Delegation for Search Problems

Justin Holmgren
NTT Research, Sunnyvale, CA, USA

Andrea Lincoln
University of California Berkeley, CA, USA

Ron D. Rothblum ✉
Technion, Haifa, Israel

Abstract

The theory of proof systems in general, and interactive proofs in particular, has been immensely influential. Such proof systems allow a prover to convince a verifier whether a given statement is true or not – namely to solve a decision problem. In this work we initiate a study of interactive proofs for *search problems*.

More precisely, we consider a setting in which a client C, given an input x, would like to find a solution y satisfying $(x, y) \in R$, for a given relation R. The client wishes to delegate this work to an (untrusted) advisor A, who has more resources than C. We seek solutions in which the communication from A is short, and, in particular, shorter than the length of the output y. (In particular, this precludes the trivial solution of the advisor sending y and then proving that $(x, y) \in R$ using a standard interactive proof.)

We show that such search delegation schemes exist for several problems of interest including (1) longest common subsequence (LCS) and edit distance, (2) parsing context-free grammars and (3) k-SAT.

2012 ACM Subject Classification Theory of computation → Interactive proof systems

Keywords and phrases Interactive Proofs, Fine-Grained Complexity, Delegation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.73

Category Track A: Algorithms, Complexity and Games

Funding *Ron D. Rothblum*: Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Acknowledgements We thank Benny Applebaum for helpful discussions.

1 Introduction

Interactive proofs, conceived by Goldwasser, Micali and Rackoff [13], allow a prover to convince a verifier that a given computational statement of the form $x \in L$ is true, where L is a language. They require that if the statement is true, then there is a strategy that will convince the verifier to accept with high probability; whereas if the statement is false, then the verifier should reject with high probability no matter what the prover does. Interactive proofs have had an incredible and enduring impact on complexity theory, cryptography, and beyond.

While interactive proofs help the verifier to determine whether a given statement is true or not, often we are interested in actually finding a solution: fixing some relation R of interest, we would like given an input x to find a solution $y \in R(x)$, where $R(x) := \{y : (x, y) \in R\}$. In other words, we are interested in solving *search problems*.


In this work, we consider a setting in which a computationally bounded client C is given an input x and would like to find some $y \in R(x)$ with help from a powerful advisor A.[1] We refer to a protocol for this setting as a "search delegation scheme" and note that the notion has immediate connections to cloud computing. Continuing the parallel to interactive proofs, our main focus is on protocols in which the advisor is viewed as untrusted. However, as discussed below, protocols that assume a trusted advisor are also non-trivial, and arguably even more natural.

One trivial approach to designing search delegation schemes is for the advisor to simply compute some $y \in R(x)$, send y to the client, and then prove that indeed $y \in R(x)$. The downside of this solution is that y may be so long that just sending y may already be extremely costly. Thus, in this work we are interested in search delegation schemes with *laconic* advisors, i.e. advisors that are restricted to sending a short (h-bit) hint, where $h \ll |y|$ is a parameter. Note that an h-bit hint can at most increase the probability with which the client successfully computes some $y \in R(x)$ by a multiplicative factor of 2^h. We seek to understand how much of this increase is actually realizable, and for which values of h.

The restriction that the advisor can only send a short hint makes the construction of search delegation schemes, even with a trusted advisor, algorithmically non-trivial. Moreover, given a search delegation scheme with a trusted advisor, we can in many cases upgrade the scheme (using interactive proofs or arguments generically and in a black-box way) to provide correctness guarantees to the client even when interacting with an untrusted advisor. In fact protocols for trusted advisors comprise the technical heart of our constructions, even those for untrusted advisors.

Before proceeding, we mention several prior works that can be cast as studying proof-systems for search problems, but with communication that is as long as the solution. First, the search version of NP (e.g., finding a satisfying solution to a formula, or a 3-coloring of a graph), often referred to as FNP or PC [11], consists of search problems whose solutions are efficiently verifiable. However, no distinction is made between the hint length vs. the length of the solution. There are also several works in cryptographic contexts that consider using interaction for solving search problems (e.g., [10, 8, 4]). Additionally, there has been a study of fine-grained complexity and approximation that connects the fine-grained hardness of approximation to interactive proof systems for space-bounded computation [7]. However, in these works the communication is as large as the output y (similarly to the canonical solution in which the result is sent by the advisor and then verified).

1.1 Search Delegation

We define a search delegation scheme as follows:

▶ **Definition 1.** *A* search delegation scheme *for a relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ consists of a client C and an advisor A. The two parties interact (possibly using randomness) on common input x and at the end of the interaction the client outputs a string denoted by $(C, A)(x)$. The delegation scheme has completeness error $c = c(|x|)$ and soundness error $s = s(|x|)$ if:*

- *(Completeness:) For every input x such that $R(x) \neq \emptyset$, it holds that $(C, A)(x) \in R(x)$ with probability at least $1 - c(|x|)$.*
- *(Soundness:) For every input x and every (potentially malicious) advisor A^*, it holds that $(C, A^*)(x) \in R(x) \cup \{\perp\}$ with probability at least $1 - s(|x|)$.*

[1] We use the nomenclature of client/advisor rather than verifier/prover since the goal of the interaction is not to prove correctness but rather to find a solution. Nevertheless, we emphasize that the role of the client (resp., advisor) is analogous to that of the verifier (resp., prover).

Here and throughout this work the $\perp$ symbol represents a "reject" by the client. In case the errors are not specified explicitly we default to *perfect completeness* (i.e., $c \equiv 0$) and *negligible soundness error* (i.e. $s(n) = n^{-\omega(1)}$).

The main resources that we focus on are the running time of the client, and the amount of communication between the parties (a secondary goal is bounding the running time of the (honest) advisor).

1.1.1 Variant Definitions

As discussed above, we will mainly be interested in search delegation schemes in which the communication complexity is less than the output length of the relation. Given this, Definition 1 is interesting even if we only require completeness to hold (without any guarantees against cheating advisors). We refer to schemes satisfying this weaker definition as *honest-advisor search delegation schemes*.

Actually, it will be convenient to present many of the protocols in this work as such honest-advisor search delegation schemes. Note that in case the relevant relation R has an efficient interactive proof for proving that $(x, y) \in R$ (with communication $\ll |y|$), then we can easily "bootstrap" an honest-advisor scheme into a full-fledged one. See Section 5 for details.

We will also sometimes consider a relaxation of the soundness condition of Definition 1 in which soundness is only required to hold only against malicious advisors A^* that are polynomially bounded. This is similar to the notion of computationally sound proofs, aka arguments, from the literature. We refer to protocols satisfying this weaker notion as computational search delegation schemes.

Lastly we remark that in contrast to standard interactive proofs, it is not clear how to reduce the completeness and soundness errors in search delegation schemes. The problem is that if the basic protocol is repeated, each invocation may yield a different candidate solution. While in general we do not know how to check which (if any) of the solutions is valid, in cases for which there *is* an (efficient) method of doing so, we can reduce the errors by repetition.

1.2 Our Results

We construct search delegation schemes for a number of problems of interest from the literature. In all of our schemes the client is given a hint whose length is sub-linear in the output length of the problem, and runs in significantly less time than the best known algorithms for the problem.

1.2.1 Longest Common Subsequence and Edit Distance

A *common subsequence* between strings $x \in \Sigma^n$ and $y \in \Sigma^m$, over an alphabet Σ, is a string z that appears in both x and y as a (possibly non-contiguous) sub-string. The *longest common subsequence* (LCS) problem asks, given x and y, to find a common subsequence of maximal length. When $n = m$, it is known that LCS requires $n^{2-o(1)}$ time if the strong exponential time hypothesis (SETH) holds [1].

The *edit distance* (Edit) problem is a related and central problem with important applications to computational biology. We consider a search version of the problem in which the goal is, given strings x and y, to find a minimal sequence of operations for transforming x into y, where the allowed operations are single-character insertions, deletions, and substitutions. Similar to LCS, there is a known running time lower bound for Edit of $n^{2-o(1)}$ based on SETH [5].

Our first result is an honest-advisor search delegation schemes for solving both LCS and Edit. We show how to solve both problems given a sub-linear hint string and so that the client runs in $o(n \cdot m)$ time. More generally, for every parameter a, we construct a protocol with an a-bit hint and running time $O\big(n+m+a+\frac{n \cdot m}{a} \cdot \log(n \cdot m)\big)$. For example, specializing to the case that $n = m$ and taking $a = n/\mathsf{polylog}(n)$ we get sub-linear communication $n/\mathsf{polylog}(n)$ with quasi-linear client running time.

Furthermore, using cryptographic techniques we can bootstrap the protocols to hold against computationally bounded cheating advisors, with essentially the same parameters (assuming the existence of collision-resistant hash functions and using additional rounds of interaction).

1.2.2 Parsing Context-Free Grammars

Context-free grammars are a computational model which is particular well suited for describing programming language structure. A context-free grammar is composed of production rules of the form $A \to \alpha$, where A is a variable and α is a string composed of variables and "terminals" (aka alphabet symbols). A word is derived by a grammar by applying production rules starting with some initial variable by replacing each occurrence of a variable using a corresponding production rule for that variable. The final word, composed only of terminals (i.e., after all variables have been replaced), is said to be derived by the grammar. We follow the convention that variables are described in upper case and terminals in lowercase.

A derivation tree (aka parse tree) is a tree describing the derivation of the word - namely, the root of the tree is labeled by the initial variable and the children of each vertex are labeled based on a production rule applied to the label of the parent. Thus, the leaves of the tree are labeled by the terminals of the derived word. The important computational task of *parsing* is, given a description of a grammar and a word w, to output a derivation tree describing the derivation of w using the grammar (or to output $\perp$ if the word cannot be derived by the grammar).

The best known algorithm for parsing context-free grammars (in Chomsky[2] normal form), due to Valiant [20], takes time $O(n^{\omega_b})$, where $\omega_b \geq 2$ is the exponent for Boolean matrix multiplication[3]. There is also evidence that a substantially faster algorithm does not exist [17, 2].

As our second main result, we construct a search delegation scheme for context-free parsing (of Chomsky normal form grammars) with sub-linear communication and quasi-linear work for the client. In more detail, for every parameter a, we construct a full-fledged search delegation scheme (even against computationally unbounded cheating advisors) in which the advisor sends a bits and the clients running time is $\tilde{O}(a + n^{\omega_b}/a^{\omega_b - 1})$. For example with $a = n/\mathsf{polylog}(n)$ we get quasi-linear client time, and for $a = n^{\frac{\omega_b - 2}{\omega_b - 1}} < \sqrt{n}$ we obtain a quadratic-time client with $\tilde{O}(\sqrt{n})$ communication.

---

[2] A grammar is in Chomsky normal form if the production rules only have one of the following three forms: $A \to BC$, $A \to a$, or $A \to \varepsilon$, where ε denotes the empty string. Recall that any grammar can be readily modified to be in Chomsky normal form.

[3] Boolean matrix multiplication is defined similarly to standard matrix multiplication except that the inner product operation is replaced by an OR of ANDs. The boolean matrix multiplication exponent is defined as the minimum number, ω_b, such that a $n^{\omega_b + o(1)}$ algorithm exists. The best known upper bound on ω_b is roughly 2.37286 [3].

1.2.3 k-SAT

The search version of the k-SAT problem is to find a satisfying assignment for a given k-CNF formula (i.e., a CNF boolean formula whose clauses contain at most k literals). The problem is well-known to be NP complete when $k \geq 3$. Moreover if SETH holds, then for all $\epsilon > 0$ there exists k such that k-SAT cannot be solved in time $O(2^{(1-\epsilon)n})$.

Nevertheless, there has been a fascinating line of work studying non-trivial algorithms for k-SAT for fixed constant k. The current fastest algorithm for k-SAT builds on PPSZ [14]. The PPSZ algorithm [19] is another classical, albeit slightly slower, algorithm whose running time is $\approx 2^{c_k \cdot n}$, where c_k tends to $1 - \Theta(1/k)$.

A naive approach for a search delegation scheme for k-SAT would be for the advisor to simply specify the value of a of the variables and then having the client solve the residual formula using a generic k-SAT algorithm. For example, using PPSZ we would obtain a client run-time of $\approx 2^{c_k \cdot (n-a)}$.

We give a non-trivial improvement on this scheme, in which given an a-bit hint, the client runs in time $2^{c_k \cdot n - a + o(n)}$. Denoting the run-time of the best known algorithm for k-SAT by $2^{c'_k \cdot n + o(n)}$, we get that the client beats the performance of the naive solution whenever $a > \frac{c_k - c'_k}{1 - c'_k} \cdot n$. We remark that for this result we assume the server and client have access to shared randomness of length $\Omega(n)$.

1.2.4 Separations

In addition to these positive results, we also give negative results indicating that solving a search problem (finding $y \in R(x)$) with a short hint can be much harder than solving the corresponding decision problem (i.e. checking whether a given y satisfies $y \in R(x)$). Our separation results all depend on cryptographic assumptions and are based on the elementary fact that an a-bit hint can boost an algorithm's success probability by at most a multiplicative factor of 2^a.

1.3 Organization

In Section 2 we describe our delegation protocols for LCS and Edit. We continue in Section 3 to describe our protocol for parsing context-free grammars, and in Section 4 we describe our protocol for k-SAT.

2 LCS and Edit Distance

The Longest Common Subsequence (LCS) problem asks, given strings $x \in \Sigma^n$ and $y \in \Sigma^m$ (over a common alphabet Σ), to find increasing sequences $i_1, \ldots, i_\ell \in [n]$ and $j_1, \ldots, j_\ell \in [m]$, for ℓ as big as possible, such that $x_{i_k} = y_{j_k}$ for all $k \in [\ell]$. The best known algorithms for LCS run in worst-case $O(n \cdot m)$ time, and when $m = n$ this is nearly optimal – LCS requires $n^{2-o(1)}$ time if the strong exponential time hypothesis (SETH) holds [1].

The search version of the Edit Distance problem, which we denote by Edit, asks, given strings $\mathbf{x} \in \Sigma^n$ and $\mathbf{y} \in \Sigma^m$, to find a minimum length sequence of operations that transforms $\mathbf{x}$ into $\mathbf{y}$, where allowed operations are single character insertions, deletions, and substitutions. Edit is also known to be solvable in $O(n \cdot m)$ time, and to require at least $n^{2-o(1)}$ time if SETH holds [5].

In this section we construction search delegation protocols for both LCS and Edit.

▶ **Theorem 2.** *For every $a = a(n, m)$, there is an honest-advisor search delegation scheme for* LCS *and for* Edit *in which the advisor sends an a-bit hint, and the client runs in time $O\big(n + m + a + \frac{nm \log(nm)}{a}\big)$.*

Furthermore assuming collision-resistant hash functions, the protocols can be improved to an untrusted-advisor search delegation scheme with computational soundness.

We start in Section 2.1 with an overview for the protocol for LCS. The remaining sections are devoted to the formal proof of Theorem 2. Since the protocols for LCS and Edit are similar, we present a common framework that captures both problems and is described in Section 2.2.

2.1 Protocol Overview

The idea underlying both protocols is to use trusted advice to facilitate a "divide-and-conquer" approach. This yields an honest advisor delegation protocol which in turn can be transformed, via Corollary 16, to a full-fledged protocol with computational soundness (assuming the existence of collision resistant hash functions).

For sake of this overview, we restrict our attention to LCS and describe a simple protocol with short advice ($O(\log(n) + \log(m))$ bits) that reduces the client's work by a factor of 2. The protocol for Edit is very similar and for both protocols the client's work can be further reduced by recursion, see the subsequent subsections for details.

Given input $(\mathbf{x}, \mathbf{y}) \in \{0,1\}^n \times \{0,1\}^m$, the trusted advisor first computes a longest common subsequence $\mathbf{z}$. Fix a partial mapping $\mu : [n] \to [m]$ from the coordinates of $\mathbf{x}$ to those of $\mathbf{y}$ that correspond to the subsequence $\mathbf{z}$ (i.e., $x_i = y_{\mu(i)}$, for every i for which $\mu(i)$ is defined).

The advisor finds and sends indices $i^\star \in [n]$ and $j^\star \in [m]$ such that no index $i < i^\star$ of $\mathbf{x}$ is mapped to an index $j > j^\star$ of $\mathbf{y}$. One option is that this is due to the fact that $\mu(i^\star) = j^\star$, but this does not have to be the case (e.g., if i^* is not mapped at all).[4] The specific choice of $i^\star$ and $j^\star$ will be crucial for bounding the client's runtime, but we defer the discussion of how they are chosen for the moment.

The indices $i^\star$ and $j^\star$ now define two LCS instances that the client solves directly (i.e., using the standard dynamic programming based algorithm for LCS): the first is $\mathbf{x}[1, i^\star - 1]$ vs. $\mathbf{y}[1, j^\star - 1]$ and the second is $\mathbf{x}[i^\star + 1, n]$ vs. $\mathbf{y}[j^\star + 1, n]$. Denote the solution found for the former by $\mathbf{z}_L$ and for the latter by $\mathbf{z}_R$.

Assuming that $i^\star$ was not mapped to $j^\star$, the client simply outputs $\mathbf{z}_L \circ \mathbf{z}_R$ and in case $i^\star$ was mapped to $j^\star$ it outputs $\mathbf{z}_L \circ \mathbf{x}[i^\star] \circ \mathbf{z}_R$. (Since we assume the advisor is trusted, it can simply specify whether or not $i^\star$ is mapped to $j^\star$.) It is straightforward to see that the output of the client is a valid solution – namely, a longest common subsequence for $\mathbf{x}$ and $\mathbf{y}$. The more important challenge is analyzing the efficiency of the protocol.

First note that if we use a time $c \cdot n \cdot m$ algorithm for LCS, for some constant c, we obtain a client that runs in time roughly: $c \cdot i^\star \cdot j^\star + c \cdot (n - i^\star) \cdot (m - j^\star)$. Note that for some values of $i^\star$ and $j^\star$ (e.g., $i^\star = j^\star = 0$) this gives no saving! Fortunately, we are able to show that there always exist $i^\star$ and $j^\star$ such that:

$$i^\star \cdot j^\star + (n - i^\star) \cdot (m - j^\star) \leq \frac{nm}{2}, \tag{1}$$

which in turn implies a factor 2 saving for the client.

To see that Equation (1) holds we consider two cases:

[4] Actually, it will be convenient to allow $i^\star$ and $j^\star$ to be real numbers rather than integers, but the reader can ignore this subtlety for the overview.

Case 1

There is an index $i < n/2$ that is mapped, via μ, to an index $j > n/2$. We fix $i^\star$ and $j^\star$ to be such indices. Note that since $i^\star$ is mapped to $j^\star$ they satisfy that $i < i^\star$ cannot be mapped to $j > j^\star$. Moreover:

$$i^\star \cdot j^\star + (n - i^\star) \cdot (m - j^\star) = \frac{1}{2}\big(n \cdot m + (n - 2i^\star) \cdot (m - 2j^\star)\big) \leq \frac{nm}{2},$$

where the equality can be verified by elementary arithmetic manipulations and the inequality follows from the fact that $n - 2i^\star < 0$ and $m - 2j^\star > 0$.

Case 2

There is no index $i < n/2$ that is mapped to $j^\star > m/2$. In such case, we define $i^\star = n/2$ and $j = n/2$. Note that by definition, there is no $i < i^\star$ that is mapped to $j > j_\star$. It is now straightforward that:

$$i^\star \cdot j^\star + (n - i^\star) \cdot (m - j^\star) = \frac{nm}{2}.$$

Thus, in both cases we obtain a factor 2 savings in the client's runtime as desired. In the full protocol, described below, we obtain a further savings by essentially recursing on both LCS instances rather than solving them directly.

2.2 The Minimum Cost String Alignment Problem

A cost function is a function $c : \mathbb{Z} \times \mathbb{Z} \to \mathbb{R}$. The two cost functions that we focus on are $c_{\mathsf{LCS}}(a, b) \stackrel{\text{def}}{=} a + b$ and $c_{\mathsf{Edit}}(a, b) \stackrel{\text{def}}{=} \max(a, b)$.

▶ **Definition 3.** *The* Minimum Cost String Alignment problem *with respect to a cost function c, denoted* MCSA$[c]$*, is a relation consisting of all pairs $\big((\mathbf{x}, \mathbf{y}), (\mathbf{i}, \mathbf{j})\big)$ for which, if $\mathbf{x} \in \Sigma^n$ and $\mathbf{y} \in \Sigma^m$,*

- $\mathbf{i} \subseteq [n]^\ell$, $\mathbf{j} \subseteq [m]^\ell$ *for some* $\ell \in \mathbb{Z}^+$*;*
- $i_1 < \cdots < i_\ell$ *and* $j_1 < \cdots < j_\ell$*;*
- $x_{i_k} = y_{j_k}$ *for all* $k \in [\ell]$*; and*
- $\mathbf{i}$ *and* $\mathbf{j}$ *minimize* $\sum_{k=1}^{\ell+1} c(i_k - i_{k-1} - 1, j_k - j_{k-1} - 1)$ *subject to the prior three constraints, where we define* $i_0 = j_0 = 0$*,* $i_{\ell+1} = n + 1$*, and* $j_{\ell+1} = m + 1$*.*

▶ Remark 4. To facilitate recursion, we will often consider MCSA instances where the strings $\mathbf{x}$ and $\mathbf{y}$ are most naturally viewed as substrings of larger strings, i.e. $\mathbf{x} = \mathbf{X}_I$ and $\mathbf{y} = \mathbf{Y}_J$. In these cases, we will view the symbols of $\mathbf{x}$ and $\mathbf{y}$ as indexed not by some $[n]$ and $[m]$, but by I and J respectively. The definition naturally extends to this setting, and we will frequently use this generalization.

The following two propositions show that LCS and Edit are special cases of MCSA with respect to different cost functions.

▶ **Proposition 5.** LCS *is the same as* MCSA$[c_{\mathsf{LCS}}]$.

Proof. For any common subsequence $x_{i_1} = y_{j_1}, \ldots, x_{i_\ell} = y_{j_\ell}$ of $\mathbf{x} \in \Sigma^n$ and $\mathbf{y} \in \Sigma^m$, define $i_0 = j_0 = 0$, $i_{\ell+1} = n + 1$, and $j_{\ell+1} = m + 1$ as in Definition 3

We have

$$\begin{aligned}
&\sum_{k=1}^{\ell+1} c_{\mathsf{LCS}}(i_k - i_{k-1} - 1, j_k - j_{k-1} - 1) \\
=&\sum_{k=1}^{\ell+1} i_k - i_{k-1} + j_k - j_{k-1} - 2 \\
=&i_{\ell+1} - i_0 + j_{\ell+1} - j_0 - 2\ell - 2 \\
=&n + m - 2\ell,
\end{aligned}$$

which is minimized when ℓ is maximized. ◀

▶ **Proposition 6.** Edit *is the same as* $\mathsf{MCSA}[c_{\mathsf{Edit}}]$.

Proof. Suppose $x_{i_1} = y_{j_1}, \ldots, x_{i_\ell} = y_{j_\ell}$ is a common subsequence of $\mathbf{x}, \mathbf{y}$. Define $i_0 = j_0 = 0$, $i_{\ell+1} = n+1$, and $j_{\ell+1} = m+1$ as in Definition 3. Then one way of editing $\mathbf{x}$ into $\mathbf{y}$ is by editing $\mathbf{x}_{(i_{k-1}, i_k)}$ into $\mathbf{y}_{(j_{k-1}, j_k)}$ for each $k \in [\ell+1]$, which takes at most $\max(i_k - i_{k-1} - 1, j_k - j_{k-1} - 1)$ operations.

On the other hand, suppose we are given a minimal sequence of operations $\mathcal{O} = (\mathsf{op}_1, \ldots, \mathsf{op}_d)$ that edits $\mathbf{x}$ into $\mathbf{y}$. This leads to a categorization of the symbols of $\mathbf{x}$ as unchanged, modified, or deleted; and a similar categorization of the symbols of $\mathbf{y}$ as unchanged from $\mathbf{x}$, modified from $\mathbf{x}$, or inserted. The unchanged symbols of $\mathbf{x}$ and of $\mathbf{y}$ form a common subsequence $x_{i_1} = y_{j_1}, \ldots, x_{i_\ell} = y_{j_\ell}$ and partition the original operation sequence into $\ell + 1$ subsequences $\mathcal{O}_1, \ldots, \mathcal{O}_{\ell+1}$ such that for each $k \in [\ell + 1]$, $\mathcal{O}_k$ edits $\mathbf{x}_{(i_{k-1}, i_k)}$ into $\mathbf{y}_{(j_{k-1}, j_k)}$. By the minimality of $\mathcal{O}$, each $\mathcal{O}_k$ must contain either only insertions and modifications or only deletions and modifications. Thus we have $|\mathcal{O}_k| = \max\big(i_k - i_{k-1} - 1, j_k - j_{k-1} - 1\big)$. ◀

The following proposition is a standard exercise in dynamic programming, whose proof we omit (see, e.g., [9]).

▶ **Proposition 7.** LCS *and* Edit *can be solved on input* $\mathbf{x}, \mathbf{y} \in \Sigma^n \times \Sigma^m$ *in* $O(n \cdot m)$ *time.*

The main technical lemma of Section 2 is a search delegation scheme for $\mathsf{MCSA}[c]$. running time.

▶ **Lemma 8.** *For every cost function* c *such that* $\mathsf{MCSA}[c]$ *is solvable in* $O(n \cdot m)$ *time and for every* $a = a(n, m)$*, there is an honest-advisor search delegation scheme for* $\mathsf{MCSA}[c]$ *that uses* a *bits of advice, and runs in time* $O\big(n + m + a + \frac{nm \log(nm)}{a}\big)$.

Furthermore assuming collision-resistant hash functions, this can be improved to an untrusted-advisor search delegation scheme with computational soundness.

2.3 Solution-Dependent Optimal Substructure

From this point on, we fix some cost function c such that $\mathsf{MCSA}[c]$ is solvable in $O(n \cdot m)$ time, and simply write MCSA in place of $\mathsf{MCSA}[c]$. We fix a specific c here because not all functions c necessarily have a $O(n \cdot m)$ time algorithm, however, both LCS and Edit do (see Proposition 7).

The main lemma in this section is that if some optimal alignment of $\mathbf{x}$ to $\mathbf{y}$ matches x_i with y_j, then any alignment of $\mathbf{x}$ to $\mathbf{y}$ that matches x_i with y_j is optimal if and only if it optimally aligns $\mathbf{x}_{<i}$ to $\mathbf{y}_{<j}$ and $\mathbf{x}_{>i}$ to $\mathbf{y}_{>j}$.

▶ **Proposition 9.** *Let* $\big((\mathbf{x},\mathbf{y}),(\mathbf{i},\mathbf{j})\big) \in \mathsf{MCSA}$ *with* $\mathbf{i} = (i_1,\ldots,i_\ell)$ *and* $\mathbf{j} = (j_1,\ldots,j_\ell)$.

For any $k \in [\ell]$, *any* $\ell_L, \ell_R \in \mathbb{Z}^+$, *any* $(\mathbf{i}'_L,\mathbf{j}'_L) \in [i_k-1]^{\ell_L} \times [j_k-1]^{\ell_L}$, *and any* $(\mathbf{i}'_R,\mathbf{j}'_R) \in [i_k+1,n]^{\ell_R} \times [j_k+1,m]^{\ell_R}$, *we have*

$$\big(\mathbf{i}'_L \circ i_k \circ \mathbf{i}'_R,\ \mathbf{j}'_L \circ j_k \circ \mathbf{j}'_R\big) \in \mathsf{MCSA}\big(\mathbf{x},\mathbf{y}\big)$$

if and only if $(\mathbf{i}'_L,\mathbf{j}'_L) \in \mathsf{MCSA}(\mathbf{x}_{<i_k},\mathbf{y}_{<j_k})$ *and* $(\mathbf{i}'_R,\mathbf{j}'_R) \in \mathsf{MCSA}(\mathbf{x}_{>i_k},\mathbf{y}_{>j_k})$.

Proof. By definition, the "total cost" of $\big(\mathbf{i}'_L \circ i_k \circ \mathbf{i}'_R,\ \mathbf{j}'_L \circ j_k \circ \mathbf{j}'_R\big)$ with respect to $(\mathbf{x},\mathbf{y})$ is the sum of the cost of $(\mathbf{i}'_L,\mathbf{j}'_L)$ with respect to $(\mathbf{x}_{<i_k},\mathbf{y}_{<j_k})$ and the cost of $(\mathbf{i}'_R,\mathbf{j}'_R)$ with respect to $(\mathbf{x}_{>i_k},\mathbf{y}_{>j_k})$. ◀

2.4 Our Protocol

Following the discussion in the introduction, we assume first that the advisor is honest. Suppose we are given an input $(\mathbf{x},\mathbf{y}) \in \Sigma^n \times \Sigma^m$ and have an advice budget of $a = a' \cdot \log(nm)$.

The advice in our protocol consists of an ordered rooted binary tree $\mathcal{T}$ with $O(a)$ vertices, such that:

- Every vertex v of $\mathcal{T}$ is labeled with a pair of intervals $(v.I, v.J)$ with $v.I \subseteq [n]$ and $v.J \subseteq [m]$. The root v_0 has $v_0.I = [n]$ and $v_0.J = [m]$.
- Every internal vertex v of $\mathcal{T}$ is additionally labeled with either:
  - (Case 1) $(v.i, v.j) \in v.I \times v.J$; or
  - (Case 2) $(v.i_1, v.i_2, v.j_1, v.j_2) \in v.I^2 \times v.J^2$.

  such that if $v.L$ and $v.R$ denote the left and right children of v, and if $(\mathbf{i}_L,\mathbf{j}_L)$ and $(\mathbf{i}_R,\mathbf{j}_R)$ are arbitrary elements of $\mathsf{MCSA}(\mathbf{x}_{v.L.I},\mathbf{y}_{v.L.J})$ and $\mathsf{MCSA}(\mathbf{x}_{v.R.I},\mathbf{y}_{v.R.J})$ respectively, then either:
  - (Case 1) $(\mathbf{i}_L \circ v.i \circ \mathbf{i}_R,\ \ \mathbf{j}_L \circ v.j \circ \mathbf{j}_R) \in \mathsf{MCSA}(\mathbf{x}_{v.I},\mathbf{y}_{v.J})$; or
  - (Case 2) $(\mathbf{i}_L \circ v.i_1 \circ v.i_2 \circ \mathbf{i}_R,\ \ \mathbf{j}_L \circ v.j_1 \circ v.j_2 \circ \mathbf{j}_R) \in \mathsf{MCSA}(\mathbf{x}_{v.I},\mathbf{y}_{v.J})$.

  This property gives a means to compute a solution to $\mathsf{MCSA}(\mathbf{x}_{v.I},\mathbf{y}_{v.J})$ in $O(1)$ time given any solutions to $\mathsf{MCSA}(\mathbf{x}_{v.L.I},\mathbf{y}_{v.L.J})$ and $\mathsf{MCSA}(\mathbf{x}_{v.R.I},\mathbf{y}_{v.R.J})$. Thus if we have solutions to $\mathsf{MCSA}(\mathbf{x}_{\ell.I},\mathbf{y}_{\ell.J})$ for all leaf nodes $\ell \in \mathcal{T}$, then we can compute a solution to $\mathsf{MCSA}(\mathbf{x},\mathbf{y})$ in time $O(|\mathcal{T}|)$.
- For every internal vertex v of $\mathcal{T}$, $v.L.I$ and $v.R.I$ are disjoint subsets of $v.I$. Similarly $v.L.J$ and $v.R.J$ are disjoint subsets of $v.J$, such that

$$|v.L.I| \cdot |v.L.J| + |v.R.I| \cdot |v.R.J| \le \frac{|v.I| \cdot |v.J|}{2}. \tag{2}$$

- Every vertex v of $\mathcal{T}$ with depth less than $\lfloor \log a' \rfloor$ satisfies $\min\big(|v.I|, |v.J|\big) = O(1)$.

2.4.1 Solving MCSA(x, y) given $\mathcal{T}$

As alluded to in the second property of $\mathcal{T}$, the client directly solves $\mathsf{MCSA}\big(\mathbf{x}_{\ell.I},\mathbf{y}_{\ell.J}\big)$ for every leaf node ℓ in $\mathcal{T}$. This takes time

$$O\Big(\sum_{\ell} |\ell.I| \cdot |\ell.J|\Big).$$

We split this sum into two sums: one over leaves of depth less than $\lfloor \log a' \rfloor$ and the other over leaves of depth at least $\lfloor \log a' \rfloor$.

- (Low-Depth Leaves) For low depth leaves ℓ, we bound $|\ell.I| \cdot |\ell.J|$ by $O(|\ell.I| + |\ell.J|)$. This is justified by the last property of $\mathcal{T}$, which asserts that either $|\ell.I|$ or $|\ell.J|$ is a constant. Now, we bound

$$\sum_{\text{low-depth } \ell} |\ell.I| + |\ell.J| \leq n + m,$$

  using the fact that $v.L.I$ and $v.R.I$ (respectively $v.L.J$ and $v.R.J$) are disjoint subsets of $v.I$ (respectively $v.J$).
- (High-Depth Leaves) By Equation (2), we know that

$$\sum_{\text{depth-}d \text{ leaves } \ell} |\ell.I| \cdot |\ell.J| \leq n \cdot m \cdot 2^{-d},$$

  and thus

$$\sum_{d \geq \lfloor \log a \rfloor} \sum_{\text{depth-}d \text{ leaves } \ell} |\ell.I| \cdot |\ell.J| \leq \frac{2nm}{a'}.$$

Finally, using the second property of $\mathcal{T}$, the client processes its leaf solutions into a solution to $\mathsf{MCSA}(\mathbf{x}, \mathbf{y})$. This takes time $O(|\mathcal{T}|) = O(a)$.

In total, the client's runtime is

$$O\left(n + m + a + \frac{nm}{a \log(nm)}\right).$$

2.4.2 Constructing $\mathcal{T}$

We construct $\mathcal{T}$ iteratively. On input $\mathbf{x} \in \Sigma^n, \mathbf{y} \in \Sigma^m$, start with a tree that consists only of a root vertex labeled with $I = [n]$ and $J = [m]$.

We say that a leaf vertex ℓ is $\mathsf{expandable}$ if either $\min(|\ell.I|, |\ell.J|) \geq 2$. For any interval $[a, b] \subseteq \mathbb{Z}$, let $\mathsf{middle}(I)$ denote $(a + b)/2$.

While there is an expandable leaf of $\mathcal{T}$:

1. Pick an expandable leaf of minimum-depth, and call it ℓ.
2. Compute $(\mathbf{i}, \mathbf{j}) \in \mathsf{MCSA}(\mathbf{x}_{\ell.I}, \mathbf{y}_{\ell.J})$.
3. Do either of the following (at least one will be applicable), defining i_0 and j_0 as $\min(I) - 1$ and $\min(J) - 1$ respectively, and defining $i_{|\mathbf{i}|+1}$ and $j_{|\mathbf{i}|+1}$ as $\max(I) + 1$ and $\max(J) + 1$ respectively:
   - **Case 1**: For some k, (i_k, j_k) "crosses" $(\mathsf{middle}(\ell.I), \mathsf{middle}(\ell.J))$. That is, either $i_k \leq \mathsf{middle}(\ell.I)$ and $j_k \geq \mathsf{middle}(\ell.J)$, or $i_k \geq \mathsf{middle}(\ell.I)$ and $j_k \leq \mathsf{middle}(\ell.J)$.
     In this case, add child nodes $v.L$ and $v.R$ to v, with $v.L.I = v.I \cap (-\infty, i_k)$, $v.L.J = v.J \cap (-\infty, j_k)$, $v.R.I = v.I \cap (i_k, \infty)$, and $v.R.J = v.J \cap (j_k, \infty)$. Set $v.i = i_k$ and $v.j = j_k$.
   - **Case 2:** For some k, $i_k \leq \mathsf{middle}(v.I) < i_{k+1}$ and $j_k \leq \mathsf{middle}(v.J) < j_{k+1}$.
     In this case, add child nodes $v.L$ and $v.R$ to v, with $v.L.I = v.I \cap (-\infty, i_k)$, $v.L.J = v.J \cap (-\infty, j_k)$, $v.R.I = v.I \cap (i_{k+1}, \infty)$, and $v.R.J = v.J \cap (j_{k+1}, \infty)$. Set $v.i_1 = i_k$, $v.i_2 = i_{k+1}$, $v.j_1 = j_k$, and $v.j_2 = j_{k+1}$.

   To see that one of the above cases will always be possible, let k^* and $\hat{k}$ be such that $i_{k^*} \leq \mathsf{middle}(I) < i_{k^*+1}$ and $j_{\hat{k}} \leq \mathsf{middle}(J) < j_{\hat{k}+1}$. If $k^* = \hat{k}$ then we are in case 2. Otherwise we are in case 1.

   The labelings of the children of v ensure the second property of $\mathcal{T}$ by Proposition 9. The last property of $\mathcal{T}$ is guaranteed by the fact that we always expand a lowest-depth expandable leaf.

We now turn to the third property of $\mathcal{T}$, i.e. establishing that Equation (2) holds. This is easy to see in Case 2. In Case 1, observe that either $|v.L.I| \leq |v.I|/2$ and $|v.L.J| \geq |v.J|/2$ or $|v.L.I| \geq |v.I|/2$ and $|v.L.J| \geq |v.J|/2$. We then have

$$\begin{aligned}|v.L.I| \cdot |v.L.J| + |v.R.I| \cdot |v.R.J| &\leq |v.L.I| \cdot |v.L.J| + \big(|v.I| - |v.L.I|\big) \cdot \big(|v.J| - |v.L.J|\big) \\ &= \frac{|v.I| \cdot |v.J|}{2} + \frac{1}{2} \cdot \big(|v.I| - 2|v.L.I|\big) \cdot \big(|v.J| - 2|v.L.J|\big) \\ &\leq \frac{|v.I| \cdot |v.J|}{2},\end{aligned}$$

as desired.

2.4.3 Advisor Efficiency

We note that the advisor's messages can be computed in linear time given any $(\mathbf{i}, \mathbf{j}) \in \mathsf{MCSA}(\mathbf{x}, \mathbf{y})$. Thus the advisor's running time is dominated by the cost of finding such a $(\mathbf{i}, \mathbf{j})$, which by Proposition 7 takes time $O(n \cdot m)$.

3 Parsing Context-Free Grammars

We start with background on context-free grammars in Section 3.1. Then, in Section 3.2 we state our main result and prove it in Section 3.3.

3.1 Context-Free Grammars

We first define context-free grammars. Parts of the following overview are directly based on [12]. See the standard textbook [15] for more detailed background.

▶ **Definition 10** (Context-free grammar). *A* context-free grammar *is a tuple $G = (V, \Sigma, R, A_{\mathsf{start}})$ such that V is a (finite) set of symbols, referred to as* variables*; Σ is a (finite) set of symbols, referred to as* terminals*; $R \subseteq V \times (V \cup \Sigma)^*$ is a (finite) relation, where each $(A, \alpha) \in R$ is referred to as a* production rule *and is denoted by $A \to \alpha$; $A_{\mathsf{start}} \in V$ is a variable that is referred to as the* start variable*.*

Let $G = (V, \Sigma, R, A_{\mathsf{start}})$ be a context-free grammar, and let $\alpha, \beta \in (V \cup \Sigma)^*$ be strings of terminals and variables. We say that α directly yields β, denoted by $\alpha \Rightarrow \beta$, if there exists a production rule $A \to \gamma$ in R such that β is obtained from α by replacing exactly one occurrence of the variable A in α with the string $\gamma \in (V \cup \Sigma)^*$. We say that α yields β, denoted $\alpha \overset{*}{\Rightarrow} \beta$ if there exists a finite sequence of strings $\alpha_0, \ldots, \alpha_k \in (V \cup \Sigma)^*$ such that $\alpha_0 = \alpha$, $\alpha_k = \beta$, and $\alpha_0 \Rightarrow \ldots \Rightarrow \alpha_k$.

A grammar is said to be in *Chomsky normal form* if all of the production rules are of the form: (1) $A \to B \circ C$, where A, B and C are variables, (2) $A \to a$, where A is a variable and a is a terminal, or (3) $A \to \varepsilon$, where A is a variable and ε denotes the empty string. We note that every context-free grammar can be transformed into Chomsky normal form (with at most a quadratic blowup in the size of the grammar).

3.1.1 Derivation Tree

Let $G = (V, \Sigma, R, A_{\mathsf{start}})$ be a context-free grammar. For $A \in V$ and $\alpha \in (V \cup \Sigma)^*$, a derivation tree, corresponding to the derivation[5] $A \overset{*}{\Rightarrow} \alpha$, is a rooted, directed, ordered, and labeled tree T (with edges oriented away from the root) that satisfies the following properties:

[5] The literature usually focuses on derivations trees for words composed of terminals only whereas we allow for a mix of variables and terminals.

- Each internal vertex is labeled by some variable, and the root is labeled by the variable A.
- Each leaf is labeled by a terminal or variable, where the i^{th} leaf is labeled by the i^{th} symbol of α.
- For every internal vertex v, if v is labeled by A' and its children are labeled by $\alpha_1, \ldots, \alpha_d \in (V \cup \Sigma)$ (where d denotes the number of children of v), then the production rule $A' \to \alpha_1 \circ \cdots \circ \alpha_k$ must belong to R, where $\circ$ denotes concatenation.

Note that for every derivation $A \stackrel{*}{\Rightarrow} \alpha$ there exists (at least one) corresponding derivation tree.

3.1.2 Parsing

The *parsing* problem is, given a grammar G and a word $w \in \Sigma^n$, to find a derivation tree corresponding to the fact that G derives w, or output $\perp$ if no such tree exists. A more general problem also considered in the literature is outputting *all* of the derivation trees corresponding to w but we focus here on the simpler task of finding *some* derivation tree. Thus, a parser for a grammar G is an algorithm that given as input a word w outputs a corresponding derivation tree, or $\perp$ if no such tree exists. As customary, we view the grammar as constant size and measure the complexity as a function of the input length.

We rely on the following result due to Valiant [20]:

▶ **Theorem 11** ([20]). *Every context-free grammar in Chomsky normal form has a time $O(n^{\omega_b})$ parser.*

Here and throughout, ω_b is the exponent for Boolean matrix multiplication (i.e., matrix multiplication in which the inner product operations is replaced with an OR of ANDs). The current known upper bound on ω_b is equal to the best known bound on ω [3] , the standard exponent for matrix multiplication.

Trees and the Lewis-Stearns-Hartmanis Lemma

In this section we only consider trees that are rooted, directed, and ordered (such as derivation trees defined above). Thus, throughout this section, whenever we say tree, we mean a rooted, directed, and ordered tree (with edges oriented away from the root). Note that the fact that the tree is ordered induces an ordering of its leaves. We define the arity of a tree to be the maximal number of children of any vertex in the tree. We follow the data-structure literature and define a subtree of a tree T as a tree consisting of a node in T and all of its descendants in T.[6] We use $L(T)$ to denote the number of leaves in the tree T.

The classic Lewis-Stearns-Hartmanis Lemma [18] shows that every binary tree on n leaves has a subtree with between $n/3$ and $2n/3$ leaves. We use here a more general form of this lemma, given by Goldreich *et al.* [12] who show that for every desired parameter t, any binary tree has a subtree with approximately t leaves (actually [12] further generalize to trees of different constant arity):

▶ **Lemma 12** ([12, Lemma 2.5]). *Let T be a binary tree and let $t \in [L(T)]$. Then, there exists a subtree T' of T with $L(T') \in [t/2, t]$ leaves.*

(The Lewis-Stearns-Hartmanis lemma corresponds to the special case when $t = 2n/3$.)

[6] This definition differs from the graph-theoretic definition that defines a subtree as any connected subgraph of a tree. For example, the root of a tree is a subtree in the graph theoretic sense but not according to our definition (unless the tree has exactly one vertex).

3.2 Delegating Context-Free Parsing

▶ **Theorem 13.** *Let G be a context-free grammar in Chomsky normal form. Then, for every parameter $a = a(n)$, there exists a search delegation scheme for the parsing problem for G, where the client runs in time $O(n^{\omega_b}/a^{\omega_b-1} + a \cdot \log n)$ and the communication complexity is $O(a \cdot \log n)$.*

For example, taking $a = n/\mathsf{polylog}(n)$, we obtain a quasi-linear time client with sub-linear communication. Alternatively, taking $a = n^{\frac{\omega_b-2}{\omega_b-1}} < \sqrt{n}$ we obtain a quadratic-time client with $\tilde{O}(\sqrt{n})$ communication.

3.3 Proof of Theorem 13

Let $G = (V, \Sigma, R, A_{\mathsf{start}})$ be a context-free grammar in Chomsky normal form. Recall that our task is to construct a delegation scheme for finding a derivation tree for a given word w derived by G. However, to facilitate recursion, we will actually solve a more general problem where the word $w \in (\Sigma \cup V)^*$ can be a mix of terminals and variables.

Consider a derivation tree T for $w \in (\Sigma \cup V)^*$. Since the grammar is in Chomsky normal form, the tree is binary and so for a given parameter t, Lemma 12 guarantees that T has a subtree T' of size roughly t. More precisely, a subtree T' with between $t/2$ and t leaves. Let x' be the substring of x corresponding to the tree T'.

We first construct an honest-advisor delegation scheme and then show how to deal with an untrusted advisor. In the scheme, the advisor first finds x' within x – that is, indices i and j such that $x' = x[i+1, \ldots, j]$). The advisor also finds the variable A associated with the root of T' in the derivation tree T. The advisor sends (i, j, A) to the client. The client is now left with two tasks:

- Find the derivation tree T' corresponding to $A \stackrel{*}{\Rightarrow} x'$.
- Find a derivation tree T'' corresponding to $A_{\mathsf{start}} \stackrel{*}{\Rightarrow} x[1, \ldots, i-1] \circ S \circ x[j+1, \ldots, n]$.

We solve the first of these directly, that is, by invoking a context-free parser (as in Theorem 11). The second problem is solved recursively. Note that two resulting derivation trees T' and T'' can be easily grafted together to construct a derivation tree corresponding to $A_{\mathsf{start}} \stackrel{*}{\Rightarrow} x$. However, some care needs to be given also for this task, since we do not want to process the entire tree T' again just for grafting. Rather, we ensure that the tree is represented using a data structure so that the grafting takes $O(1)$ time (e.g., using pointers). We also note that the base case of the recursion is solved by directly invoking a context-free parser.

Denote by $W(n, a)$ the running time of the client if we allow a recursive calls (and note that the corresponding communication complexity is $O(a \cdot \log n)$. Given the time $c \cdot n^{\omega_b}$ context-free parser of Theorem 11, where c is some constant, we have:

$$W(n, a) \leq c \cdot t^{\omega_b} + W(n - t/2, a - 1) + O(\log n). \tag{3}$$

Expanding, we see that any solution to Equation (3) must satisfy

$$W(n, a) \leq c \cdot a \cdot t^{\omega_b} + c \cdot (n - a \cdot t/2)^{\omega_b} + O(a \cdot \log n).$$

Setting the parameter $t = 2n/a$, we obtain:

$$W(n, a) \leq c \cdot a \cdot (2n/a)^{\omega_b} + O(a \cdot \log n) = O(n^{\omega_b}/a^{\omega_b-1} + a \cdot \log n).$$

3.3.1 Coping with cheating advisors

In order to deal with a dishonest advisor, we note that given a candidate derivation tree, there is a linear (in the size of the tree) time algorithm that checks that its validity (by simply checking that each vertex is consistent with the grammar). Thus, we can easily transform our protocol to handle dishonest advisors by having the client check its answer and outputting $\perp$ in case the generated tree is invalid.

4 k-SAT

In this section, we construct a search delegation scheme for k-SAT, where k is some constant. Recall that an instance of k-SAT is a CNF formula on n variables $(x_1, \ldots, x_n)$ with m clauses, each of which is a disjunction of at most k literals (variables or their negations).

A trivial delegation scheme for k-SAT on a given formula $\varphi : \{0,1\}^n \to \{0,1\}$ is to simply send the first a bits of a satisfying assignment for φ. This reduces the number of variables to $n - a$, so if $T_k(n)$ denotes the time to solve an n-variable k-SAT instance, this hint reduces the client's running time to $T_k(n-a)$. In particular, if $T_k(n) = 2^{c_k \cdot n}$ for some $c_k < 1$, then a bits of hint yield a factor $2^{c_k \cdot a}$ speedup.

We achieve an improved factor 2^a speedup for a specific k-SAT algorithm due to [19], which we henceforth refer to as PPSZ. PPSZ is relatively simple, and has a running time that is close to the state of the art algorithm due to Hansen *et al.* [14].

▶ **Theorem 14.** *Let $(c_k)_{k \in \mathbb{Z}^+}$ be constants such that the analyzed running time of the PPSZ algorithm for k-SAT, with success probability*[7] *$\frac{1}{2}$, is $2^{c_k n + o(n)}$. Then, for every $a = a(n)$ and $\kappa = \kappa(n) \leq n^{O(1)}$, there exists a search delegation scheme in which the advisor sends a bits and the client's run-time is $2^{c_k \cdot n - a + o(n)}$, assuming the advisor and client have access to a shared $\mathsf{poly}(n)$-length random string. The completeness error is $2^{-\kappa}$ and the soundness error is 0.*

To prove Theorem 14, roughly speaking, we observe that PPSZ can be viewed as a nondeterministic polynomial-time algorithm that uses $c_k n$ bits of nondeterminism - in other words, an exhaustive search for a $c_k n$-bit string. Such an emulation clearly takes $\tilde{O}(2^{c_k n})$ time. The hint in our protocol then consists of the first a bits of this nondeterminism rather than the first a bits of a satisfying assignment for φ.

This description is not completely accurate – PPSZ is actually a randomized algorithm. Each choice of randomness defines an exhaustive search problem such such that the exhaustive search has noticeable probability of yielding a satisfying assignment for φ. We describe PPSZ in a way that elucidates this structure in Appendix A.

Finally, we can deal with an untrusted advisor by simply verifying that at the end we have actually found a satisfying assignment to φ.

5 From Honest Advisor to Dishonest Advisor

In this section we describe a simple transformation from an honest-advisor search delegation to a full-fledged one (i.e., secure against an untrusted advisor).

[7] Since we can amplify success probability by repetition, c_k would be the same if we we required success probability as small as $1/\mathsf{poly}(n)$ or as large as $1 - 2^{-\mathsf{poly}(n)}$.

▶ **Lemma 15.** *Suppose that the relation $R \subseteq \{0,1\}^n \times \{0,1\}^m$ has an honest-advisor search delegation scheme with communication $c_{\mathsf{srch}} = c_{\mathsf{srch}}(n,m)$ and client running time $t_{\mathsf{srch}} = t_{\mathsf{srch}}(n,m)$. Suppose also that membership in the relation R can be verified by an interactive proof with communication $c_{\mathsf{prf}} = c_{\mathsf{prf}}(n,m)$ and verifier running time $t_{\mathsf{prf}} = t_{\mathsf{prf}}(n,m)$. Then, R has a full-fledged search delegation scheme with $c_{\mathsf{srch}} + c_{\mathsf{prf}}$ communication and $t_{\mathsf{srch}} + t_{\mathsf{prf}}$ client run-time.*

In case the interactive proof for R is only computationally sound, then the resulting delegation scheme is also only computationally sound.

Proof. Follows immediately by first running the honest-advisor protocol and then checking the solution using the interactive proof. ◀

For example, using Kilian's [16] celebrated argument-system we obtain the following corollary:

▶ **Corollary 16.** *Suppose that the relation $R \subseteq \{0,1\}^n \times \{0,1\}^m$ is decidable in polynomial-time and has an honest-advisor search delegation scheme with communication $c_{\mathsf{srch}} = c_{\mathsf{srch}}(n,m)$ and client running time $t_{\mathsf{srch}} = t_{\mathsf{srch}}(n,m)$. Then, assuming that there exist collision-resistant hash functions, the relation R has a computationally sound delegation scheme with communication $c_{\mathsf{srch}} + \mathsf{poly}(\kappa, \log(n+m))$ and verifier runtime $\tilde{O}(n+m) + \mathsf{poly}(\kappa, \log(n+m))$, where κ denotes a cryptographic security parameter.*

References

1. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.14.
2. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is valiant's parser. *SIAM J. Comput.*, 47(6):2527–2555, 2018. doi:10.1137/16M1061771.
3. Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 522–539. SIAM, 2021. doi:10.1137/1.9781611976465.32.
4. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part I*, volume 6198 of *Lecture Notes in Computer Science*, pages 152–163. Springer, 2010. doi:10.1007/978-3-642-14165-2_14.
5. Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.*, 47(3):1087–1097, 2018. doi:10.1137/15M1053128.
6. Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006), 16-20 July 2006, Prague, Czech Republic*, pages 252–260. IEEE Computer Society, 2006. doi:10.1109/CCC.2006.6.
7. Lijie Chen, Shafi Goldwasser, Kaifeng Lyu, Guy N. Rothblum, and Aviad Rubinstein. Fine-grained complexity meets IP = PSPACE. *CoRR*, abs/1805.02351, 2018. arXiv:1805.02351.

8 Kai-Min Chung, Yael Tauman Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 483–501. Springer, 2010. doi:10.1007/978-3-642-14623-7_26.

9 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

10 Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. *IACR Cryptol. ePrint Arch.*, page 547, 2009. URL: http://eprint.iacr.org/2009/547.

11 Oded Goldreich. *Computational complexity – A conceptual perspective*. Cambridge University Press, 2008. doi:10.1017/CBO9780511804106.

12 Oded Goldreich, Tom Gur, and Ron D. Rothblum. Proofs of proximity for context-free languages and read-once branching programs. *Inf. Comput.*, 261:175–201, 2018. doi:10.1016/j.ic.2018.02.003.

13 Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. doi:10.1137/0218012.

14 Thomas Dueholm Hansen, Haim Kaplan, Or Zamir, and Uri Zwick. Faster k-sat algorithms using biased-ppsz. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 578–589. ACM, 2019. doi:10.1145/3313276.3316359.

15 John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

16 Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732, 1992. doi:10.1145/129712.129782.

17 Lillian Lee. Fast context-free grammar parsing requires fast boolean matrix multiplication. *J. ACM*, 49(1):1–15, 2002. doi:10.1145/505241.505242.

18 Philip M. Lewis, Richard Edwin Stearns, and Juris Hartmanis. Memory bounds for recognition of context-free and context-sensitive languages. In *SWCT (FOCS)*, pages 191–202, 1965. doi:10.1109/FOCS.1965.14.

19 Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k-sat. *J. ACM*, 52(3):337–364, 2005. doi:10.1145/1066100.1066101.

20 Leslie G. Valiant. General context-free recognition in less than cubic time. *J. Comput. Syst. Sci.*, 10(2):308–315, 1975. doi:10.1016/S0022-0000(75)80046-8.

A PPSZ with Hints: Details

In this section we provide details about PPSZ and its analysis. We then discuss how a hint can be used to achieve optimal savings up to an additive log factor. We focus our attention on k-SAT instances that are *sparse* (the number of clauses is linear in the number of variables) and have *unique solutions* because when $k \geq 5$ there is a randomized reduction from general k-SAT to this restricted subset of instances [6].

PPSZ Overview

PPSZ has the following high-level structure. It first performs a polynomial-time "conditioning" step on the input formula k-CNF formula φ that produces an equivalent formula φ_s that has more clauses. Then PPSZ runs Search (see Algorithm 1) on φ_s. This produces I random candidate values of $(\pi, y) \in S_n \times \{0,1\}^n$ (where, as usual, S_n denotes the group of

permutations over $[n]$). Each pair is fed into the algorithm MODIFY (see Algorithm 2) until MODIFY produces a satisfying assignment to φ_s (and hence to φ). At a high level, MODIFY goes through the variables of φ_s one by one (in an order specified by π), and iteratively simplifies φ_s by setting the i^{th} variable to y_i. If at any stage there is a "unital" clause (a clause with exactly one variable x), then x is set to 1 if the clause is (x) and to 0 if the clause is $(\neg x)$. When this happens, we say that x is forced.

PPSZ Analysis

Let z denote the unique satisfying assignment to φ, and let $Forced(\varphi, \pi, y)$ denote the set of forced variables when running MODIFY(φ, π, y).

PPSZ prove that for $\pi \leftarrow S_n$, we have

$$\mathbb{E}_\pi\left[\left|Forced(\varphi_s, \pi, z)\right|\right] \geq (1 - c_k)n,$$

where c_k is some constant.

Call $\pi \in S_n$ good if $\left|Forced(\varphi_s, \pi, z)\right| \geq (1 - c_k)n - 1$. If SEARCH happens to sample a good π, then there is a $\approx 2^{-c_k n}$ probability that y will be sampled to agree with z outside of $Forced(\varphi_s, \pi, z)$, which causes MODIFY to terminate with a satisfying assignment for φ.

Finally, a random π is good with probability at least $1/n$. To see this, let F denote $\left|Forced(\varphi_s, \pi, z)\right|$. Markov's inequality then implies (because F is $[0, n]$-valued) that $\Pr\left[F \geq \mathbb{E}[F] - 1\right] \geq 1/n$.

Thus after $I = 2^{c_k n} \cdot \mathsf{poly}(n)$ trials, MODIFY will with overwhelming probability output a satisfying assignment for φ.

Algorithm 1 The SEARCH Algorithm from [19].

```
SEARCH(formula φ, integer I):
for i ∈ [1, I] do
    pick y uniformly at random {0,1}^n
    pick π uniformly at random
    z=MODIFY(φ, π, y)
    if z satisfies φ then
        return z
    end if
end for
return Unsatisfied
```

▶ **Lemma 17.** *Let c_k denoting the constant in the exponent of the running time of the PPSZ algorithm for k-SAT. Let $\kappa = \kappa(n) \leq n^{O(1)}$ denote a statistical security parameter.*

There for any $a = a(n)$, there is an honest-advisor search delegation scheme for k-SAT with:

- *completeness error $2^{-\kappa}$;*
- *a bits of communication;*
- *$\mathsf{poly}(n)$ bits of shared randomness; and*
- *client running time of $2^{c_k n - a} \cdot \mathsf{poly}(n)$.*

Proof Sketch. In our protocol, we assume that the advisor and client have access to shared randomness, which determines a sequence of permutations $\pi_1, \pi_2, \ldots \in S_n$. The advisor first sends the client an i such that π_i is good. With all but $2^{-\kappa}$ probability, i is $O(\kappa \log n)$, so this constitutes just $\log \kappa + \log \log n + O(1)$ bits of communication.

Algorithm 2 The MODIFY algorithm from [19].

MODIFY(formula φ, permutation π, assignment y):
$\varphi_0 = \varphi$
for $i \in [1, n]$ **do**
  if φ_{i-1} contains the unit clause $(x_{\pi[i]})$ **then**
    $z_{\pi[i]} = 1$
  else if φ_{i-1} contains the unit clause $(\bar{x}_{\pi[i]})$ **then**
    $z_{\pi[i]} = 0$
  else
    $x_i = y_i$
  end if
  $\varphi_i = \varphi_{i-1}$ with $x_{\pi_i} = z_{\pi_i}$
end for
return z

Algorithm 3 The MODIFY' algorithm.

MODIFY'(formula φ, permutation π, settings y):
$\varphi_0 = \varphi$
$j = 0$
for $i \in [1, n]$ **do**
  if φ_{i-1} contains the unit clause $(x_{\pi[i]})$ **then**
    $z_{\pi[i]} = 1$
  else if φ_{i-1} contains the unit clause $(\bar{x}_{\pi[i]})$ **then**
    $z_{\pi[i]} = 0$
  else
    $x_i = y_j$
    $j = j + 1$
  end if
  $\varphi_i = \varphi_{i-1}$ with $x_{\pi_i} = z_{\pi_i}$
end for
return z

Next, let $j_1, \ldots, j_{n'} \in [n]$ denote indices such that on input (φ_s, π_i, z), MODIFY assigns unforced values to $x_{j_1}, \ldots, x_{j_{n'}}$ (in that order). Because π_i is good, we have $n' \leq c_k n + 1$. If the total remaining advice budget is a, then the advisor sends $z_{j_1}, \ldots, z_{j_a}$ and n'.

Upon receiving $\big((y_1, \ldots, y_a), n'\big)$, the client then repeatedly samples $y_{a+1}, \ldots, y_{n'}$, computes $z := \text{MODIFY}'\big(\varphi_s, \pi_i, (y_1, \ldots, y_{n'})\big)$ and hopes for a successful trial, i.e. one in which z is a satisfying assignment for φ (this happens with probability at least $2^{-c_k n - 1}$ over the choice of $y_{a+1}, \ldots, y_{n'}$). After $\approx \kappa \cdot 2^{c_k n}$ trials, the client will have at least one successful trial with all but $2^{-\kappa}$ probability. ◀