Relation between DAG-like query complexity and Certificate Complexity

Author: Jeff Smith
Supervisor: Marc Vinyals

October 29, 2024

Contents

1 Introduction

1.1 Why Study Computational Complexity

Computational complexity studies how much computation is necessary and sufficient to perform computational tasks [BW02]. This is important in cryptography, where we need quickly verifiable problems that are hard to compute directly. In this field, we aim to identify how much can / cannot be computed and how much memory and time it takes to compute problems [Sip97]. This allows us to compare different programs with each other to build more effective solutions and model the efficiencies of algorithms under various circumstances.

1.2 Weak Models

Finding tight bounds for standard computation models like Turing machines can be very challenging. Sometimes, we consider a stronger model like oracles to find a lower bound on the complexity. There are interesting results here, like the existence of oracles where $P = NP$ and others where $P \neq NP$ [BGS75]. Doing this can give us an idea of problems conditional on things that cannot be currently simulated. Other times, we try to consider a weaker model. We do this because it gives us a different and often more structured way of thinking about a problem. It is regularly easier to prove more substantial claims in a weaker model. For example, a big topic that involves weak models is the P vs NP problem. We follow an idea from [CR79]. We know that if P = NP, then CoNP = NP. We also notice that TAUT is a known CoNP-complete problem. If we can prove that there are no polynomial certificates for TAUT, we will prove $P \neq NP$.
It is often easier to do this in a weaker model where we restrict the certificates to weaker certificates. If we can do this in a weaker system, it may provide a helpful method to prove in a more general model.

1.3 Query Complexity

Query complexity is a subfield of computational complexity. In this, we aim to compute a boolean function $f : \{0,1\}^n \to \{0,1\}$ using queries to the input. A query asks for the value of the bit x_i and the answer to this value. The bit we query may depend on all the previous queries. The final algorithm can be described by a binary tree named a decision tree [BW02]. In query complexity, we study how deep decision trees are that compute different functions.
We also look at something called the certificate complexity. The preliminaries formally define this, but a non-deterministic decision tree can simulate it. This is the same as having an oracle telling us which variable to query for minimum depth.

1.4 Forgetting

Forgetting is an old but interesting complexity measure. Instead of measuring the complexity as the amount of time or space it takes to complete a program, we measure it as the amount of space we concurrently need at any step in the program. Then, we allow ourselves to remove unnecessary data to take advantage of this difference in complexity. This paper focuses on dag-like query complexity, a forgetting measure for query complexity. This will be better defined in the preliminaries. Dag-like query complexity is still being researched; some recent papers around forgetting include [AD08], [Gar+20], and [Fil+24].

1.5 Interesting Recent Developments

There are many interesting recent developments in query complexity. For example, in [Fil+24], they focused on showing a result of a different proof system called Hitting. However, they also had a section on dag-like query complexity.
In [Amb+17], they improved the separation between different query complexity measures. In this, they also refuted a nearly 30-year-old conjecture.
In [Hua19], they proved the longstanding sensitivity conjecture.

1.6 This Paper's Contributions and Goals

In this paper, we construct useful characterizations to interpret more easily when we find a common structure of boolean functions. We also set out to prove boolean functions can be queried 'efficiently,' which is defined precisely below. We proved this for a type of function we define and call 'independent functions,' and we set out to prove this for monotone boolean functions. We plan to use this to generalize it to arbitrary boolean functions. This is a known open problem in query complexity.

2 Preliminaries

Definition (Boolean function). A boolean function on n variables is a function $f : \{0,1\}^n \to \{0,1\}$.

Definition (Decision Tree). This definition comes from [Juk14]. A decision tree is an algorithm for computing a boolean function. To define it, we first define its tree; then, we define how it interacts with the tree as an algorithm. A variable labels each node of the tree, and the branches from that node are labelled by the variable's possible values.. The leaves are labelled by a possible function output. The algorithm starts at the root and works down the tree. It checks the function's variable of the node it is looking at and follows the branch the variable is in the input. It does this until it reaches the leaf and returns the leaf's value.
The depth of a decision tree is the depth of the underlying tree.

Definition (Deterministic query complexity). The decision tree computes a function f if simulating it for any input to f would result in the same output as f outputs. We define the deterministic query complexity of a function f, $[D(f)]$ to be the smallest depth of a decision tree that computes f [Juk14].

Definition (Certificate complexity). A boolean function can also be computed by a non-deterministic decision tree. A non-deterministic tree computes f if and only if for every input a where $f(a) = 1$ there is a path from a root to a leaf such that all literals along this path are consistent with the input a. We define the 1-certificate complexity of a function f, $[C_1(f)]$ as the minimum depth non-deterministic decision tree that computes f [Juk14]. We define the 0-certificate complexity of a function f as the dual of the 1-certificate complexity, or symbolically $C_0(f) = C_1(\neg f)$. We finally define the certificate complexity as the largest of both or $C(f) = \max\{C_0, C_1\}$

Definition (Assigned variable). An assigned variable is a variable and its value. In a boolean function, this is either a 0 or a 1.
In this paper, it is often implied if we are considering an assigned variable or a variable.

Definition (1-term). A 1-term of a boolean function f is an assigned subset of its variables such that fixing these variables to the assigned constant results in f being the constant 1 function. A 0-term is similarly defined where f is the constant 0 function [Juk14].

Definition (minterms and maxterms). A minterm of a boolean function f is a 1-term that is minimal under set-inclusion. Similarly, a maxterm is a 0-term that's minimal under set inclusion [Juk14].

Definition (certificate). For a function f, a 1-certificate is a minterm with at most $C_1(f)$ variables. Similarly, a 0-certificate is a maxterm with at most $C_0(f)$ variables. A certificate is either a 1-certificate or a 0-certificate.

When discussing the width of a function, our measure is studied through a characterization of a two-player game [Pud00] [AD08]. The game was generalized from search problems on CNFs to arbitrary functions by Goos et al [Gar+20]. We view it as a game between two players: a querier and an adversary. The querier maintains a partial assignment $p : [n] \to \{0,1,*\}$. At each step, the querier can either query a variable $i \in p^{-1}(*)$ in which case the adversary picks an $\alpha \in \{0,1\}$ and assigns $p(i) := \alpha$, or forgets a variable $i \in p^{-1}(\{0,1\})$ assigning $p(i) := *$. The adversary may later define $p(i)$ as something different from what it was upon the initial query. The game only ends if p is a certificate for f [Fil+24].

Definition (width). In a particular instance of a game π on a function f, we define the width of the game $w(\pi)$ to be the maximum size of p at any step of the game. For any function f, we define the width of our function $[w(f)]$ as the largest possible width for any game given an optimal querying strategy.

Definition (Certificate agreement). For a function $f : \{0,1\}^n \to \{0,1\}$, and an input $(\alpha_1, \alpha_2, ...\alpha_n)$ where for each $i \in [n]$, $\alpha_i \in \{0,1\}$ we say a certificate agrees with our input if for each variable x_i in our certificate, $\alpha_i = x_i$.

Definition (Certificate disagreement). For a function $f : \{0,1\}^n \to \{0,1\}$, and an input $(\alpha_1, \alpha_2, ...\alpha_n)$ where for each $i \in [n]$, $\alpha_i \in \{0,1\}$ we say a certificate agrees with our input if there is a variable x_i in our certificate where $\alpha_i \neq x_i$.

Definition (Disproven certificate). A certificate is disproven if, when querying a function, we find a variable that disagrees with our certificate.

Definition (min-certificate). In this paper, we define a min-1-certificate of $f : \{0,1\}^n \to \{0,1\}$ to be a certificate where there is an assignment of its n variables where there is no smaller 1-certificate that agrees with our input. We similarly define a min-0-certificate. We define a min-certificate as either a min-0-certificate or a min-1-certificate.

Theorem 1. *For every 0-certificate c_0, and every 1-certificate c_1 there is an assigned variable $x \in c_1$ where x disproves c_0.*

Proof. This proof is done with reference to [Juk14].
Assume, for the sake of contradictions that our claim is false. Therefore, there is a 1-certificate c_1 and a 0-certificate c_0 where there is no $x \in c_1$ where x disproves c_0. However, this means that there is an input where both c_1 and c_0 agree with our input. As c_1 agrees with f, f must return 1. As c_0 agrees with f, f must return 0. However, f cannot return both 1 and 0 for a single input. Therefore, our assumptions must be false, and our claim is true. □

Theorem 2. $D(f) \geq w(f) \geq C(f)$

Proof. To show $D(f) \geq w(f)$, we notice that if we exactly simulate the deterministic querier without forgetting anything, we terminate with $D(f)$ variables in memory. To show $w(f) \geq C(f)$, we notice a nondeterministic querier can query the $w_{\text{out}}(f)$ variables at the end of a forgetting queriers game. As $w_(f)/gew_{\text{out}}(f)$, it follows that $w(f) \geq C(f)$. □

There is a further bound that $D(f) \leq C(f)^2$ [Juk14], and there are functions where this is optimal. We also know of functions f where $D(f) = \sqrt{C(f)}$, but we do not know of functions g where $w(g)$ is significantly different to $C(g)$. However, we will explore boolean relations where this difference does exist.
However, first, we will construct characterizations that are used regularly in this paper.

3 Characterizations

We begin by constructing some characterizations of important components of query complexity. These are done to allow us to identify these components easily.
We start by characterizing 1-terms.

Lemma (Characterization of a 1-term). *For a function f, a set of assigned variables q is a 1-term if and only if for every 0-certificate c_0 there is an $x \in q$ where x disproves c_0.*

Proof. We will first prove that if q is a 1-term, then for every 0-certificate, c_0 there is an $x \in q$ that disproves c_0.
In theorem 1 we proved that for every 0-certificate c_0, and every 1-certificate c_1 there is an $x \in c_1$ where x disproves c_0. This proof can trivially generalise to 0-terms and 1-terms, so for any 0-certificate c_0, there is an $x \in q$ where x disproves c_0. Thus we have shown this side of our claim.
Next, we will prove that if - in a set of queried states q - for every 0-certificate there is an $x \in q$ that

disproves it, then q is a 1-term.
Assume, for the sake of contradiction, there is an input p where $q \subseteq p$ and $f(p) = 0$. This implies the existence of a 0-certificate, $c_0 \subset p$ that proves $f(p) = 0$. However, we assumed that for any 0-certificate, c_0, there was an $x \in q$ that disproves the certificate. This means that this 0-certificate must be disproven by q, so it cannot prove q. Thus, we have found a contradiction, and our original claim is true.
We have shown that both sides imply each other, so they must be the same. □

Trivially, we can similarly characterize a 0-term.
Next, we will characterize a minterm more simply.

Lemma (Characterization of a minterm). *For a function f, a set of assigned variables q is a minterm if and only if for any $x \in q$, there is a $c_0 \in C_0$ where x uniquely disproves c_0.*

Proof. First, we will prove the forward direction. To do this, we will consider if q is a minterm and assume for the sake of contradiction that our claim is false.
Thus we have that either there is a $x \in q$ where there is no 0-certificate c_0 where x disproves c_0 or that there is an $x \in q$ where for any 0-certificate that x disproves c_0, there is a different $y \in q$ where y also disproves c_0. If there was such an $x \in q$, this would imply that q was not a minterm, as $q \backslash \{x\}$ would still disprove all 0-certificates and thus be a 1-term by the characterization of a 1-term. Thus, we must have such an $x \in q$ and, to find a contradiction, we will prove that $q \backslash \{x\}$ is still a 1-term.
Consider now, some 0-certificate c_0; we note that either x disproves c_0 or x does not disprove c_0. If x disproves c_0, by our assumption hypothesis, there is a different $y \in q$ where y disproves c_0. Hence, using y, we have an x that disproves c_0 in this case.
If x does not disprove c_0, then, as q is a 1-term, there must be a $y \in q$ where y disproves c_0 by the Characterization of a 1-term. As x does not disprove c_0, $y \neq x$, hence $y \in q \backslash \{x\}$. Hence, we have shown that whether x disproves c_0 or x does not disprove c_0 for an arbitrary 0-certificate, there is a $y \in q \backslash \{x\}$ where y disproves c_0. Thus $q \backslash \{x\}$ is a 1-term by the Characterization of a 1-term. However, this directly contradicts q being a minterm. Thus, we have found a contradiction and have shown that this side is true.
Next, we will prove the reverse direction. Assume that q is a one-term where for any $z \in q$ there is a 0-certificate c_0 where z disproves c_0 and there is no $x \in q$ where for any 0-certificate that x disproves c_0, there is a different $y \in q$ where y also disproves c_0. Consider some $x \in q$, we will prove that $q \backslash \{x\}$ isn't a 1-term, thus showing that if we remove any x from q, f could return 0.
By our assumption, there is a 0-certificate c_0 that x disproves where there is no different y that also disproves it. Thus $q \backslash \{x\}$ would not disprove this certificate.
By the characterization of a 1-term, this means $q \backslash \{x\}$ is not a 1-term. As this is true for a general $x \in q$, it must be true for all $x \in q$. Thus, we have shown that removing any x from q would result in the function not being a 1-term, so q is a minterm, proving the reverse side.
As both sides imply each other, both statements are equivalent. □

Trivially, we can characterize a maxterm in a similar way.
Sometimes, it is not clear if a minterm is a certificate. It is sometimes easier to see that a minterm is a min-certificate. Because of this, we show that both approaches are equivalent.

Lemma (Characterization of certificate complexity). *For a function f, $C(f)$ is equal to the size of the largest min-certificate.*

Proof. To do this, we will prove that $C(f)$ is at least the size of the largest min-certificate and that $C(f)$ is at most the size of the largest min-certificate.
First, we will prove that $C_1(f)$ is at least the size of the largest min-certificate. By definition, $C(f)$ is the minimum depth of a non-deterministic decision tree that decides f. Consider the input with the largest min-certificate, i. Our decision tree must decide i, let mc be the min-certificate for i. Consider the path in this decision tree that proves i. This must result in a certificate for i. By definition of a min-certificate, this must thus be at least as large as mc. Thus we have shown $C(f)$ is at least the size of the largest min-certificate.
Next we will prove that the size of the largest min certificate is at least $C(f)$. Consider a non-determinstic decision tree that sees an input i and queries the min-certificate for i. This will be

able to prove any input of f within the size of the min-certificate. Also, it will always return a certificate after querying only the size of a min-certificate of the input it reads queries. Thus, this non-deterministic decision tree will have depth at most as large as the size of the largest min-certificate. As $C(f)$ is the size of a minimum depth non-deterministic decision tree that solves f, and there is a non-deterministic decision tree that solves f in the size of the largest min-certificate queries, it follows that $C(f)$ is at most the size of the largest min-certificate.
Hence, we have shown that $C(f)$ is at least the size of the largest min-certificate and that $C(f)$ is at most the size of the largest min-certificate. Thus, they are equal. □

This is very useful because it allows us to compare minterms with certificates. This can allow us to prove we have queried less than or equal to a function's certificate complexity directly by showing that our queried states are a min-certificate or a subset of a min-certificate.

Definition (efficient). In this paper, we define a function f to be efficiently solvable if $w(f) \leq C_0(f) + C_1(f) - 1$.

We define it this way because, in the coming problems, we generally find that many problems can be solved with this efficiency. A reason we expect this to be true in general is because in all example functions we attempted, this was the case, and we could find it by building up a 0-certificate by querying 1-certificates and remembering an element of our certificate that disproves it.

4 Boolean Relations

We now look at a setting with a known significant separation between certificate complexity and width. We can define a function Search which takes in a boolean function f on some n inputs and n boolean inputs, $x = (x_1, x_2, ...x_n)$ where $f(x) = 1$. The function returns the certificate that verifies this input. In [BW01], they talked about Search(3-CNF, x). This trivially has a certificate complexity of 3 for any 3-CNF and input x. They proved that the width of Search(3-CNF, x) is linearly bounded on the number of inputs. This means there is a significant difference in the complexity of these operations in this case. We are generally focused on functions throughout this paper, but we wanted to take inspiration from this difference in relations. This is also a common idea where we often consider the cheatsheet function defined and used in [AKK16] which has been recently used for new results in query complexity.
In this paper, we consider a different problem: the Sink functions. We consider a directed graph $G = (V, E)$ with numbered edges. We then define the Sink function, $\text{Sink}_G : \{0,1\}^m \to \{0,1\}$ where $m = |E|$ and the i'th input represents the i'th edge. When evaluating $\text{Sink}_G(x_1, x_2, ..., x_m)$, we construct a directed graph G' with the same underlying graph as G. The directions of an edge x_i of G' is the same as that of G if $x_i = 0$ and the reverse if $x_i = 1$. $\text{Sink}_G(x_1, x_2, ..., x_m)$ returns 1 if and only if there is a vertex in G' that is a sink.
We consider the boolean relation Search(G', x), which returns the set of edges around the sink vertex in G'. We notice that the Sink function has short Search certificates, but a large width. To do this, we notice that the certificate complexity is the largest number of edges around a node. This is because we know there is a sink, so it has to be around a vertex. We can non-deterministically select all the edges around this node as our certificate. Therefore, the certificate complexity of this relation is the maximum degree of any node in G'. However, when we consider the width, when we query any vertex, the adversary may change which vertex is the sink each time the querier queries. This means that the querier will need to remember why each vertex is not a sink at each step. So, they must simultaneously remember one edge for each node and the degree of the last queried node's edges. This means the w(Search(Sink, x)) is at most $2|V|$. which means they are not equal.
When we consider the reason for this difference, we realize that certificate complexity does not have to worry about the function returning 0, so it simply returns the correct solution. However, when querying, the querier does not gain the same information from knowing there is a solution, because the solution can change location as they query new variables. This means they have to build up all the evidence that it has a solution before the querier has to provide the solution.
In this paper, this is why we define efficient querying of a function f to mean $w(f) \leq C_0(f) + C_1(f) - 1$, because we need to build a 0-certificates to force the adversary to give us the solution.

5 Independent Certificates

We consider the same sink function as is defined above.

Theorem 3. $w(Sink) \leq (C_0(Sink) + C_1(Sink) - 1)$

Proof. A 1-certificate of sink is a set of edges queried to show they all point towards the point. If we have queried them and find them all pointing towards the node, it is a sink, so the function returns 1. If we have not queried an edge around the point, it could still not be a sink because it can point away.
A 0-certificate of sink is a set of edges, one from each node, pointing away from the point. Therefore $C_0(sink) = |V|$. If we have this set of nodes, all the nodes cannot be sinks, so the function returns 0. If a node does not yet have an edge pointing away, that node could be a sink, so we will not know.

Consider an arbitrary graph on n nodes. Consider a querier and an adversary playing a game where the querier wants to know if the graph has a sink, but the answerer wants them not to know. The querier starts with an arbitrary graph, $G = (V, E)$. They do not know which direction any edge points. The querier can query an edge, where the adversary chooses the direction of the edge. The querier can also forget an edge's direction. If the querier re-queries an edge after forgetting, the adversary may change the direction.

Consider the querier strategy:

Algorithm 1: algorithm for querying for sinks in a graph.

```
Create a mental list $L$. This will represent permanant memory;
for node $v \in V$ do
    Query all edges $e \in E$ where $v \in e$, so $e = \{v, w\}$ for some $w \in V$;
    If there are no edges $e$ where $e = (v, w)$ the querier says there is a sink and it is v;
    Otherwise, choose an edge $e$ where $e = (v, w)$, add it to $L$ and forget all variables not in
      $L$;
If we have no more edges that could be a sink, return 0 because there is no sink;
```

We will now prove this algorithm will work in a finite amount of time and never queries more than $C_0(Sink) + C_1(sink) - 1$ items at once.
We will do this inductively by proving that each iteration will either solve sink correctly and efficiently, or will continue to the next step with i queries where i is the number of previous iterations and each x_i proves a different node is not a sink.
We will first consider the first iteration from the querier's perspective. We start by creating our mental list L and taking a node $v \in V$. We then query all edges associated with v, and check if any edge points away from v. This is a min-1-certificate, so we query less that $C_1(sink)$ edges. Here, we will either find an edge going from v, or we will not. If there no edge is going from v, this algorithm returns 1 correctly because v is a sink. This will be efficient because we queried at most $C_1(f)$ edges and $C_0(f) > 0$. If there is such an edge, then this algorithm remembers that edge but forgets all other edges we queried as they are not in L. If this is the only node, then as v is not a sink, no nodes are sinks, thus we will solve it efficiently. Otherwise, we have queried 1 edge in L, which is the number of nodes we've queried. Thus this algorithm works if the first node happens to be a sink.
Consider now if we have queried some number $'m'$ nodes so far and remembered i edges in L.
We first note that i has to be less than n because if we have n queried edges each proving a different node is not a sink, then no node can be a sink, so we will return 0. We will consider a new node u and query all edges including u. This will be a min-1-certificate, so we will query less that $C_1(sink)$ edges. We will also have in memory $i < n$ edges, this is less than a min-0-certificate, so we will have queried at most $C_0(sink) + C_1(sink) - 1$ edges by the characterization of a 0-certificate. We will either find an edge going from u or we will not. If no edge is going from u, then u is a sink, and we correctly return 1 efficiently. If there is a node, we remember this in L and forget all new edges not in L. If this is the last node, then all nodes cannot be sinks, and we return 0 correctly and efficiently because no node was a sink. Otherwise, we will have our original i edges proving

different nodes are not sinks and another edge proving this new node is not a sink, so we will terminate with $i + 1$ edges in memory proving different nodes are not sinks. Thus, the inductive step is true.
This algorithm will also always terminate because there is a finite number of nodes, and each iteration has us proving another node to not be a sink, so after n iterations, we will have proven all nodes are not sinks or that there is a sink.
Thus, we have shown that this algorithm is correct and efficient at solving the sink problem. □

We now consider how to abstract further from this problem to understand when and why this algorithm works. Considering how we solved this problem, we queried a 1-certificate and found evidence that it was false and remembered only that evidence. We did this for all 1-certificates to find a 0-certificate. At a glance, this reasoning makes it appear it should work for all problems, and to show how this is not the case, we will first define the strategy more generally as so:

Algorithm 2: Disprove 1-certificates Algorithm

```
Create a permanent memory list $L$;
for all 1-certificates $c_1$ do
    Query the entire 1-certificate;
    If the 1-certificate is satisfied, return 1;
    Otherwise, remember a variable that contradicts this certificate in $L$;
    Forget all the variables not in $L$;
return 0;
```

We first show that this algorithm will be correct. We know that if it never finds a true 1-certificate, there will be a variable in l proving each certificate false. Hence, by our Characterization of a 0-term, this will be a 0-term. Thus, if it returns, it will do so correctly. We also notice that there is a finite number of 1-certificates for any given function, and querying a certificate takes a finite amount of time. We also only query each certificate once, so this will terminate. Therefore, it will terminate correctly.
However, when proving efficiency, this algorithm will fail. To show this, consider the indexing function.
This is a function on n + 2^n variables, where the function f($x_0, x_1, ..., x_{n-1}, Y_0, Y_1, ..., Y_{2^n-1}$) where it returns $Y_{x_0 x_1 ... x_{n-1}}$ where $x_0 x_1 ... x_{n-1}$ is a binary string from the first n variables.
For the indexing function, a certificate is the set of all x variables and the accompanied Y variable [Juk14]. By theorem 2, this means $w(index) \geq n + 1$. It is also easy to see that a querier could also query the first n variables, then the associated Y variable, so $w(index) = n+1$. Consider running the above algorithm on the indexing function. Consider a querier following the disprove 1-certificates algorithm. We will prove that the adversary can ensure that the querier is not efficient at solving the problem.

Proof. Consider if when the querier queries the certificate, the adversary always answers with the Y variable of the certificate. We know that there is one in each certificate, so this is always possible. As everytime the querier asks about a certificate, the adversary responds so that the Y variable is unsatisfied, there is no satisfied 1-certificates. Thus this algorithm must terminate with no 1-certificates satisfied. Consider if there is a $0 \leq i \leq 2^n - 1$ for which Y_i is not in the querier's mental list L. As the only variable added to the list is a Y variable, all the x variables and Y_i are not queried. As $0 \leq i \leq 2^n - 1$, the binary representation is a possible assignment for x; therefore, another certificate is still unqueried. Therefore, if this algorithm terminates with no remaining certificates, every Y variable is in the querier's mental list. However, we notice that 2^n Y variables need to be queried, whereas there are only $n + 1$ variables in a certificate. Therefore, this is not efficient, as $C_0(index) + C_1(index) - 1 = n + 1 + n + 1 - 1 = 2n + 1$. However, we query at least 2^n variables and for any $n > 3$, this is larger than $2n + 1$, since n can be more than 3, this algorithm is not efficient. □

As we see, this means that there are some functions where the Disprove 1-certificates Algorithm is not efficient at solving them.

We now look at why this algorithm is efficient for the sink function but not for the indexing function. In the sink function, any assignment of any variable can only prove a single node is not a sink. This means that for any two 1-certificates $c_1^1, c_1^2 \in C_1$, the intersection of variable assignments is empty.
We will name functions with this property '1-independent functions'. We will now prove that all such functions are efficiently solvable. We can define a '0-independent function' similarly.

Theorem 4. *A 0-certificate of a 1-independent function has as many elements as there are 1-certificates.*

Proof. Consider a 1-independent function f, and consider a 0-certificate of this function. By the characterization of a 0-term q, we notice that for any 1-certificate c_1 of f, there must be an $x \in q$ where x disproves c_1. However, as f is an independent function, x cannot disprove another 1-certificate. As this is true for all 1-certificates, this means there are at least as many different x variables in q disproving different 1-certificates as there are 1-certificates. As any 0-certificate is also a 0-term, and every 0-certificate has at least as many elements as there are 1-certificates, this means that $C_0(f)$ is at least as big as the number of 1-certificates. □

Theorem 5 (Independent Functions)**.** *Independent functions are efficiently solvable.*

Proof. Consider a 1-independent function f. Consider also a querier who uses the Disprove 1-certificates Algorithm on f. We will now inductively prove that after m iterations, L will have m elements and that this algorithm is efficient if it terminates at that iteration.
We will first do the base case. First, the querier will query a 1-certificate. This means there are at most $C_1(f)$ variables queried. If we find the certificate consistent, the function must return 1, so the querier will solve it efficiently. If this is the only 1-certificate, and we finish, we return 0 because no 1-certificate is possible. This will be correct and solved efficiently. Otherwise, we can remember a single variable variable in L that disproves the 1-certificate and forget every variable not in L. Thus, L has size 1 when we finish this iteration, and our base case is true.
Next, assume that this statement is still true after some number of iterations. As the algorithm only queries as many times as there are certificates, the previous m will be less than the number of 1-certificates of f. By Theorem 4, this means $m < C_0(f)$. Next we query a 1-certificate, let this be $p \leq C_1(f)$ queries, we will have queried $m + p < C_0(f) + C_1(f)$ Thus we have queried $\leq C_0(f) + C_1(f) - 1$ and we are still efficient. If this 1-certificate has nothing proving it false, then the querier returns 1 correctly and efficiently. Otherwise, there is a variable proving it false, and the querier adds it to L. If this is the last 1-certificate, no 1-certificates are possible, so the querier correctly returns 0 efficiently. Otherwise, the querier forgets all variables not in L. so remembers the previous $m+1$ variables which is the number of iterations so far. Therefore, the inductive step is true.
Thus, we have shown that this algorithm will terminate, and we proved that when it terminates it will be efficient. Therefore, the querier will always be efficient when using the disprove 1-certificates algorithm on any independent function. So, all independent functions are efficiently solvable. □

Some examples of functions with independent certificates include the Sink function and Tribes. Tribes is defined below.
Tribes is a boolean function on some $n \cdot m$ variables.
To define Tribes, we will consider $\text{AND}_n : \{0,1\}^n \rightarrow \{0,1\}$ where $\text{AND}_n(x_1, x_2, ..., x_n) = x_1 \wedge x_2 \wedge ... \wedge x_n$.
We also define $\text{OR}_n : \{0,1\}^n \rightarrow \{0,1\}$ where $\text{OR}_n(x_1, x_2, ..., x_n) = x_1 \vee x_2 \vee ... \vee x_n$.
Then tribes is defined as $\text{AND}_n \circ \text{OR}_m$ for some $n, m \in \mathbb{N}$. We notice that a 0-certificate of AND_n is any variable being a 0. We also notice that a 0-certificate of OR_m is all variables being 0. This means a 0-certificate of Tribes is the set of variables in any disjunction. As each disjunction is in its own OR_n, there are no repeated variables, so this function has independent 0-certificates. Thus, it is efficiently queryable.
A potential method to consider in the future to generalize this proof to all functions is to limit the number of 1-certificates any variable disproves to a fixed constant. Then, inductively show that the function is efficiently queryable for any limit. However, how to prove or disprove this inductive statement still needs to be clarified.

When we proved that the disprove 1-certificate algorithm was not efficient for the indexing function, that we terminated with a maxterm, but not a 0-certificate in L, causing this inefficiency. We turn to monotone functions to avoid this issue because any minterm in a monotone function is a 1-certificate and any maxterm in a monotone function is a 0-certificate. This will be a proven theorem.
This may provide us further insight to conduct the inductive argument above.

6 Monotone Functions

To define a monotone function, we first need to define $\leq$. For two inputs $x, y \in \{0,1\}^n$, we say $x \leq y$ if for all $i \in [n], x_i \leq y_i$. A monotone function f is a function where if $x \leq y$, then $f(x) \leq f(y)$ [Juk14]. We now consider the next idea. This is to prove that monotone functions can be queried efficiently. We will first explore some properties of monotone functions. We consider a monotone function f.

Theorem 6 (minterm-certificate equivelance)**.** *For any monotone function, all minterms are 1-certificates, and all maxterms are 0-certificates.*

Proof. Consider a monotone function f and a minterm q of a function. As f is a monotone function, q's variables must all have a 1-assignment. We prove this by contradiction by assuming for the sake of contradiction that there is a variable in q that has a 0-assignment. As q is a minterm, if an input agrees with q, then f returns 1. As there is a y in q had a 0-assignment, we can consider the same input, but we set y to 1. As f is monotone, and the original input is less than this input, this implies f of the new input returns 1. However, this means that $q \backslash \{y\}$ must be a 1-term, so q is not a minterm. So it must be the case that all variables in q have a 1-assignment.
We now construct the input where any variable in q is assigned a 1 and any variable not in q is assigned a 0. As our input agrees with q, f must return 1. So there is a 1-certificate to prove this. We also notice that any 1-certificate is a minterm, and we have already shown that any variable in a minterm has a 1-assignment. This means a 1-certificate of our input must be a subset of q. However, q is a minterm, so the only 1-term subset of q is q itself. So q is a min-1-certificate of this input. This means q is a 1-certificate.
As q was a general minterm, it follows that all minterms are 1-certificates for any monotone function.
Similarly, we can show that all variables in maxterms have 0-assignments, and all maxterms are 0-certificates. □

We now define 1-covering a variable as follows:

Definition (1-covering). A variable x is called 1-covered in a set of variables q with respect to a function if for any 1-certificate c_1 in q that is disproved by x, there is another variable y in q disproving it.

We similarly define 0-covering. This means the variable is 1-covered if it is not a unique disprover of any 1-certificate in the set q.
We now construct an algorithm for the querier where they remove any 1-covered variables as it queries. This way, we ensure we remove variables that are not disproving any certificates any longer.

Algorithm (Minterm Storing)

```
Create a mental list $L$;
for all 1-certificates $c_1$ do
    If L already disproves $c_1$, go to the next certificate  Query all of $c_1$  If our currently queried
    states $q$ agrees with $c_1$, return 1  Otherwise, there must be a variable, $x$ that disproves $c_1$. Add
    this to $L$  Forget all variables not in $L$  If there is a variable that is 1-covered in $L$, forget that
    variable and remove it from $L$
return 0
```

We now show this algorithm terminates with L as a maxterm if this algorithm returns 0, and thus a 0-certificate if our function is monotone.

Proof. We conduct this proof inductively. To do this, we show that after each iteration, for all previously used certificates, there is a variable in L disproving it. We also show that after each iteration, for each variable, there is a 1-certificate that the variable uniquely disproves.
We first consider the base case, the first iteration.
We consider an arbitrary 1-certificate c_1 and query it. If our currently queried states q agree with c_1, then our function returns 1, and we correctly return 1.
Otherwise, there is a variable, x that disproves c_1, we add this to L. We now forget all variables not in L, so $q = \{x\}$. We finish the first iteration as there are no 1-covered variables. x disproves our 1-certificate uniquely, so our claim is true in the base case.
We now consider the inductive step.
Assume that our claim has been true for some m iterations, consider the $m + 1$'th iteration. We consider a new 1-certificate k_1 and query it. If our currently queried states q agree with k_1, then our function returns 1, and we correctly return 1.
Otherwise, there is a variable y that disproves c_1; we add this to L and consider L_1 to be the old state of L. We now forget all variables not in L. This means that $L = L_1 \cup \{y\}$. By the inductive hypothesis, there is a variable in L_1 that disproves all previous certificates uniquely. We also have that y disproves k_1. This is unique, because if L already disproved k_1, we would have moved to the next certificate. However, there may be variables in L that are 1-covered by y.
If there are any variables z that are 1-covered in L, we forget these variables and remove them from L.
As z is 1-covered, for any 1-certificate v_1 that z disproves, there is another variable $w \in L$ that disproves v_1. Therefore, all certificates are still disproven by a variable in L. We also have that no variables are 1-covered. Therefore, for each variable, there is a 1-certificate they uniquely disprove. Thus, our inductive hypothesis is true.
As there are finitely many 1-certificates, and each iteration disproves an extra 1-certificate, this will terminate.
Consider if this algorithm returns 0 by exhausting all 1-certificates. Then we notice that L will be a set of variables that disprove all 1-certificates, where no variables are 1-covered.
By the characterization of a maxterm, this implies L is a maxterm. □

It also appears that L is increasing in size each iteration, which would imply in a similar way to how we proved for the disprove 1-certificate algorithm that this would be efficient for monotone functions. However, this is not true because the size of L does not always increase for any order of certificates on any monotone function. We approach finding a function where the size of L is not increasing by constructing a function that simultaneously forgets multiple variables of q. This means that adding a new variable needs to make multiple variables 1-covered. We construct this function by defining its 1-certificates, which exactly defines the function itself. We consider the function $f : \{0,1\}^6$ with one certificates:

- $c_1^1 = \{x_1 = 1, x_2 = 1\}$
- $c_1^2 = \{x_2 = 1, x_3 = 1\}$
- $c_1^3 = \{x_4 = 1, x_5 = 1\}$
- $c_1^4 = \{x_5 = 1, x_6 = 1\}$
- $c_1^5 = \{x_2 = 1, x_5 = 1\}$

This is a monotone function, because if the function returns 1, one of the 1-certificates verifies this. As each certificate has only 1-assigned variables, changing a variable assigned a 0 to a 1 is not changing the variable in the certificate. Therefore, the input still agrees with the certificate, and the functions still returns 1. It is also clear that this function has a 1-certificate complexity of 2. We now prove that the 0-certificate complexity is 3.

Proof. If we consider the input $x = (0, 1, 0, 1, 0, 1)$, we notice $f(x) = 0$ as all 1-certificates can be disproven. This is because $x_1 = 0$ disproves c_1^1, $x_3 = 0$ disproves c_1^2, $x_5 = 0$ disproves c_1^3, c_1^4 and c_1^5.

However, we also notice that if we change x_1 to 1, then c_1^1 proves f will return 1. If we change x_3 to 1, then c_1^2 proves f will return 1. If we change x_5 to 1, then c_1^5 proves f will return 1. Therefore $\{x_1, x_3, x_5\}$ is a min-0-certificate for this input, so $C_0(f) \geq 3$ by the characterization of certificate complexity.
We also notice that if the function returns 0, then c_1^5 is disproven. Therefore either $x_2 = 0$ or $x_5 = 0$. If $x_2 = 0$, then we notice that c_1^4 and c_1^5 must be disproven. Therefore either $x_5 = 0$, in which case $\{x_2 = 0, x_5 = 0\}$ is a 0-certificate or both $x_4 = 0$ and $x_6 = 0$ in which case $\{x_2 = 0, x_4 = 0, x_6 = 0\}$ is a 0-certificate. If $x_5 = 0$, we can prove similarly with $x_1 = 0$ and $x_3 = 0$ or $x_2 = 0$. Therefore, there is always a 0-certificate of size ≤ 3. Therefore $C_0(f) = 3$. □

Therefore, to be efficient in querying this function, we need to solve it with a width $w(f) \leq C_0(f) + C_1(f) - 1 = 2 + 3 - 1 = 4$.
We will now prove that there is an order of querying the certificates where the Minterm Storing Algorithm is not efficient.

Proof. Consider if the querier queries the certificates in order $c_1^1, c_1^2, c_1^3, c_1^4, c_1^5$ and the adversary tells them that $x_1 = 0, x_3 = 0, x_4 = 0, x_6 = 0$ and $x_2 = 0$, respectively, after each certificate as the disprover of the certificate. Assume the querier follows the minterm storing algorithm and they remember that these variables disprove the certificates. Any step before the querier remembers $x_2 = 0$, $x_1 = 0$ will uniquely disprove c_1^1, $x_3 = 0$ will uniquely disprove c_1^2, $x_4 = 0$ uniquely disproves c_1^3, $x_5 = 0$ uniquely disproves c_1^4. So none of the variables will be 1-covered in L. This means we will remember all four variables before querying another two when we query c_1^5. However, this means we will remember 6 variables at once, whereas $C_0(f) = 3$ and $C_1(f) = 2$. So $C_0(f) + C_1(f) - 1 = 4 < 6$. so it was not queried efficiently.
□

However, the querier will know this and could choose a different way of querying this problem to address it. This can be done as so:

Proof. Consider if the querier instead queried c_1^5 first. When the queryer queries c_1^5, the adversary has to respond with either $x_2 = 0$ or $x_5 = 0$, otherwise, c_1^5 proves we return 1, and our claim will be true.
If $x_5 = 0$, then the querier will know that c_1^5, c_1^4 and c_1^3 do not agree with the input. If we choose to query c_1^4, we go to the next certificate as it is already disproven by $x_5 = 0$. If we next choose to query c_1^3, we also go to the next certificate. If we now query c_1^2, we either find $x_2 = 0$ or $x_3 = 0$, or we return 1 because c_1^2 is true. If $x_2 = 0$, then we will have no more certificates to prove. If $x_3 = 0$, then we will remember this and query c_1^1. As this is the last certificate, this will verify either way, and we will only forget variables from here. This means we will terminate with a width of 4, and this is efficient.
If $x_2 = 0$, then the querier will know that c_1^5, c_1^2 and c_1^1 do not agree with the input. Similarly, if we choose to query c_1^2 then c_1^1, followed by c_1^3 and c_1^4, this will also be efficient.
We have now exhausted all possibilities and queried with a width of 4, so this is efficiently queriable using the minterm storing algorithm. □

As such, there may still always exist an order of 1-certificates queried for any monotone function where this algorithm works. We notice that choosing c_1^5 first lead us to the best solution. This happened because when x_2 and x_5 are the variables that disprove the most certificates. So when we query c_1^5, we always disprove 3 certificates. Compare this to the other certificates that all contain a variable that can only disprove a single certificate. Therefore, the responder can continue to give us results with very little information, and then when we query c_1^5, it gives us information that a lot of what we queried was 1-covered. To fix this, if we start with the certificate with the most overlapping variables, it may be possible to show that this will work. An idea of how we could construct this would be as follows. We consider the function defined over a boolean function that takes in a certificate of the function and returns the minimum number of certificates any variable in the certificate disproves.
We then start with the certificate that with the largest value from this function. Using this ordering

of certificates, we still need to prove whether this algorithm will work for all monotone functions with the Minterm Storing Algorithm.

7 Conclusion

We have provided useful characterizations of really important query complexity terms. These were central to helping us identify when we had found a minterm or a certificate. We further investigated what it would mean for a querier to be efficient, and we identified a definition of efficiency that appears to be optimal if we can show all boolean functions can be queried accordingly. We then investigated some algorithms a querier can use and when those algorithms are efficient. By doing this, we discovered that any boolean function with independent certificates is efficiently queryable. It is likely worth exploring further here to identify if we can expand this proof to functions with limited numbers of dependencies between certificates. It may also be interesting to explore if it is possible to expand this proof to functions where any variable can disprove a limited number of certificates. We looked further into monotone functions, because they are much easier to identify when we have a certificate. Here, we find an algorithm that immediately appears to be efficient for monotone functions, but does not work. We further explore why this occurs and explain how we may expand this to prove that general monotone functions are efficient. We expect there may be interesting findings in an ordering for querying 1-certificates that may result in the minterm finding algorithm being efficient.

Hence, this paper raises new questions and provides strategies that may improve future research into dag-like query complexity.

8 Acknowledgements

This endeavour would not have been possible without Marc Vinyals, who provided me with a topic to research and weekly meetings where we discussed what I had achieved and how I would proceed in the future. His supervision of this research project has been amazing and I am incredibly thankful to have had his supervision throughout this project.

References

[AD08] Albert Atserias and Víctor Dalmau. “A combinatorial characterization of resolution width”. In: *J. Comput. Syst. Sci.* 74.3 (2008), pp. 323–334. DOI: `10.1016/J.JCSS.2007.06.025`. URL: `https://doi.org/10.1016/j.jcss.2007.06.025`.

[AKK16] Andris Ambainis, Martins Kokainis, and Robin Kothari. “Nearly Optimal Separations Between Communication (or Query) Complexity and Partitions”. In: *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*. Ed. by Ran Raz. Vol. 50. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 4:1–4:14. DOI: `10.4230/LIPICS.CCC.2016.4`. URL: `https://doi.org/10.4230/LIPIcs.CCC.2016.4`.

[Amb+17] Andris Ambainis et al. “Separations in Query Complexity Based on Pointer Functions”. In: *J. ACM* 64.5 (2017), 32:1–32:24. DOI: `10.1145/3106234`. URL: `https://doi.org/10.1145/3106234`.

[BGS75] Theodore P. Baker, John Gill, and Robert Solovay. “Relativizations of the P =? NP Question”. In: *SIAM J. Comput.* 4.4 (1975), pp. 431–442. DOI: `10.1137/0204037`. URL: `https://doi.org/10.1137/0204037`.

[BW01] Eli Ben-Sasson and Avi Wigderson. “Short proofs are narrow - resolution made simple”. In: *J. ACM* 48.2 (2001), pp. 149–169. DOI: `10.1145/375827.375835`. URL: `https://doi.org/10.1145/375827.375835`.

[BW02] Harry Buhrman and Ronald de Wolf. "Complexity measures and decision tree complexity: a survey". In: *Theor. Comput. Sci.* 288.1 (2002), pp. 21–43. DOI: `10.1016/S0304-3975(01)00144-X`. URL: `https://doi.org/10.1016/S0304-3975(01)00144-X`.

[CR79] Stephen A. Cook and Robert A. Reckhow. "The Relative Efficiency of Propositional Proof Systems". In: *J. Symb. Log.* 44.1 (1979), pp. 36–50. DOI: `10.2307/2273702`. URL: `https://doi.org/10.2307/2273702`.

[Fil+24] Yuval Filmus et al. "Proving Unsatisfiability with Hitting Formulas". In: *15th Innovations in Theoretical Computer Science Conference, ITCS 2024, January 30 to February 2, 2024, Berkeley, CA, USA*. Ed. by Venkatesan Guruswami. Vol. 287. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 48:1–48:20. DOI: `10.4230/LIPICS.ITCS.2024.48`. URL: `https://doi.org/10.4230/LIPIcs.ITCS.2024.48`.

[Gar+20] Ankit Garg et al. "Monotone Circuit Lower Bounds from Resolution". In: *Theory Comput.* 16 (2020), pp. 1–30. DOI: `10.4086/TOC.2020.V016A013`. URL: `https://doi.org/10.4086/toc.2020.v016a013`.

[Hua19] Hao Huang. "Induced subgraphs of hypercubes and a proof of the Sensitivity Conjecture". In: *CoRR* abs/1907.00847 (2019). arXiv: `1907.00847`. URL: `http://arxiv.org/abs/1907.00847`.

[Juk14] Stasys Jukna. "Boolean Function Complexity Advances and Frontiers". In: *Bull. EATCS* 113 (2014). URL: `http://eatcs.org/beatcs/index.php/beatcs/article/view/275`.

[Pud00] Pavel Pudlák. "Proofs as Games". In: *Am. Math. Mon.* 107.6 (2000), pp. 541–550. URL: `http://www.jstor.org/stable/2589349`.

[Sip97] Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997. ISBN: 978-0-534-94728-6.