A Method to Prove Query Lower Bounds

Jagadish M
Dept. of Computer Science and Engg.
Indian Institute of Technology Bombay
Mumbai 400076, India
jagadish@cse.iitb.ac.in

Sridhar Iyer
Dept. of Computer Science and Engg.
Indian Institute of Technology Bombay
Mumbai 400076, India
sri@iitb.ac.in

ABSTRACT

The query-model or decision-tree model is a computational model in which the algorithm has to solve a given problem by making a sequence of queries which have 'Yes' or 'No' answers. A large class of algorithms can be described on this model and we can also prove non-trivial lower bounds for many problems on this model.

Many lower bounds on the query-model are proved using a technique called adversary argument. In CS courses, a common example used to illustrate the adversary argument is the following problem: Suppose there is an unweighted graph G with n vertices represented by an adjacency matrix. We want to test if the graph is connected. How many entries in the adjacency matrix do we have to probe in order to test if the graph has this property (property being 'connectivity')? Each probe is considered as a query.

Since the adjacency matrix has only n^2 entries, $O(n^2)$ queries are sufficient. It is also known that $\Omega(n^2)$ queries are necessary. Proving this lower bound is more difficult and is done using the adversary argument.

In literature, we find that lower bound proofs of this problem rely too much on 'connectivity' property and do not generalize well. When the property being tested is changed, the proof changes significantly. Our contribution is a method that gives a systematic way of proving lower bounds for problems involving testing of many graph-properties. We did a pilot experiment and found that students were able to understand and apply our method.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education

General Terms

Algorithms

ITiCSE'14, June 21–25, 2014, Uppsala, Sweden.

http://dx.doi.org/10.1145/2591708.2591736 .

Keywords

Query complexity, Lower bounds, Adversary arguments

1. INTRODUCTION

A major focus of computer science is on design and analysis of provably efficient algorithms. Once we have designed a correct algorithm for a problem, a natural question to ask is: 'Is this the best possible algorithm for this problem?'. The only way to know the answer for certain is to *prove* that no better algorithm exists. On the Turing-machine computational model, this question is very difficult to answer and very little is known. So researchers have been studying the question of proving the optimality of an algorithm on simpler models of computation. One such model that is widely studied is the 'query model' (also known as decision-tree model).

Query Model. In this model, the algorithm is not given the input directly. The algorithm has to output the answer by asking queries specified by the problem. The efficiency of the algorithm is measured by the number of queries it takes in the worst case. The query-model is popular because a large class of algorithms can be implemented on this model. Hence, proving optimality of an algorithm on the query model proves its optimality within a large class of algorithms. For example, consider the problem of sorting n numbers in an array. All comparison-based algorithms like Quicksort, Mergesort, Heapsort, etc. can be implemented as query algorithms where a query corresponds to a comparison between a pair of elements from the array.

Query Complexity. In the query model, we are concerned only with the number of queries asked by the algorithm and any other computation it may do is irrelevant. The *cost* of an algorithm that solves a problem P is the number of queries the algorithm takes in the worst case to solve P. The cost of an algorithm can usually be expressed as a function of the input size n. We say algorithm A is better than algorithm B if there exists a constant c_o such that the cost of algorithm A is less than B for every problem instance of size $n > c_0$. The *query complexity* of a problem P is the cost of the best algorithm that solves P. We denote the query complexity of the problem of size n by $T(n)$. Query complexity is a property of the problem and cost is a property of an algorithm.

Given below are four examples of query-model problems. We will refer to all of them later, but our method is applicable only to problems similar to the CONNECTIVITY problem.

Formal descriptions of the first three problems can be found in [3], [2] and [5], respectively.

Problem E1. SORTING. Given an array A of n numbers, sort the elements in the array using only comparison queries. A comparison query is of the type, 'Is $A[i]$ smaller or bigger than $A[j]$?'.

Problem E2. ELEMENT DISTINCTNESS. Given an array A of n numbers, find if all the numbers are distinct, using only comparison and equality queries. An equality query is of the type, 'Is $A[i]$ equal to $A[j]$?'.

Problem E3. 3-SUM. Given an array A of n numbers, find if there exists three numbers in the array, say a, b and c, such that $a+b+c=0$, using only linear-equation queries. A linear-equation query is of the type, 'Is $3A[1]+2A[3]-A[6] > 0$?'.

Problem E4. CONNECTIVITY. Suppose there is a graph $G=(V,E)$ with n vertices. We know the vertex set V of the graph but not the edge set E. Vertices are labelled from 1 to n. For any two vertices, a and b with $a, b \in V$, we are allowed to ask the following query: 'Is (a,b) an edge in G?'. If G is given by an adjacency matrix, this query is equivalent to probing the entry at the ath row and bth column of the adjacency matrix of G. What is the query complexity of determining if the graph is connected?

If the objective of the problem is to determine whether an input satisfies a given property or not, then we call such a problem as 'property-testing' problem. A property-testing problem has exactly two possible answers: 'True' or 'False'. For example, problems E2, E3 and E4 are property-testing problems.

Tight lower bounds.[1] We call a lower bound tight if it asymptotically matches an upper bound of $T(n)$. We are interested in obtaining tight asymptotic bounds on $T(n)$ for problems involving testing of graph-properties (similar to Prob. E4).

Overview. There are four main techniques for proving query lower bounds. The four problems E1-E4 are generally used as examples to illustrate each technique (Sec. 2). However, our focus is only on proving lower bounds for testing graph properties. The known methods for proving lower bounds for this kind of problems are ad-hoc in nature and rely on problem-specific observations. Our main contribution is a method that gives a more systematic way of proving lower bounds (Sec. 5). In Sec. 6, we show that our approach works for many problems within the scope. We did a pilot experiment and observed that students are able to understand and apply our method (Sec. 7).

2. RELATED WORK

We review known methods for proving query lower bounds and provide context for our work.

Information-theoretic proof. An explanation of this method can be found in Chapter 8 of [3]. The main assertion of the proof is the following:

Fact. If a problem P has M possible outputs and the input to the problem can be accessed only via 'Yes/No' queries, then $\log_2 M$ is a query lower bound for P.

[1]Most of the terminology we use is standard. Definitions of algorithmic terms and graph theoretic terms can be found in [8] and [10], respectively.

In other words, $\log_2 M$ is the minimum number of queries any correct algorithm must ask, in the worst case. For example, in the sorting problem, if the input consists of n numbers, then there are $n!$ possible outputs. Each output corresponds to a different ordering of n numbers. So we can claim that the lower bound for sorting problem is $\log_2 n! = \Omega(n \log n)$.

However, this method is not useful for property-testing problems. In a property-testing problem, there are only two outputs: 'True' or 'False'. The information-theoretic proof only gives a trivial lower bound of one ($\log_2 2 = 1$).

Algebraic Methods. In 1983, Ben-Or gave an algebraic technique that proves a tight $\Omega(n \log n)$ lower bound for ELEMENT-DISTINCTNESS problem. His technique, in one stroke, proves non-trivial lower bounds for twelve different problems [2]. But problems E3 and E4 are not amenable to this technique.

Reduction. Reduction is another way of proving lower bounds. At undergrad-level, reduction is often taught in the context of NP-hardness. If we want to show that problem H is NP-hard, we do so by proving that some known NP-hard problem (like SAT) reduces to H in polynomial time. Query lower bounds can also be proved in a similar vein. Suppose we know that a problem R has query complexity of $\Omega(g(n))$ on a certain computation model, we can show that another problem S has query complexity $\Omega(g(n))$ on the same model by reducing problem R to S in $o(g(n))$ time. For example, we know that finding the convex hull of a set of n points takes $\Omega(n \log n)$ operations because the sorting problem reduces to it (Prob. 33.3-2 in [3]).

The 3-SUM problem is known to have a query complexity of $\Theta(n^2)$ [5]. The lower bound proof of 3-SUM does not result in any general technique. But the strength of 3-SUM lies in the fact that this problem is an excellent candidate problem for proving lower bounds by reductions. Several problems in computational geometry have been shown to be as hard as 3-SUM [6]. But a lower bound for Prob. E4 cannot be proved by reduction either.

Adversary Arguments. What can we do if none of the above three methods work? We take recourse to proving by 'first principles', commonly known as proving by *adversary argument*. Recall that a lower bound of $g(n)$ for a problem P means that any *correct* algorithm must make at least $g(n)$ queries to solve P. So if we show that any algorithm that makes strictly less than $g(n)$ queries must be incorrect, then we have proved a lower bound of $\Omega(g(n))$ for the problem. Imagine that the queries asked by the algorithm are answered by an adversary. The adversary's objective is to maximize the number of queries that the algorithm asks and the algorithm's objective is to minimize the number of queries. If an adversary can force any correct algorithm to ask at least $g(n)$ queries, then $g(n)$ is a lower bound on the query complexity of the problem. It is necessary to understand the format of adversary arguments in order to follow the proofs in this paper. Introduction to adversary method can be found in [4] or [9]. Two proofs that give *exact* lower bounds for Prob. E4 can be found in [4] and [1]. Both the proofs are based on adversary arguments. However, these proofs do not generalize easily to other graph properties. In the next section, we give a different proof for this problem using an adversary argument. Though our proof gives

only an asymptotic tight lower bound, it generalizes easily to other graph properties.

3. ADVERSARY ARGUMENT REVISITED

Terminology

Notation. Throughout the text, the notation $G = (V, E)$ refers to an undirected unweighted graph G with n vertices, where V is the vertex set and E is the edge set. We may assume that the vertices are labeled from 1 to n. We denote a complete graph having r vertices by K_r.
Definition. If there is no edge present between a pair of vertices u and v in a graph, we say (u, v) is a *non-edge* in the graph.

3.1 Testing Connectivity

Problem E4. We are given access to a graph $G = (V, E)$ via queries of the form 'Is (a, b) an edge in G?'. We know the vertex set V but not the edge set. Prove that it takes at least $\Omega(n^2)$ queries in the worst case to determine if G is connected.

Proof: We prove that any correct algorithm must take at least $n^2/4$ queries. For contradiction's sake, let us assume that there is a correct algorithm $\mathcal{A}$ that makes less than $n^2/4$ queries. We first describe the adversary's strategy for answering queries asked by the algorithm and then give the analysis.

Let G_1 be the graph made of two complete subgraphs A and B, such that A has nodes labelled from 1 to $n/2$ and B has nodes labelled from $n/2+1$ to n (Fig. 1 (a)). Note that graph G_1 has the same vertex set V as G.

Adversary's Strategy. The adversary first picks the graph G_1 tentatively as input (in its mind) and answers the queries posed by the algorithm as follows:

> When the algorithm asks 'Is (a, b) an edge in G?', the adversary says 'Yes' if (a, b) is an edge in G_1 and 'No' if (a, b) is a non-edge in G_1.

The adversary also keeps track of all the queries asked by the algorithm.

Analysis. Consider the moment when all the queries are made and $\mathcal{A}$ has declared whether G is connected or not. By our assumption, $\mathcal{A}$ has made less than $n^2/4$ queries. Since there are $n^2/4$ non-edges in G_1, there exists some pair of vertices (say (i, j)) such that:

- Pair (i, j) is a non-edge in G_1.
- $\mathcal{A}$ did not ask the query 'Is (i, j) an edge in G?'

Let G_2 be the graph obtained by replacing the non-edge (i, j) in G_1 by an edge (Fig. 1 (b)). G_1 is not connected but G_2 is connected. But both the graphs are consistent with all the answers the adversary has given. The adversary can prove that algorithm $\mathcal{A}$ is wrong as follows:

If the algorithm outputs 'True. G is connected', the adversary 'reveals' that $G = G_1$ and shows that the algorithm is wrong. If the algorithm outputs 'False. G is not connected', the adversary reveals that $G = G_2$ and again proves the algorithm wrong.

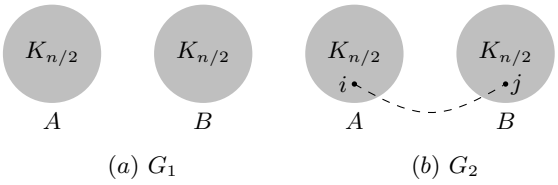

Figure 1. (a) G_1 is the graph the adversary tentatively keeps as input in its mind. Every pair of vertices (p, q) with $p \in A$ and $q \in B$ in a non-edge in G_1. So G_1 has $n^2/4$ non-edges. (b) G_2 is the graph obtained by replacing the non-edge (i, j) in G_1 with an edge. G_2 is connected but G_1 is not. Since (i, j) was not queried by the algorithm, both G_1 and G_2 could have been possible inputs to the problem.

So regardless of what the algorithm outputs, the adversary can always contradict the algorithm. Hence, every correct algorithm must make at least $n^2/4$ queries.

□

4. SCOPE OF PROBLEMS

Our method is applicable to problems of the following kind.

Generic-Problem. There is a graph $G = (V, E)$ to which we do not have direct access. We know the vertex set but not the edge set. There are n vertices in G and we may assume that they are labelled from 1 to n. We are allowed to ask queries of the type 'Is (a, b) an edge in the graph G?', where $a, b \in V$. The problem is to find if G has a given property $\mathcal{P}$ by asking such queries.

4.1 Our approach

In order to prove a lower bound for the Generic-Problem, when property $\mathcal{P}$ is specified explicitly, we do the following:

- Construct a critical graph G for property $\mathcal{P}$ which has many non-edges. We will define a critical graph shortly.
- Apply Theorem 5.1 which says that existence of a critical graph with many non-edges implies a lower bound for the problem.

5. MAIN THEOREM

Definition. Given a property $\mathcal{P}$, a graph G_c is said to be *critical* with respect to the property if the following two conditions are met.
(C1) G_c does not have the property $\mathcal{P}$.
(C2) Replacing any non-edge in G_c by an edge endows the graph with property $\mathcal{P}$.

For example, if the property is 'Connectivity', then the graph shown in Fig. 1 (a) is an example of a critical graph.

We will now prove our main result using ideas very similar to the proof of Connectivity problem (Sec. 3.1).

Theorem 5.1. *Suppose G_c is a critical graph for the property $\mathcal{P}$. If G_c has n vertices and x non-edges, then x is a query lower bound for the* Generic-Problem.

Proof: We show that any correct algorithm must make at least x queries in the worst case. We prove by contradiction. Assume that there is some algorithm $\mathcal{A}$ that solves the Generic-Problem by making strictly less than x queries.

We first give an adversary's strategy for answering queries by $\mathcal{A}$ and then give the analysis.

Adversary's Strategy. The adversary first constructs the graph G_c (in its mind) and answers the queries posed by the algorithm as follows: When the algorithm asks 'Is (a, b) an edge in G?', the adversary says 'Yes' if (a, b) is an edge in G_c and 'No' if (a, b) is a non-edge in G_c.

Analysis. Consider the moment when all the queries are made and $\mathcal{A}$ has declared whether G has the property $\mathcal{P}$ or not. By our assumption, $\mathcal{A}$ has made less than x queries. Since there are x non-edges in G_c, there exists some pair of vertices (say (i, j)) such that:

- Pair (i, j) is a non-edge in G_c.
- $\mathcal{A}$ did not ask the query 'Is (i, j) an edge in G?'

By our definition of critical graph, we have the following:

- Graph G_c does not have property $\mathcal{P}$.
- Suppose G'_c is a graph which is same as G_c except that the non-edge (i, j) in G_c is replaced by an edge in G'_c. So G'_c has exactly one more edge than G_c. More importantly, G'_c has the property $\mathcal{P}$, since G_c was a critical graph.

Either G_c or G'_c could have been the input graph for the problem since they are consistent with all the answers the adversary has given. But one graph has the property and the other one does not. Hence, whatever answer the algorithm outputs, the adversary can prove the algorithm wrong. Hence, any algorithm that makes less than x queries must be incorrect. □

Now we turn to the applications of this theorem.

6. APPLICATIONS

We prove lower bounds for many common graph properties by constructing a critical graph for each property. It turns out that all the problems we consider have a lower bound of $\Omega(n^2)$, which proves that the bounds are asymptotically tight.

For each problem, we describe how to construct a critical graph with $\Omega(n^2)$ number of non-edges. The lower bound follows from Theorem 5.1.

Here is a handy fact about the number of non-edges in a graph.

Lemma 6.1. *A graph with n vertices and m edges has x non-edges, where $x = \binom{n}{2} - m$.*

Proof: There are $\binom{n}{2}$ pairs of vertices in a graph. Between every pair of vertices, either there is an edge or a non-edge. If m is the number of edges and x is the number of non-edges we have $x + m = \binom{n}{2}$. □

6.1 Triangle Detection

A graph G is said to have a *triangle* if K_3 is a subgraph of G.

Problem P1. Given access to $G = (V, E)$ via queries of the form 'Is (a, b) an edge in G?' determine if G has a triangle.

Construction of G_c. A rooted star tree with the root node as 1 (Fig.2). In other words, G_c is a graph where we connect node 1 to every other node. There are exactly $m = n - 1$ edges in G_c.

It is easy to verify that G_c satisfies both the conditions of a critical graph.

(C1) G_c does not have triangle.

(C2) Replacing any non-edge in G_c, induces a triangle involving node 1.

No. of non-edges in G_c: $x = \binom{n}{2} - m = \Omega(n^2)$.

By Theorem 5.1, any algorithm must make at least $\Omega(n^2)$ queries. Since this step is same for all problems, in subsequent examples, we only describe the construction of the critical graph.

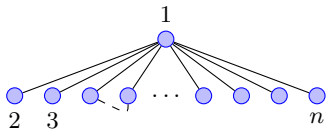

Figure 2. A critical graph for 'has a triangle' property. Node 1 is connected to every other node. The non-edges are between nodes $2, ...n$.

6.2 Hamiltonian Path

Problem P3. Given access to a graph $G = (V, E)$ via queries of the form 'Is (a, b) an edge in G?' determine if G has a Hamiltonian path.

Construction of G_c. We build two complete subgraphs A and B of equal size such that A has nodes labelled from 1 to $n/2$ and B has nodes labelled from $n/2 + 1$ to n (Fig. 3).

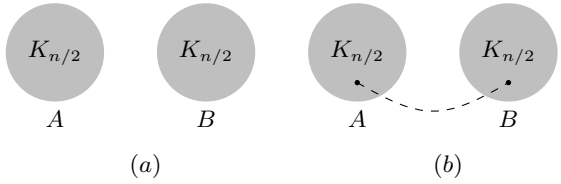

Figure 3. (a) A critical graph G_c for Hamiltonian property. G_c does not have a Hamiltonian path since it is disconnected. (b) Replacing any non-edge by an edge induces a Hamiltonian path.

6.3 Perfect Matching

Problem P3. A graph G is said to have a *perfect matching* if there exists a pairing of all nodes in G, such that every node is contained in exactly one pair and each pair has an edge between them. A necessary (but not sufficient) condition for G to have a perfect matching is that n must be an even number. Given access to $G = (V, E)$ via queries of the form 'Is (a, b) an edge in G?' determine if G has a perfect matching. Assume that n is even.

Construction of G_c. Since n is an even number, assume that $n = 2s$. If s is an odd number, we construct two complete subgraphs A and B such that A has nodes from the set $\{1, .., s\}$ and B has nodes from the set $\{s + 1, .., n\}$. If s is an even number, we construct two complete graphs A and B such that A has nodes in the set $\{1, .., s - 1\}$ and B

has nodes in the set $\{s, .., n\}$ (Fig. 4). Basically, we want to ensure that both A and B have odd number of nodes.

No. of non-edges is s^2, when s is odd and $s^2 - 1$ when s is even. Hence, number of non-edges $x \approx n^2/4$.

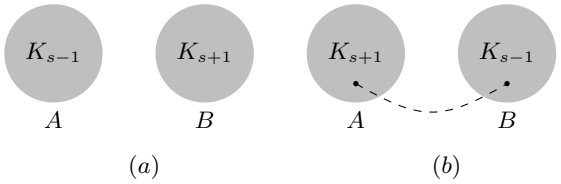

Figure 4. (a) Critical graph G_c for 'perfect matching' property when s is even. Graph G_c does not have a perfect matching since A and B have odd number of vertices. (b) If we replace any non-edge by a edge then it is easy to verify that the graph obtained has a perfect matching.

6.4 Non-Bipartite Detection

Problem P4. Given access to $G = (V, E)$ via queries of the form 'Is (a, b) an edge in G?' determine if G is non-bipartite.

Construction of G_c. G_c is a complete bipartite graph with two partitions A and B. where partition A has nodes labelled from 1 to $n/2$ and B has nodes labelled from $n/2+1$ to n. So G_c has $m = n^2/4$ edges (Fig. 5). No. of non-edges in G_c: $x = \binom{n}{2} - m \approx n^2/4$.

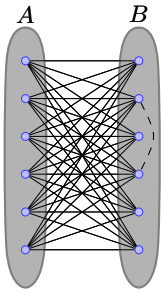

Figure 5. A critical graph for the property of 'Non-bipartiteness'. G_c is bipartite but adding replacing any non-edge by an edge makes in non-bipartite.

6.5 Cycle Detection

Problem P5. Given access to $G = (V, E)$ via queries of the form 'Is (a, b) an edge in G?' determine if G has a cycle.

Construction of G_c. A critical graph for 'has a cycle' property is a path containing all nodes. Hence G_c has exactly $n - 1$ edges (Fig. 6). No. of non-edges: $x = \Omega(n^2)$.

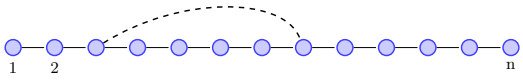

Figure 6. Critical graph G_c for 'has a cycle' property. Replacing any non-edge by an edge induces a cycle.

6.6 Degree-Three node

Problem P6. Given access to $G = (V, E)$ via queries of the form 'Is (a, b) an edge in G?' determine if G has a node whose degree is three.

Construction of G_c. A critical graph for property 'has a degree-3 node' is a cycle containing all nodes. Hence G_c has exactly n edges (Fig. 7). No. of non-edges: $x = \binom{n}{2} - n = \Omega(n^2)$.

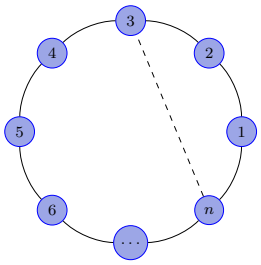

Figure 7. Critical graph G_c for 'has a degree-3 node' property. G_c has all nodes of degree two but replacing any non-edge by an edge gives a degree-3 node.

The method also works for other graph properties like planarity, two-connectedness, etc. but we have omitted these examples.

7. TEACHABILITY

We did a pilot experiment with five CS graduate students to see if they could apply our method to derive query lower bounds. We report our findings in this section.

Design. One of the authors of the paper had one-on-one interview-type session with each student. Prior to the start of the session, we made sure that the student understood the basic concepts and definitions related to query computational model. A session lasted for 2-4 hours and was divided into three phases. In the first phase, the students understood the format of adversary arguments. In the second phase, the student attempted to solve the CONNECTIVITY problem (Prob. E4) for one hour. Our objective was to allow the students to try the problem on their own so that we could identify the common approaches taken to solve the problem. The student was encouraged to think aloud during this time. If he persisted in a wrong approach for more than ten minutes, the author alerted the student to the mistake and let him continue. At the end of one hour, regardless of how the student performed, the author gave the solution to the problem along with an explanation of the method as described in Sections 3-5. In the third phase, the student attempted to derive lower bounds for the list of problems given in Sec. 6, without any help. We describe observations made in each phase below.

7.1 Phase I: Adversary Argument Explained

In the first phase, the students got familiar with the adversary argument. Two students who said they were familiar with adversary arguments were quizzed on the topic with a couple of questions. The other three students found it easy to follow the adversary argument through examples like 'n-Card Monte' and '20-questions', given in [4] and [9], respectively. Some students took more time than others to understand the format of the proof, which accounted for the variability in the duration the sessions.

7.2 Phase II: Teaching

We list common failed approaches taken while solving the CONNECTIVITY problem.

Failed approach F_1. Students find it hard to let go of the algorithm and think like an adversary. They often prove lower bounds assuming that the algorithm works in a certain way. For example, four out of five students gave the answer along the following lines: "Every vertex has $n-1$ possible incident edges. When the algorithm asks $n-1$ queries for a vertex, the adversary answers with a 'Yes' only for the last incident edge." The flaw in this reasoning is the assumption made about the algorithm's behavior. The algorithm need not ask queries in an orderly manner. The lower bound has to work for *any* algorithm, not just the one that probes vertex-by-vertex.

Failed approach F_2. All the students tried for exact bounds which is harder than proving asymptotic bounds. Even when the students were reminded that only asymptotic lower bound is sufficient, they could not figure out a way to make use of the relaxed constraint.

Failed approach F_3. Three students came up with the correct adversary strategy, but failed to do the analysis. The strategy was: The adversary always says 'No' unless saying so *definitely* disconnects the graph. This strategy is intuitive and is also correct, but unfortunately none of them were not able to prove that it works. The correct analysis of this strategy is given in [1].

7.3 Phase III: Testing

In Table 1, we give the time taken by each student to construct the correct critical graph. Some students gave different (correct) critical graphs than the ones we have given. For example, for 'Triangle' property two students gave complete bipartite graph as the answer. Similarly, for 'Cycle' property all the students gave 'any spanning tree' as the answer. As it can be seen from Table 1, most students were able to construct critical graphs quickly. We observed that three students who were not familiar with adversary arguments did equally well as the other two students. We did not anticipate this but it seems reasonable in hindsight. After all, the main reason why our method is helpful is because it shifts the focus from designing an adversary strategy to finding one critical graph. So prior knowledge of adversary arguments did not matter as much. In the words of a student, "Instead of thinking about how the queries must be answered, I just have to look for the right graph, which seems easier to do".

Phase	Topic	Students				
		S_1	S_2	S_3	S_4	S_5
I	Adv.	Yes	Yes	No	No	No
II	E4	F_2	$F_{1,2,3}$	$F_{1,2}$	$F_{1,2,3}$	$F_{1,2,3}$
III	P1	8m	10m	-	9m	-
	P2	5m	5m	9m	6m	6m
	P3	8m	-	-	10m	9m
	P4	1m	2m	4m	2m	2m
	P5	1m	2m	2m	1m	2m
	P6	1m	1m	1m	1m	1m

Table 1. Summary of data from the experiment. Phase I: 'Yes' means that the student already knew the adversary method. Phase II: Subscripts in F refer to the failed approaches taken by the student as discussed in Sec. 7.2. Phase III: Numbers give the time taken by the student to answer. A - against a problem means that the student was not able to answer that problem within 15 minutes.

8. CONCLUDING REMARKS

Remark 1. An alternate approach to proving lower bounds for graph-properties can be found in [7]. However, this proof method is beyond the scope of most CS courses since it relies on deep ideas in topology.

Remark 2. There is a lacuna in most traditional textbooks when discussing lower bounds. We suspect that this is because much work on this topic has been done only in the last two decades. But lower bounds as a topic in complexity theory has gained a lot of importance in the last few years. In support of this claim, we would like to point out that the recent book by Arora and Barak titled 'Computational Complexity: A Modern Approach' [1] has one-third of the book devoted exclusively to lower bounds!

9. REFERENCES

[1] S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[2] M. Ben Or. Lower bounds for algebraic computation trees. In *Proceedings of the fifteenth annual ACM Symposium on Theory of Computing (STOC)*, pages 80–86. ACM, 1983.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[4] J. Erickson. http://goo.gl/0z36dz. Last Accessed Jan 10, 2014.

[5] J. Erickson et al. Lower bounds for linear satisfiability problems. *Chicago Journal of Theoretical Computer Science*, 8:1999, 1999.

[6] A. Gajentaan and M. H. Overmars. On a class of $o(n^2)$ problems in computational geometry. *Computational geometry*, 5(3):165–185, 1995.

[7] J. Kahn, M. Saks, and D. Sturtevant. A topological approach to evasiveness. *Combinatorica*, 4(4):297–306, 1984.

[8] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison Wesley, second edition, 2006.

[9] Sally A. Goldman and Kenneth J. Goldman. Adversary Lower Bound Technique. http://goldman.cse.wustl.edu/crc2007/handouts/adv-lb.pdf. Last Accessed Jan 10, 2014.

[10] D. B. West. *Introduction to Graph Theory*. Prentice Hall, 2 edition, September 2000.