12. For any graph that is $\epsilon_1$-close to connectivity, the algorithm in Figure 7A returns true with probability at least $p$. For any graph that is $\epsilon_2$-far from connectivity, the algorithm in Figure 7A returns false with probability at least $p$.

Proof. Let $m = \frac{dn}{2}$ be the number of edges in the graph.

Case 1: The graph is $\epsilon_1$-close to connectivity, i.e. it consists of at most $\epsilon_1 m + 1$ connected components.

$$P \left( e > \frac{(\epsilon_1 + \epsilon_2)dn}{4} \right) = P \left( e > \frac{\epsilon_1 + \epsilon_2}{2} m \right) \leq$$

$$P \left( 1 \text{ sets}(\delta, p, n) > \frac{\epsilon_1 + \epsilon_2}{2} m + 1 \right) = P \left( 1 \text{ sets}(\delta, p, n) > (\epsilon_1 m + 1) + \delta n \right) \leq \left( 1 - p \right).$$

Case 2: The graph is $\epsilon_2$-far from connectivity, i.e. it consists of at least $\epsilon_2 m + 2$ connected components.
\[
P\left( e \leq \frac{(\epsilon_1 + \epsilon_2)d}{4} \right) = P\left( e \leq \frac{\epsilon_1 + \epsilon_2}{2} m \right) \leq P\left( 1_{\text{sets}}(\delta, p, n) \leq \frac{\epsilon_1 + \epsilon_2}{2} m + 1 \right) = P\left( 1_{\text{sets}}(\delta, p, n) \leq (\epsilon_2 m + 1) - \delta n \right) \leq (1 - p).
\]

Case 3: The graph is neither \( \epsilon_1 \)-close to nor \( \epsilon_2 \)-far from connectivity. We don’t care. q. e. d.

In a similar way, we can obtain tolerant testers for 2-edge-connectivity, 3-edge-connectivity and eulerianity.

**Theorem 3.** There are tolerant testers for connectivity, 2-edge-connectivity and eulerianity with complexity \( O\left( \frac{1}{(1-p)(\epsilon_2 - \epsilon_1)^2} \log_e\left( \frac{1}{(\epsilon_2 - \epsilon_1)d} \right) \right) \). There is a tolerant tester for 3-edge-connectivity with complexity \( O\left( \frac{1}{(1-p)(\epsilon_2 - \epsilon_1)^3} \log_e\left( \frac{1}{(\epsilon_2 - \epsilon_1)d} \right) \right) \).

**Proof.** For connectivity this follows from Lemmata 4, 5, 12. The proofs for the others are similar. q. e. d.

### 7 Remarks

While the results on correctness also hold for multigraphs, the results on query complexity don’t.

All of our algorithms can be parallelized easily and we did so in the implementations. The parallel versions offer an advantage when multiple pending queries can be answered more efficiently, e.g. in the case of large (too big to fit into RAM) graphs stored on an SSD (current SSDs typically achieve maximum throughput for random reads at about 16 simultaneous pending reads) or in the case of the queries being processed by a remote server on a network.

### References

[1] Petra Berenbrink, Bruce Krayenhoff, and Frederik Mallmann-Trenn. Estimating the number of connected components in sublinear time. Information Processing Letters, 114(11):639 – 642, 2014.

[2] Jacobo Bernoulli. Positiones arithmeticae de seriebus infinitis Earumque summa finita. Dissertation, 1689.

[3] Henry R. Brahana. A proof of Petersen’s theorem. Annals of Mathematics, 19(1):59–63, 1917.

[4] Andrea Campagna, Alan Guo, and Ronitt Rubinfeld. Local Reconstruction and Tolerant Testers for Connectivity and Diameter, pages 411–424. Springer, Berlin, Heidelberg, 2013.

[5] Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. SIAM Journal on Computing, 34(6):1370–1379, 2005.
[6] Kapali P. Eswaran and Robert E. Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5:653–665, 1976.

[7] Leonhard Eulero. Solutio problematis ad geometriam situs pertinetis. *Commentarii academiae scientiarum Petropolitanae*, 8:128–140, 1741.

[8] Oded Goldreich and Dana Ron. Property Testing in Bounded Degree Graphs. *Algorithmica*, 32(2):302–343, Feb 2002.

[9] Carl Hierholzer. Ueber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen*, 6:30–32, 1873.

[10] Dénes König. *Theorie der endlichen und unendlichen Graphen*. Chelsea, 1935.

[11] Johann B. Listing. Vorstudien zur Topologie. Vandenhoeck und Ruprecht, 1847.

[12] Sharon Marko. Distance approximation in bounded-degree and general sparse graphs. Master’s thesis, 2005.

[13] Dalit Naor, Dan Gusfield, and Charles Martel. A fast algorithm for optimally increasing the edge connectivity. *SIAM Journal on Computing*, 26:1139–1165, 08 1997.

[14] Michal Parnas and Dana Ron. Testing the diameter of graphs. *Random Structures & Algorithms*, 20(2):165–183, 2002.

[15] Dana Ron. Algorithmic and Analysis Techniques in Property Testing. *Foundations and Trends in Theoretical Computer Science*, 5(2):73–205, February 2010.

[16] Efim A. Диниц, Alexander В. Карзаанов, and Michael В. Ломоносов. О структуре системы минимальных реберных разрезов графа. In А.А. Фридман, editor, *Исследования по Дискретной Оптимизации*, pages 290–306. Наука, Москва, 1976.

[17] Renatus F. Slusius. *Mesolabum*. Apud Guilielmum Henricum Street, Serenissimae Sue Celsitudinis Typographum, 1668.

[18] Richard M. Stallman, Joshua Gay, and Lawrence Lessig. *Free Software, Free Society: Selected Essays of Richard M. Stallman*. GNU Press, 2002.

[19] Pafnuti L. Tchébychef. Des valeurs moyennes. *Journal de mathématiques pures et appliquées 2e série*, 12:177–184, 1867.

[20] Alexander В. Карзаанов and Евгений А. Тимофеев. Эффективный алгоритм нахождения всех минимальных реберных разрезов неориентированного графа. *Кибернетика*, (2):8–12, 1986.