On the Complexity of Trial and Error

Xiaohui Bei
Nanyang Technological University
Singapore
xhbei@ntu.edu.sg

Ning Chen
Nanyang Technological University
Singapore
ningc@ntu.edu.sg

Shengyu Zhang
The Chinese Univ. of Hong Kong
Hong Kong
syzhang@cse.cuhk.edu.hk

ABSTRACT

Motivated by certain applications from physics, biochemistry, economics, and computer science in which the objects under investigation are unknown or not directly accessible because of various limitations, we propose a trial-and-error model to examine search problems in which inputs are *unknown*. More specifically, we consider constraint satisfaction problems $\bigwedge_i C_i$, where the constraints C_i are hidden, and the goal is to find a solution satisfying all constraints. We can adaptively propose a candidate solution (i.e., *trial*), and there is a verification oracle that either confirms that it is a valid solution, or returns the index i of a violated constraint (i.e., *error*), with the exact content of C_i still hidden.

We studied the time and trial complexities of a number of natural CSPs, summarized as follows. On one hand, despite the seemingly very little information provided by the oracle, efficient algorithms do exist for Nash, Core, Stable Matching, and SAT problems, whose unknown-input versions are shown to be as hard as the corresponding known-input versions up to a factor of polynomial. The techniques employed vary considerably, including, e.g., order theory and the ellipsoid method with a strong separation oracle.

On the other hand, there are problems whose complexities are substantially increased in the unknown-input model. In particular, no time-efficient algorithms exist for Graph Isomorphism and Group Isomorphism (unless **PH** collapses or $\mathbf{P} = \mathbf{NP}$). The proofs use quite nonstandard reductions, in which an efficient simulator is carefully designed to simulate a desirable but computationally unaffordable oracle.

Our model investigates the value of input information, and our results demonstrate that the lack of input information can introduce various levels of extra difficulty. The model accommodates a wide range of combinatorial and algebraic structures, and exhibits intimate connections with (and hopefully can also serve as a useful supplement to) certain existing learning and complexity theories.

Categories and Subject Descriptors

F.0 [**Theory of Computation**]: General

STOC'13, June 1 - 4, 2013, Palo Alto, California, USA.

Keywords

Complexity, query, constraint satisfaction problem, unknown input, trial and error

1. INTRODUCTION

In a broad sense, computer science studies computation and other information processing tasks. Theoretical computer science, in particular, focuses on understanding the ultimate power and limits of computation in various models. A central question in theoretical computer science is to find the minimum cost of an algorithm for computing a function f on inputs x. Algorithm design and computational complexity analysis assume that the input is given explicitly. However, in many scenarios, we actually *lack input information*.

- In a normal-form game, it is usually assumed that the payoff function of every player is given explicitly (or can be computed easily in a certain way). However, in many circumstances, players do not necessarily know their payoffs or possible strategies, particularly when they are exploring a new environment (such as a new business model). In some cases, they may not even know the number of other players, let alone their strategies [24].
- In a two-sided matching marketplace, every individual has a preference over the agents of the other side. However, the individuals themselves may not know their (precise) preferences. For example, in a job market, an applicant may not know precisely how much he or she would like the job in question because of a lack of information about the nature of the job, the company culture, and his or her future relations with colleagues. At the same time, it is generally quite difficult for recruiters to judge which candidates best fit the position.
- In the event of an infectious disease outbreak, due to an unknown virus, biochemists need to find diagnostic reagents that have no serious side effects. In a simplified formulation, this involves a search for a reagent that satisfies a collection of constraints (e.g., one constraint may be that the reagent should not contain certain medical ingredients composed in a certain way, as otherwise, its reaction with the virus could cause a severe headache). If the biochemists knew everything about the virus (e.g., its DNA sequence, chemical composition, etc.), then it would be much easier for them to find a diagnostic reagent. If the virus is largely unknown, however, then they are left with an effectively unknown-constraints formula to satisfy. Of course, they could try to employ modern DNA technologies to gain most information on the virus, but

doing so usually takes a long time, and identifying a diagnostic reagent to control the ensuing pandemic is a matter of urgency.

In summary, an input, while it exists, may be *unknown* because of our limited knowledge and control of the system, or of our lack of experience in a new environment. There are numerous other scenarios with unknown inputs, e.g., animal behavior studies, neural science, and hidden web databases, to name just a few.

1.1 Trial and Error

Trial and error is a basic methodology in problem solving and knowledge acquisition, and it has also been used extensively in product design and experiments [32]. Generally speaking, the approach proceeds by adaptively posing a sequence of candidate solutions and observing their validity. If a proposed candidate solution is found to be valid, then the mission is accomplished. Otherwise, an error is signaled from one of the characteristics of the studied object. An important feature of the approach is its solution orientation: the goal is to find one solution, with little care paid to other considerations such as why the solution works [1].

Trial and error is also a commonly used approach in the aforementioned examples. In economics, individuals adopt and adaptively adjust their strategies based on observed market reactions. Such self-motivated, but self-regulating, types of behavior, as implied by Adam Smith's "invisible hand" theory, can converge to a socially desirable state (even without the individuals involved having any knowledge of one another). In a company, an employee will usually look for a more suitable position when dissatisfied with his or her current one, and senior management will usually encourage personnel adjustments to enhance performance. Biomedical scientists conduct clinical trials to test their designed reagents, and if an unacceptable side effect is observed, then they collect and analyze feedback data to help with future diagnostic reagent tests [25].

The most critical ingredient in the trial and error approach is how to employ previously returned errors to propose future trials. This procedure is algorithmic in nature, but it does not seem to have been formally addressed from an algorithmic perspective. This paper aims to investigate this approach on a broad category of problems with unknown inputs.

1.2 Model and Preliminaries

Motivated by the foregoing examples, we investigate the effects of the lack of input information from a computational viewpoint on the basis of the trial and error approach. The central question we consider is the following.

> *How much extra difficulty is introduced due to the lack of input knowledge?*

In this paper, we explore this question in search problems. Suppose that on an input (instance) I, there is a set $S(I)$ of *solutions*. A search problem is to find a solution $s \in S(I)$ to input I.[1] Numerous problems arising from a variety of applications studied in algorithm design and computational complexity are search problems. Typical examples include searching for a Nash equilibrium in a multi-player game, searching for a satisfying assignment in a conjunctive normal form (CNF) formula, and finding a stable matching to pair individuals with preferences in a two-sided market.

All of these problems, in addition to the motivating examples discussed earlier, naturally fall into the broad category of *constraint satisfaction problems* (CSPs). Suppose that there is a space $\Omega = \{0,1\}^n$ of candidate solutions. Corresponding to an input I, there are a number of constraints $C_1, C_2, \ldots, C_m(,\ldots)$, where each $C_i \subseteq \{0,1\}^n$ is a relation on the solution variables defined on given domains.[2] The solutions of I are defined as those s that satisfy all constraints C_i, i.e., $s \in \bigcap_i C_i$. Note that the number of constraints can range from constant to polynomial, exponential, or even infinite. CSPs are a subject of intensive research in theoretical computer science, artificial intelligence, and operations research, and they provide a common basis for exploration of a large number of problems with both theoretical and practical importance.

This paper addresses the situation in which the input I is unknown. For a search problem A, we denote by A_u the same search problem with unknown inputs. For example, in the StableMatching problem, the input contains the preference lists of all men and women; in $\mathsf{StableMatching}_\mathsf{u}$, these preference lists are unknown to us. The constraints are that all man-woman pairs (m, w) are not blocking pairs, and the task is to find a solution that satisfies all constraints, namely a stable matching [20].

Similar to the way in which a biochemist proposes a chemical reagent and then performs clinical tests, here our method of searching for a solution of a CSP is also the trial and error approach. We propose a candidate solution s: If s is not a valid solution, then we are told so by a *verification oracle* V, and, what is more, V also gives us the *index* of one constraint that is not satisfied. Otherwise, s satisfies all constraints, and then we cannot observe any violated constraint; equivalently, V returns a confirmative answer, and our job is done. Some remarks are necessary:

- If more than one constraint is violated, then (the index of) any one of them can be returned by V. We make no assumption about which one, not only because worst-case analysis is standard in algorithm and complexity studies, but also because in many applications, such as drug tests, the verification oracle is carried out by Nature or human bodies, and thus how and which violation is returned is truly beyond our current understanding.
- Note that V does not reveal the constraint itself, but only its index or label. For example, we know something like "the third constraint is violated" in the proposed assignment of the SAT_u problem, or "the second player has a better mixed strategy" for the proposed strategy in the Nash_u

[1] One might wish to find more or even all solutions. Here, we follow the standard requirement for searching problems in complexity theory [21, 3] by asking for only one (arbitrary) solution.

[2] Note that it is possible for a problem to have different CSP definitions, which, depending on the available set of tools, may in turn lead to different complexities for solving the problem. For example, to identify an unknown substance, we can employ different (physical, chemical, etc.) test methods, which, in general, require different costs. This phenomenon reflects the intrinsic variety of a problem and the diversity of its solutions. Thus, to define an unknown-input problem, we need to indicate explicitly its corresponding CSPs. For the problems investigated in this paper, we arguably use their most natural definitions.

problem, but the exact content of the constraint (i.e., the literals in the clause of SAT_u or the player's utility function of Nash_u) is still unknown to us, which is consistent with our motivating examples. If a headache is observed in a drug development clinical trial, then we do not always know which components of the proposed reagent caused the problem: We have only a label of "headache" for the proposed reagent.

Surprisingly, despite this seemingly very little information and the worst-case assumption on the verification oracle, we still have efficient algorithms for many problems.

Given the verification oracle V, an algorithm is an interactive process with V. We choose candidate solutions (i.e., trials), and the oracle returns violations (i.e., errors). The process is adaptive, i.e., the newly proposed solution can be based on the historical information returned by the oracle.

Because our focus is on how much *extra* difficulty is introduced by the lack of input information for a search problem A, we single out this complexity by comparing the unknown-input and known-input scenarios. To this end, we equip our algorithms with another oracle, the *computation oracle*, which can solve the known-input version of the same problem A. Overall, our algorithms can access two oracles, the verification oracle and the computation oracle (we do not allow them to invoke each other).

As is standard in complexity theory, a query to either oracle has a unit time cost. The *time complexity* of a problem with unknown inputs is the minimum time needed for an algorithm to solve it for all inputs and all verification oracles consistent with the input. We employ the standard notation in computational complexity theory for complexity classes such as $\mathbf{P}$ and $\mathbf{NP}$ and also for oracles. For example, $\mathsf{A}_u \in \mathbf{P}^{\mathsf{V},\mathsf{A}}$ means that problem A_u can be solved by a polynomial-time algorithm with verification oracle V and the computation oracle that can solve the known-input version of A. If this occurs, then we consider the extra complexity (resulting from the unknown input) not to be very high. The central question can therefore be translated to the following. Given a search problem A, is $\mathsf{A}_u \in \mathbf{P}^{\mathsf{V},\mathsf{A}}$? If the given known-input problem A is in $\mathbf{P}$, then the computation oracle can be omitted, and the problem becomes "Is $\mathsf{A}_u \in \mathbf{P}^{\mathsf{V}}$?"

We also define the *trial complexity* of an unknown-input problem A_u as the minimum number of queries to the verification oracle that any algorithm needs to make, regardless of its computational power.[3] As is standard in query complexity theory, we can consider deterministic or (Las Vegas) randomized algorithms. The latter can be assumed to be error-free because of the verification oracle V, and we count the cost as the expected number of queries to V. We denote by $\mathsf{D}(A_u)$ and $\mathsf{R}(A_u)$ the deterministic and randomized trial complexities of A_u, respectively.

We investigate trial complexity not only because it provides a rigorous proof of computational hardness, but also because it measures the number of trials (or in another perspective, errors) that must be undertaken to find a solution. Note that in many scenarios, such as diagnostic reagent development, trials constitute the major expense, both financially and temporally, and in almost all of the motivating examples discussed earlier, an important goal is to design protocols or experiments with a small number of trials.

[3]It is thus the "query complexity" to the verification oracle. Here, we adopt the term "trial complexity" to avoid any potential confusion of the two types of oracle queries (corresponding to the two oracles).

1.3 Our Results and Techniques

We consider a number of problems, that are motivated by the aforementioned examples, to investigate the trial and time complexities resulting from the lack of input knowledge. (The formal definitions of these problems and their natural formulation as CSPs are deferred to subsequent sections.)

THEOREM 1. *For the following problems* A*, we have* $\mathsf{A}_u \in \mathbf{P}^{\mathsf{V},\mathsf{A}}$.

- Nash*: Find a Nash equilibrium of a normal-form game.*
- Core*: Find a core of a cooperative game.*
- $\mathsf{StableMatching}$*: Find a stable matching of a two-sided market with preferences.*
- SAT*: Find a satisfying assignment of a CNF formula.*

Nash is a fundamental problem in game theory, and its complexity has been characterized (as **PPAD**-complete) [16, 14]. Core is also a fundamental problem in cooperative game theory [35]. Both problems are naturally defined as CSPs. Our algorithms for both Nash_u and Core_u employ the ellipsoid method, although for Nash_u we shrink the input space, and for Core_u we shrink the solution space. One technical difficulty is that the target space may degenerate to the case of containing at most one point. (In Nash_u, there is only one input point, and in Core_u the core may contain only one point or even be empty.) Note that the standard perturbation approach, which proceeds by increasing the volume of the feasible region, is not applicable in our setting, because the linear constraints, as the input, are unknown. Here, we employ a more sophisticated ellipsoid method that works as long as the polyhedron can be specified by a *strong separation oracle*. As it turns out, this oracle can be constructed from the verification oracle V in both problems, and, crucially, the construction for Nash_u uses the existence of a Nash equilibrium in *any* game.

$\mathsf{StableMatching}$ is a problem with interesting combinatorial structures and many applications, such as the pairing of graduating medical students with hospital residencies [38, 37]. SAT is a natural CSP, with the constraints being the OR of some literals. Considering the practical significance of $\mathsf{StableMatching}$ and SAT, we take a closer look at their trial complexities.

THEOREM 2. *We have the following bounds for the trial complexity.*

- $\Omega(n^2) \le \mathsf{R}(\mathsf{StableMatching}_u) \le \mathsf{D}(\mathsf{StableMatching}_u) \le O(n^2 \log n)$*, where* n *is the number of agents.*
- *Given a formula with* n *variables and* m *clauses,* $\mathsf{R}(\mathsf{SAT}_u) \le \mathsf{D}(\mathsf{SAT}_u) = O(mn)$*. Further,* $\mathsf{R}(\mathsf{SAT}_u) = \Omega(mn)$ *if* $m = \Omega(n^2)$*, and* $\mathsf{R}(\mathsf{SAT}_u) = \Omega(m^{3/2})$ *if* $m = o(n^2)$.

The proofs of both lower bounds deviate from the standard method of applying Yao's min-max principle. Rather, they are obtained by arguing that, for an arbitrary but fixed *randomized* algorithm with an insufficient number of queries, there are input instances with disjoint solution sets between which the algorithm cannot distinguish. The existence of such input instances is proved by the probabilistic method for $\mathsf{StableMatching}_u$, and by an adaptive construction procedure for SAT_u.

The upper and lower bounds proofs for $\mathsf{R}(\mathsf{StableMatching}_\mathsf{u})$ also employ order theory [10, 17]. A key step is to characterize how fast one can shrink the set of linear orders consistent with a partial order by worst-case pair violations. We identify the average height as the correct measure; the control of which allows us to bound the speed of the shrinkage. Along the way, we examine another natural problem, $\mathsf{Sorting}_\mathsf{u}$, whose trial complexity is completely pinned down as $\Theta(n \log n)$.

It is somewhat surprising that knowing only the indices of violated constraints is already sufficient to admit quite a number of efficient algorithms. It is therefore natural to wonder whether the lack of input information adds any extra difficulty at all in finding a solution. We find that it does indeed: there are problems whose unknown-input versions are considerably more difficult than their known versions. Two representatives are GraphIso and GroupIso, the problems of deciding whether two given graphs or groups are isomorphic.

THEOREM 3. *We have the following hardness results.*

- *If* $\mathsf{GraphIso}_\mathsf{u} \in \mathbf{P}^{\mathsf{V},\mathsf{GraphIso}}$*, then the polynomial hierarchy (***PH***) collapses to the second level.*
- *If* $\mathsf{GroupIso}(\cdot, \mathbb{Z}_p)_\mathsf{u} \in \mathbf{P}^{\mathsf{V}}$*, then we have* $\mathbf{P} = \mathbf{NP}$*. Here,* $\mathsf{GroupIso}(\cdot, \mathbb{Z}_p)$ *is the group isomorphism problem with the second group known as* $\mathbb{Z}_p$ *for a prime* p*.*

However, if SAT *is given as the computation oracle, then we have deterministic polynomial-time algorithms for* GraphIso *and* GroupIso*, i.e.,* $\mathsf{GraphIso}_\mathsf{u} \in \mathbf{P}^{\mathsf{V},\mathsf{SAT}}$ *and* $\mathsf{GroupIso}_\mathsf{u} \in \mathbf{P}^{\mathsf{V},\mathsf{SAT}}$*, with* $O(n^6)$ *and* $O(n^2)$ *trials, respectively.*

Note that $\mathsf{GroupIso}(\cdot, \mathbb{Z}_p)$ (with a known input) admits a simple polynomial-time algorithm by comparing the multiplication tables. Actually, GroupIso is in $\mathbf{P}$ if the two groups are Abelian [28]. However, if the multiplication table of the input group is unknown, then, surprisingly, the problem becomes **NP**-hard. Interestingly, this substantial increase in computational difficulty occurs only for time complexity, not for trial complexity, which can be seen as a tradeoff (from below) between the two complexity measures—a phenomenon not commonly seen in other query models.

This hardness result for $\mathsf{GroupIso}(\cdot, \mathbb{Z}_p)_\mathsf{u}$ is proved by a nonstandard reduction from the classic **NP**-complete problem of finding a Hamiltonian cycle. We use an algorithm $\mathcal{A}$ for $\mathsf{GroupIso}(\cdot, \mathbb{Z}_p)_\mathsf{u}$ to find a Hamiltonian cycle in a given graph G in the following way. Assuming the existence of a Hamiltonian cycle C, which does not change the **NP**-completeness of the problem, we define a group H via C and run $\mathcal{A}$ on input $(H, \mathbb{Z}_p)$. An issue here is that because the reduction algorithm has only polynomial time, it cannot find such a Hamiltonian cycle for defining H. A related issue is how to provide the verification oracle V for $\mathcal{A}$ without knowing C. These issues can be overcome by (i) making use of the crucial property that $\mathcal{A}$ does not know its input, and (ii) designing an efficient simulator V' for the verification oracle V. Due to the time constraint, V' cannot perfectly mimic V to answer all of $\mathcal{A}$'s questions correctly. However, it is designed with the favorable property that whenever it produces an incorrect answer, a Hamiltonian cycle in G has just been found. ($\mathcal{A}$'s correctness on H may already have been compromised, but we are not further concerned with it. We simply use $\mathcal{A}$'s code to serve the purpose of finding a Hamiltonian cycle in G.)

Finally, beyond all of the foregoing problems that can be solved in $\mathbf{P}^{\mathsf{V},\mathsf{SAT}}$, we show via an information theoretical argument that certain other problems, such as Subset Sum, have an exponential lower bound for the randomized trial complexity.

THEOREM 4. $\mathsf{R}(\mathsf{SubsetSum}_\mathsf{u}) = \Omega(2^n)$.

In a followup work [9], we showed that solving an unknown linear programming with m constraints and n variable requires $\Omega(m^{n/2})$ queries to the oracle, i.e., it also has an exponential lower bound on the trial complexity. This result implies that our efficient algorithm solving Nash_u does not completely resort to the computation oracle Nash: it is indeed the combinatorial structure of Nash equilibrium (i.e., its existence) that yields our algorithm. See more discussions in [9].

Our results illustrate the variety of time and trial complexities that arise from the lack of input information for different problems, and imply distinct levels of the criticality of input information for different problems.

In addition to the specific techniques previously mentioned for each problem, a general remark is that, at a very high level, our algorithms are in line with the candidate elimination approach, similar to many existing learning algorithms [30]. However, our framework allows a space for possible inputs *and* a space for possible solutions—the interplay between them seems to be the main source of combinatorial structures, and how well the two spaces are combinatorially related accounts for the complexity of the problem. Some algorithms (e.g., that for Core_u) obtain their efficiency by directly shrinking the solution space. Even for those that shrink the input space (e.g., those for SAT_u and Nash_u), the key is to explore the relation of the two spaces and to design trials such that even a worst-case violation can be used to cut out a decent fraction of the input space.

1.4 Relation to Existing Work

Our model with unknown inputs bears a resemblance to certain other problems and models, e.g., learning, algorithm design in unknown environments, ellipsoid method, and query complexity. However, there are fundamental distinctions between these models and ours. In this section, we discuss the relation of our work to these models and problems (more detailed discussions are referred to the full version of the paper [8]).

Learning. Our model has strong connections to various learning theories (e.g., concept learning with membership or equivalence query [2], decision tree learning, reinforcement learning [6], and (semi-)supervised learning [5]), but fundamental differences also exist. The high-level philosophy of these models is "sample and predict", which is very different from our trial and search (for a solution).

1. Learning theories, in essence, aim to identify the unknown object *itself*, either exactly (as in concept learning) or approximately (sometimes in the form of a prediction, as in PAC learning and active learning). In our model, however, the ultimate goal is quite different: we attempt only to find a solution of an unknown object, without necessarily learning the object itself. For certain applications (such as the aforementioned development of diagnostic reagents), finding a solution is indeed the main mission.

2. It is important to note that in our model, a solution may be found long before the exact input is learnt. Further, in certain cases, such as $\mathsf{SAT_u}$ and $\mathsf{Nash_u}$, the exact input may take an exponential number of queries or even be impossible to learn. For $\mathsf{SAT_u}$, even if we relax the requirement by allowing to output any formula within a cluster in which all formulas have the same set of satisfying assignments as the hidden input formula, it is still exponentially harder than finding a solution (Proposition 9). Our algorithm, in contrast, is able to find a solution in polynomial time without learning the exact input formula.
3. Both similarities and differences abound in existing learning theories, and deciding which one to use in a specific application largely depends on the available method of accessing the unknown. As we have demonstrated, there are a fairly large number of scenarios in which the only available access to the unknown is provided by the verification oracle, but existing learning theories do not seem to address such situations.

In summary, with its solution-oriented objective and advantages in computational efficiency, the present work is hopefully to serve as a useful supplement to existing learning theories, particularly in contexts in which the unknown object itself is impossible or unaffordable to learn and the only available access to the unknown is through a solution-verification process.

Complexity. In the query model (also known as decision tree model), an algorithm makes queries in the form of "$x_i = ?$", and the task is to compute a function f on the unknown x by the minimum number of queries [13]. Although this area has the same flavor of computing a function without learning all input variables, it is quite different from our model in the form of queries allowed.[4] Therefore, our results on trial complexity can be viewed as an extension of the traditional query model by allowing a much larger class of queries with natural motivations.

In some cryptographic tasks, input instances are hidden from one party. For example, in instance-hiding proof systems [7], the verifier tries to compute a function f on input x by interacting with one or more provers, without leaking any information on x to the prover(s). Although this model is clearly very different from ours, an interesting research direction would be to explore the connections between our model and various cryptographic tasks.

2. STABLE MATCHING

In a Gale-Shapley two-sided matching market model [20], we are given a set of men M and a set of women W, where $|M| = |W| = n$. Each man $m \in M$ has a strict and complete *preference list*,[5] denoted by $\succ_m$, ranking all the women in W, where $w_1 \succ_m w_2$ means that m prefers w_1 to w_2. The preference list $\succ_m$ is assumed to be transitive (i.e., if $w_1 \succ_m w_2$ and $w_2 \succ_m w_3$, then $w_1 \succ_m w_3$). The preference list $\succ_w$ of every woman $w \in W$ is defined similarly.

Given a matching between M and W, denoted by μ, we say that $m \in M$ and $w \in W$ form a *blocking pair* if both prefer each other to their matched partner in μ, i.e., $w \succ_m \mu(m)$ and $m \succ_w \mu(w)$, where $\mu(m)$ and $\mu(w)$ are the woman matched to m and the man matched to w in μ, respectively. Matching μ is called *stable* if it contains no blocking pair. The StableMatching problem is to find a matching that is stable, namely, a matching that satisfies the following set of constraints, labeled by man-woman pairs.

$$\text{Either } \mu(m) \succ_m w \text{ or } \mu(w) \succ_w m, \forall m \in M, w \in W. \quad (1)$$

A stable matching always exists, and can be computed by Gale-Shapley's deferred acceptance algorithm in time $O(n^2)$.

In the unknown-input version of the stable matching problem, denoted by $\mathsf{StableMatching_u}$, we do not know the preference lists $\succ_m$ and $\succ_w$. What we can do is to propose candidate matchings as potential solutions. If a proposed matching is indeed stable, then the verification oracle V returns Yes, and the problem is solved. If it is not stable, then one constraint (i.e., a blocking pair) is revealed by V. Our first result is an $O(n^2 \log n)$ upper bound on the randomized trial complexity of $\mathsf{StableMatching_u}$.

THEOREM 5. *There is a polynomial-time randomized algorithm solving* $\mathsf{StableMatching_u}$ *with* $O(n^2 \log n)$ *trials.*

Before describing the idea underlying the proof of this result, we first look at the unknown-input version of another basic problem, Sorting, which has a close relationship with StableMatching. In $\mathsf{Sorting_u}$, there is a set of n elements $S = \{a_1, a_2, \ldots, a_n\}$ in some underlying linear (total) order $\succ$, but this order is unknown to us, and the task is to discover it. We can propose a linear order $(a_{k_1}, a_{k_2}, \ldots, a_{k_n})$ each time. If it is indeed the desired hidden total order, i.e., $a_{k_1} \succ a_{k_2} \succ \cdots \succ a_{k_n}$, then the verification oracle returns Yes, and the problem is solved; otherwise, a pair of elements (a, b) is returned such that a is before b in the proposed order, whereas $b \succ a$ in the actual order.

It is well known that the time complexity of a comparison-based sorting problem is $\Theta(n \log n)$. Note that this time complexity for Sorting is completely different from our trial complexity for $\mathsf{Sorting_u}$. In the following, we will show that the trail complexity bound for $\mathsf{Sorting_u}$ is actually also $\Theta(n \log n)$.

LEMMA 6. $\mathsf{Sorting_u}$ *can be deterministically solved using* $O(n \log n)$ *trials, and the running time can be made polynomial for randomized algorithms. Meanwhile, any randomized algorithm that solves* $\mathsf{Sorting_u}$ *needs at least* $\Omega(n \log n)$ *trials, even with unbounded computational power.*

PROOF. First, we define the notation as follows. A *partially ordered set* (or *poset*) is a set S equipped with an irreflexive transitive relation $>$. A *linear extension* of a poset $(S, >)$ is a linear order $\succ$ over set S such that $a \succ b$ whenever $a > b$ in S. For any given poset $(S, >)$ and elements $a, b \in S$, we denote by $\Pr(a \succ b)$ the probability of $a \succ b$ where $\succ$ is chosen uniformly at random among linear extensions of $(S, >)$.

In the $\mathsf{Sorting_u}$ problem, suppose the unknown order is $\succ^*$. Notice that for each of our proposed orders $\succ$, the verification oracle returns a pair of elements $a, b \in S$, from which and $\succ$, we can infer the relation between a and b in the

[4] Other forms of queries were also considered, such as those in linear decision trees and algebraic decision trees, but they are still in a very restricted form of queries.

[5] We follow the model proposed by Gale and Shapley in their seminal work [20], where the number of men and women is the same, and every individual's preference is assumed to be complete and strict. All of these assumptions can be removed in our results, but for simplicity of exposition, we will adopt Gale and Shapley's original model.

actual order $\succ^*$. Thus, at each point of the algorithm, the information collected so far forms a poset $(S, >)$, of which the underlying unknown order $\succ^*$ is a linear extension.

For any poset $(S, >)$, we call an order $(a_{k_1}, a_{k_2}, \ldots, a_{k_n})$ *good* if there is a constant $c < 1$, such that for any $i < j$, we always have $\Pr(a_{k_j} \succ a_{k_i}) < c$. Note that this is a very strong condition because it requires a constant shrinkage for *all* possible pairs (i, j). The idea of solving $\mathsf{Sorting}_\mathsf{u}$ is that, at each step of the algorithm when our collected information forms a poset $(S, >)$, we propose a good order if it exists. The property of a good order guarantees that whatever the verification oracle returns, we can always reduce the number of candidate linear extensions by a constant fraction c. Note that at the beginning of the algorithm, the number of candidate linear extensions is $n!$ as we do not have any information about $\succ^*$. And at the end of the algorithm the unique order $\succ^*$ is found and thus the number of linear extensions is 1. Therefore, the problem $\mathsf{Sorting}_\mathsf{u}$ can be solved in polynomial time and by $O\big(\log_{1/c} n!\big) = O(n \log n)$ trials to the verification oracle, provided that

1. for any poset $(S, >)$, a good order always exists, and
2. we find a good order in polynomial time.

Next we shall address how to satisfy these two conditions. Given a poset $(S, >)$ and an element $a \in S$, define its *average height* $h(a)$ to be the average rank of a in all linear extensions of $(S, >)$, where the rank of a in a linear extension $\succ$ is the number of elements b such that $b \succ a$. It is easy to see that the average height of elements in S are all rational numbers between 0 and $n - 1$. Kahn and Saks [27] showed that for any pair of elements $a, b \in S$ satisfying $h(a) - h(b) < 1$, we must have $\Pr(b \succ a) < \frac{8}{11}$. Thus, for any poset $(S, >)$, if we sort all elements of S in order $(a_{k_1}, a_{k_2}, \ldots, a_{k_n})$ such that $h(a_{k_1}) \le h(a_{k_2}) \le \cdots \le h(a_{k_n})$, then for any $i < j$ we will have $h(a_{k_i}) - h(a_{k_j}) \le 0 < 1$, and thus $\Pr(a_{k_j} \succ a_{k_i}) < \frac{8}{11}$. This implies that this is a good order as we want.

Therefore, it remains to compute $h(a)$ efficiently for every element a. However, it was shown in [11] that counting the number of linear extensions of a given poset is $\#\mathbf{P}$-complete, and determining the average height of an element of a poset is polynomially equivalent to the linear extension counting problem, thus is also $\#\mathbf{P}$-complete. Luckily, to our purpose of having a good order, an approximation of $h(a)$ to a small enough precision suffices. To this end, we first notice that there exists a fully polynomial randomized approximation scheme (FPRAS) for the problem of counting the number of linear extensions [18, 12], where the algorithm finds, with probability $1-\delta$, a $(1+\epsilon)$-approximation of the number of the linear extensions in time $poly(n, 1/\epsilon, \log(1/\delta))$. Now given a poset $(S, >)$ and two elements $a, b \in S$, we can apply this algorithm to posets $(S, >)$, getting an output n_1, and apply the algorithm on $(S, >')$, getting an output n_2, where $>'$ is obtained from $>$ by incorporating an extra relation $(a > b)$. Then $\Pr(a \succ b)$ can be approximated by n_2/n_1 (with the precision $\frac{1+\epsilon}{1-\epsilon}$). It is easily seen from the definition of $h(a)$ and that of $\Pr(b \succ a)$ that $h(a) = \sum_{b \neq a} \Pr(b \succ a)$. By setting $\epsilon = \frac{1}{5n}$ to approximate the value of each $\Pr(b \succ a)$ and using them to compute the value of $h(a)$, we can derive in polynomial time a value $h'(a)$ such that $1 - \frac{1}{2n} < \frac{h'(a)}{h(a)} < 1 + \frac{1}{2n}$ with high probability. Since $h(a) < n$, we have

$$|h'(a) - h(a)| < \frac{h(a)}{2n} < 0.5.$$

Using $h'(a)$ to sort all elements in S in order $(a_{k_1}, a_{k_2}, \ldots, a_{k_n})$, we have by the above inequality that $h'(a_{k_i}) < h'(a_{k_j})$ for any $i < j$ with arbitrarily high probability. This implies $h(a_{k_i}) - h(a_{k_j}) < 1$, and thus, $\Pr(a_{k_j} \succ a_{k_i}) < \frac{8}{11}$. Therefore, this is a good order for the current poset $(S, >)$. The whole process can be done in polynomial time, which completes the proof of the upper bound side.

For the lower bound side, it was also shown in [27] that for any poset $(S, >)$, there exist two elements $a, b \in S$ such that $\Pr(a \succ b) > \frac{3}{11}$ and $\Pr(b \succ a) > \frac{3}{11}$. Thus, we construct the verification oracle as follows. At any step, when the previously returned information forms a poset $(S, >)$, no matter what the current proposed order is, the oracle always returns such (a, b) (or (b, a), depending on their relative position in the proposed order) as a violation. Then after this trial, at least $\frac{3}{11}$ fraction of the possible linear extensions still remains. Therefore, we have $\mathsf{R}(\mathsf{Sorting}_\mathsf{u}) \ge \log_{11/3} n! = \Omega(n \log n)$. □

Having the upper bound result for $\mathsf{Sorting}_\mathsf{u}$, we can consider the preference of each individual as a sorting problem and solve these $2n$ $\mathsf{Sorting}_\mathsf{u}$ problems together, which gives us the desired upper bound for the $\mathsf{StableMatching}_\mathsf{u}$ problem.

Note that the same results for deterministic algorithms hold; that is, there is an exponential-time deterministic algorithm using $O(n \log n)$ trials for $\mathsf{Sorting}_\mathsf{u}$ (and thus another algorithm using $O(n^2 \log n)$ trials for $\mathsf{StableMatching}_\mathsf{u}$), namely, $\mathsf{D}(\mathsf{Sorting}_\mathsf{u}) = O(n \log n)$ and $\mathsf{D}(\mathsf{StableMatching}_\mathsf{u}) = O(n^2 \log n)$. The reason is simple: we can compute the average height $h(a)$ for every a by enumerating all linear extensions of the current partial order. See Section 6 for more discussions on polynomial time deterministic algorithms.

Note that our algorithm for $\mathsf{StableMatching}_\mathsf{u}$ essentially runs $2n$ $\mathsf{Sorting}_\mathsf{u}$ instances to learn the entire unknown input of the preference lists, and analysis of the trial cost simply involves adding the trials made on these instances. Although the $\Omega(n \log n)$ lower bound for $\mathsf{Sorting}_\mathsf{u}$ implies a limit to this approach, there seems to be considerable room for improvement. It may be unnecessary to learn all of the input lists to find a solution, and there may be more sophisticated ways to solve $2n$ instances of $\mathsf{Sorting}_\mathsf{u}$ to beat the naive upper bound by addition. However, the following theorem gives an almost matching lower bound for $\mathsf{StableMatching}_\mathsf{u}$.

THEOREM 7. *Any randomized algorithm for* $\mathsf{StableMatching}_\mathsf{u}$ *needs at least* $\Omega(n^2)$ *trials even with unbounded computational power.*

A final comment is that in the traditional stable matching problem, where the preferences are known, there is a tight lower bound $\Omega(n^2)$ for computing a stable matching [34]. However, this bound refers to the computational time complexity in a completely different meaning from our trial complexity lower bound.

3. SAT

Given a CNF formula ϕ with n variables and m clauses, the SAT search problem is to find a satisfying assignment to ϕ if one exists, or to return "ϕ is unsatisfiable." In the unknown-input version SAT_u, the formula ϕ is unknown, and each time that a proposed solution x is not a satisfying assignment, the verification oracle returns an index i such that the i-th clause of ϕ evaluates to FALSE on x.

First, we clarify that the computation oracle for SAT is defined as follows. On a query ϕ which is a CNF formula, SAT returns a satisfying assignment x to ϕ, or reports that such x does not exist. So this SAT oracle solves the search problem instead of the decision problem. This is for the fair comparison since the target $\mathsf{SAT_u}$ is also a search problem. Also note that the search and the decision problems for SAT are roughly the same due to the standard self-reducibility.

Our algorithm solving the $\mathsf{SAT_u}$ problem is as follows.

Algorithm 1 ALG-SAT

Unknown input: A formula ϕ of n variables and m clauses.

Let $L_1 = L_2 = \cdots = L_m = \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$.
loop
  Let $\phi' = \bigwedge_{i=1}^{m}(\vee_{\ell \in L_i} \ell)$.
  Query the computation oracle SAT on ϕ'.
  if the computation oracle says that ϕ' is not satisfiable, **then**
    return ϕ is unsatisfiable (and terminate the program).
  else
    Suppose the computation oracle gives a satisfying assignment x of ϕ'.
    Ask the assignment x to the verification oracle V.
    if V confirms that $\phi(x) = 1$ **then**
      return x (and terminate the program).
    else
      Suppose V returns an index i.
      Let $L_i \leftarrow \{\ell \in L_i : \ell(x) = 0\}$
      (i.e., those literals in L_i evaluating FALSE on x).
    end if
  end if
end loop

The idea of the algorithm uses a standard candidate elimination approach [31], which has been used extensively in learning theory (e.g., PAC learning for CNF formulas [41]). The algorithm initially includes all literals, i.e., $x_1 \vee \bar{x}_1 \vee \cdots \vee x_n \vee \bar{x}_n$, in each clause. It proceeds by employing the SAT computation oracle to propose an assignment x consistent with the current knowledge of the clauses. If a clause's index is returned by the verification oracle upon a trial, then we know that x_i cannot be in the clause if $x_i = 1$ and that $\bar{x}_i$ cannot be in the clause if $x_i = 0$ in the assignment of the trial. We therefore remove the literals from the clause, and continue the process until either a satisfying assignment is found or the computation oracle returns the result that no satisfying assignment exists.

THEOREM 8. ALG-SAT *solves the* $\mathsf{SAT_u}$ *problem in polynomial time using* $O(mn)$ *trials, where* n *and* m *are the numbers of variables and clauses, respectively.*

Our upper bound result implies that not knowing the input formula does not add much extra complexity (up to a polynomial) to finding a satisfying assignment. For a related problem, 2SAT, in which every clause contains at most 2 literals, which can be solved in polynomial time, we can show that solving $\mathsf{2SAT_u}$ in polynomial time implies $\mathbf{P} = \mathbf{NP}$.[6] Therefore, it is indeed the power of the computation oracle SAT that yields our efficient algorithm for $\mathsf{SAT_u}$. Recall that the focus of our study is not to discover the complexity of solving $\mathsf{SAT_u}$ itself, but rather, is to understand the relative complexity of solving $\mathsf{SAT_u}$ compared with that of solving SAT.

Note that our algorithm directly shrinks the space of possible inputs (i.e., formulas), instead of the space of possible solutions (i.e., assignments). (While some variables may have their values fixed along the process of the algorithm, this may not necessarily be the case and is surely not the reason why our algorithm works within $O(mn)$ queries.) When a satisfying assignment is found by the algorithm, however, we may still do not know the exact input formula. This echoes the medical treatment application mentioned in Introduction, where finding a solution is much more urgent than learning the unknown input, and it is indeed often possible to find an effective diagnostic reagent long before completely knowing the virus.

In the complexity language, we ask whether it is computationally much harder to find out the hidden CNF formula (even with the help of the computation oracle SAT) than finding a solution. Learning the exact formula is clearly impossible since, for example, if there is a clause $x_i \vee \bar{x}_i$, then we can never know this clause. Even if we relax the requirement by clustering the formulas with the same set of satisfying assignments, and allowing to output any formula within the cluster of the hidden input formula, it is still exponentially harder than finding a solution. This separates ours from the standard concept learning model.

PROPOSITION 9. *Any randomized algorithm with* ϵ*-error,* $\epsilon < 1/2$*, needs* 2^n *trials to output a formula* ϕ' *with the same set of satisfying assignments as the hidden input formula* ϕ*.*

We also have a lower bound on the randomized trial complexity of $\mathsf{SAT_u}$, which matches the upper bound at least when $m = \Omega(n^2)$.

THEOREM 10. *If* $m = \Omega(n^2)$*, any randomized algorithm for the* $\mathsf{SAT_u}$ *problem needs at least* $\Omega(mn)$ *trials, even with unbounded computational power. If* $m = o(n^2)$*, the lower bound is* $\Omega(m^{3/2})$*.*

The proof of the lower bound is combinatorially involved, in which we construct instances such that only one candidate literal can be eliminated in the foregoing process. This is enforced with the help of certain clauses that confine all satisfying assignments to a very special form. Further, we remark that the proof can be easily adapted to show the same lower bound for the decision problem, namely to decide whether a formula (with unknown clauses) is satisfiable.

4. GROUP ISOMORPHISM

Given two groups, G and G', of the same size, the group isomorphism (GroupIso) problem is to find an isomorphism between G and G' or to report that "$G \ncong G'$". More precisely, given two groups, G and G', by their multiplication tables, $T_{n\times n}$ and $T'_{n\times n}$, our task is to output a bijection $\pi : G \to G'$ such that

$$\pi(a \circ b) = \pi(a) \circ' \pi(b) \tag{2}$$

for all $a, b \in G$ (where $\circ$ and $\circ'$ are the multiplications of G and G', respectively), or to report that "$G \ncong G'$".

Whether a polynomial time algorithm exists for the general GroupIso problem is a long-standing open question. Compared with another well-known problem, Graph Isomorphism,

[6]The idea of the reduction is similar to the one described in the next section for group isomorphism.

however, GroupIso has more group structures for potential use, and indeed, GroupIso can be solved in polynomial time if the given groups are Abelian, and (the decision version of) GroupIso is in $\mathbf{NP} \cap \mathbf{co\text{-}NP}$ (under certain complexity assumptions) if the given groups are solvable [4].

The problem is by nature a constraint satisfaction problem, searching for a bijection π that satisfies the n^2 constraints Eq.(2). In the unknown-input model, we can consider the case of both groups being unknown and that of exactly one group, say, G, being unknown. In either case, we can propose bijections π to the verification oracle V. If π is not an isomorphism, then V gives a pair (a, b) with the foregoing equality violated. Our upper bound result in this section applies to the general case in which both groups are unknown, and our lower bound result applies even to the restricted case in which one group is known.

A very natural attempt to solve $\mathsf{GroupIso}_{\mathsf{u}}$ is to keep proposing bijections π that are consistent with our current knowledge of the restrictions of a valid bijection. More precisely, whenever V returns a pair of elements (a, b) on a proposed π, it adds the restriction that we cannot *simultaneously* map the three elements $(a, b, a \circ b)$ to $(\pi(a), \pi(b), \pi(a) \circ' \pi(b))$. Thus, there are no more than $O(n^6)$ forbidden rules. Note that finding a bijection that avoids a list of forbidden pairs of triples is easy with an **NP** computation oracle. Thus, $\mathsf{GroupIso}_{\mathsf{u}}$ can be solved using SAT as the computation oracle in polynomial time and via $O(n^6)$ trials.

An immediate question that arises from this claim is the following. Does it hold with only a computation oracle GroupIso instead of SAT? (After all, only comparison with GroupIso reveals the extra difficulty arising from the unknown input on the problem.) This seems quite conceivable that this is the case, as group theory by definition handles triples in the form $(\pi(a), \pi(b), \pi(a) \circ' \pi(b))$, and we have not yet exploited the group structures in the given tables. Surprisingly, this intuition turns out to be misleading, as shown by the following hardness result, which stands even if one group G' is known to us and is a very simple group $\mathbb{Z}_p$ for a prime p. Denote the known-input version of this problem by $\mathsf{GroupIso}(\cdot, \mathbb{Z}_p)$. Note that because it is solvable in polynomial time, the computation oracle is not needed (modulo a polynomial factor in runtime) in the unknown-input model.

THEOREM 11. *If* $\mathsf{GroupIso}(\cdot, \mathbb{Z}_p)_{\mathsf{u}}$ *can be solved in polynomial time, i.e.,* $\mathsf{GroupIso}(\cdot, \mathbb{Z}_p)_{\mathsf{u}} \in \mathbf{P}^{\mathsf{V}}$, *then* $\mathbf{P} = \mathbf{NP}$. *More specifically, if* $\mathsf{GroupIso}(\cdot, \mathbb{Z}_p)_{\mathsf{u}}$ *can be solved in time* $t(p)$, *then* HamiltonianCycle *can be solved in time* $O(t(p) \cdot p)$, *where* p *is the order of the given group* $\mathbb{Z}_p$.

Idea of the proof. The hardness result is shown by a reduction that is not very standard. Given a graph H with p vertices, we employ an algorithm $\mathcal{A}$ for $\mathsf{GroupIso}(\cdot, \mathbb{Z}_p)_{\mathsf{u}}$ to find a Hamiltonian cycle in H in the following way. Assuming the existence of a Hamiltonian cycle C (it can be seen that the primality of the size of the graph and the assumption of the existence of one Hamiltonian cycle do not alter the hardness of the Hamiltonian cycle problem), define a group T via C as follows. Let $(b_0, b_1, \ldots b_{p-1}, b_0)$ be a Hamiltonian cycle C in graph H. We can fix a vertex a and assume that $b_1 = a$, which is always achievable by a cyclic shift in the labels in the Hamiltonian cycle if necessary. For simplicity, we use the vertices of H to denote the elements of group T. Now, for any b_i and b_j in group T, define their multiplication by $b_i \circ b_j = b_{i+j \bmod p}$. It is easy to see that T is a cyclic group with p elements (where b_1 is a generator of the group).

The main idea of the reduction is to run algorithm $\mathcal{A}$ on input $(T, \mathbb{Z}_p)$, and translate the output of $\mathcal{A}$, which is an isomorphism from T to $\mathbb{Z}_p$, to a Hamiltonian cycle in the given graph H. However, one problem immediately arises: Because the reduction algorithm has only polynomial time, and it cannot find such a Hamiltonian cycle, thus cannot construct the multiplication table T.

To get around this issue, we employ the crucial fact that $\mathcal{A}$ does not know its first input—all of $\mathcal{A}$'s information about T comes from interactions with its verification oracle V. Thus, it is sufficient to construct a V that answers $\mathcal{A}$'s trials. However, doing so again requires the information on T, which is exactly what we do not have. Here, the idea is to efficiently construct a *simulator* V′ to take the place of V. Given the shortage of running time, it is inevitable that we lose something in our simulator V′, and, in our final construction it turns out to be the correctness, which is the seemingly the most critical component. In other words, V′ cannot answer all of $\mathcal{A}$'s questions correctly. What makes it still qualified for our purpose is the following key property. On any π proposed by $\mathcal{A}$, V′

1. either provides a correct response to π, or
2. finds a Hamiltonian cycle in H.

This property means that the first time V′ gives a wrong answer to $\mathcal{A}$, it has just found a Hamiltonian cycle in H. ($\mathcal{A}$'s output is admittedly now out of control now, but we no longer care about the correctness of A; we have used part of $\mathcal{A}$'s code to solve our HamiltonianCycle problem.) □

Our proof using a simulator is a reminiscent of zero-knowledge proofs; more discussions are referred to [8].

5. NASH EQUILIBRIUM

In a normal-form game, there are n players. Each player i has a strategy space S_i and a payoff function $u_i : S_1 \times \cdots \times S_n \mapsto \mathbb{Q}$, which gives the utility that i obtains for every strategy profile $(s_1, \ldots, s_n) \in S_1 \times \cdots \times S_n$. A joint probability distribution $(p_1, \ldots, p_n)$ on $S_1 \times \cdots \times S_n$ is called a *(mixed) Nash equilibrium* if for any player i and any probability distribution p_i' on S_i, we have

$$\sum_{(s_1,\ldots,s_n)} \prod_j p_j(s_j) \cdot u_i(s_1, \ldots, s_n) \geq \sum_{(s_1,\ldots,s_n)} p_i'(s_i) \cdot \prod_{j \neq i} p_j(s_j) \cdot u_i(s_1, \ldots, s_n). \quad (3)$$

Note that the number of constraints given by the foregoing inequality is unbounded. It is well-known that a two-player game admits a mixed Nash equilibrium with polynomial size rationals, whereas games with three or more players may only have equilibria in irrational numbers [15]. The Nash problem is to find a Nash equilibrium in a normal-form game.

In the unknown-input version of a given game, denoted by $\mathsf{Nash}_{\mathsf{u}}$, the payoff functions $u_i(\cdot)$ are unknown. We can query a mixed strategy $(p_1, \ldots, p_n)$ each time. If it is not an equilibrium, then the oracle will return a player i and one of his better responses p_i' where the foregoing inequality fails to hold (note that i and p_i' are precisely the index of

a violated constraint).[7] Our trial and error model considers how fast a Nash equilibrium can be found from the viewpoint of a centralized authority, which is quite different from the learning models investigated in [19, 42] whose focuses are on the strategic dynamics formed by the behavior of individual players. We have the following result.

THEOREM 12. *There is a polynomial-time algorithm solving* Nash_u *for any two-player game, given a computation oracle solving* Nash.

Idea of the proof. The proof is built on the existence of a Nash equilibrium in any game [33]. Assume that each player has m strategies. There are a total of $2m^2$ values in the two payoff matrices. Note that the Nash equilibrium solution space may not be convex; thus, we cannot employ the ellipsoid method to search for an equilibrium in the solution space. One observation is that the $2m^2$ values in the matrices correspond to a point U in the space $\mathbb{R}^{2m^2}$, which can also be seen as a degenerate polyhedron in $\mathbb{R}^{2m^2}$. For any given point $X \in \mathbb{R}^{2m^2}$, we can consider it as two payoff matrices of some game. If we compute a Nash equilibrium with respect to this game using the computation oracle and query it to the verification oracle, then the returned information (if it is not a Yes) actually gives us a hyperplane that separates X from the true point U. It is now tempting to claim that the problem is solved by the ellipsoid method. However, there is a remaining issue: in our problem, the solution polyhedron degenerates to a point and has volume 0. The standard approach in the ellipsoid method for handling such degenerated cases is to add perturbations to the constraints to introduce a positive volume of the feasible solution polyhedron. However, this approach is not applicable in our context, as we do not know the constraints explicitly. Luckily, we are able to employ a much more involved machinery developed by Grötschel, Lovász, and Schrijver [22, 23], solving the strong nonemptiness problem for well-described polyhedra given by a strong separation oracle, to overcome this issue and thus solve the problem. □

Although the aforementioned claim applies only to two-player games, our approach can also be generalized to n players to obtain an ϵ-approximate mixed Nash equilibrium for any constant $\epsilon > 0$. (Note that we cannot hope to compute an exact Nash equilibrium when $n \geq 3$, as such a solution may consist of irrational numbers.) However, there is one potential issue: the input size of a game can be as large as $O(m^n)$, where m is the number of strategies of each player. Thus, for general games, our algorithm may require running time polynomial to $O(m^n)$ (which is still polynomial in the input size). However, for some multi-player games that admit concise representations, e.g., graphical games [29] on constant degree graphs, we can find an ϵ-approximate Nash equilibrium in time polynomial to m and n.

Our result implies that even if players are not completely aware of the rules of a game, we can still find a Nash equilibrium efficiently. Further, even if a Nash equilibrium has been achieved, the game itself can still remain a mystery (because beyond this point, the verification oracle cannot return any further information). The Internet provides one such example, as Scott Shenker remarked: "*The Internet is an equilibrium, we just have to identify the game*" [36].

In addition, we note that our approach can also be adopted to solve some other similar problems such as correlated equilibrium in games and competitive equilibrium in matching markets with prices (the details are quite similar to Nash_u and thus are omitted here).

[7] Note that the full set of strategies S_i may also be unknown; that is, in the process of trials, we query a probability distribution over those strategies that we have already observed. A deviation from a player can be either from the known strategies or "new" unknown strategies.

6. CONCLUDING REMARKS

In this paper, we propose a trial and error model to investigate search problems with unknown input. We consider a number of natural problems, and show that the lack of input knowledge may introduce different levels of extra difficulty in finding a valid solution. Our complexity results range from polynomially solvable, to **NP**-complete and exponential. Our model and results demonstrate the value of input information in solution finding from the computational complexity viewpoint.

The present work showcases a number of algorithms and lower bounds. Meanwhile, a number of important questions are left for future exploration. Closing the small gaps in Theorem 2 and examining more CSP problems are the obvious and specific ones. The following is a list of more problems and directions for further research.

- Information processing on hidden inputs is a common phenomenon in many scenarios, and the present work tries to address the related computational complexity issues in a specific and natural framework. What other general frameworks could be employed for systematic studies of hidden inputs from an algorithmic perspective?
- Our complexity results focus on either the trial or the time cost. It would be intriguing to consider the tradeoff between them. For instance, in $\mathsf{Sorting}_\mathsf{u}$ and $\mathsf{StableMatching}_\mathsf{u}$, our deterministic upper bounds $O(n \log n)$ and $O(n^2 \log n)$ for trial complexity are established by exponential-time algorithms. If we only allow *polynomial time* computation, then we do not have any bound better than $O(n^2)$ and $O(n^3)$ for $\mathsf{Sorting}_\mathsf{u}$ and $\mathsf{StableMatching}_\mathsf{u}$, respectively. (Note that the classic argument of graph entropy for sorting under a partial order [26] is not directly applicable here, as the allowed queries there are of the standard form of pair comparison.)

  On the lower bound side, a natural question is whether any lower bound better than $\Omega(n \log n)$ and $\Omega(n^2 \log n)$ can be proven for $\mathsf{Sorting}_\mathsf{u}$ and $\mathsf{StableMatching}_\mathsf{u}$ with polynomial time computation power. Note that the bound, if possible, would probably be very difficult to prove because it implies that $\#\mathbf{P} \neq \mathbf{FP}$ and thus $\mathbf{P} \neq \mathbf{PP}$. (If $\#\mathbf{P} = \mathbf{FP}$, then counting linear extensions, as a $\#\mathbf{P}$-complete problem, can be computed in polynomial-time, which makes our algorithms for $\mathsf{Sorting}_\mathsf{u}$ and $\mathsf{StableMatching}_\mathsf{u}$ also in polynomial time.)
- It is well known in decision tree complexity that deterministic and randomized complexities can be polynomially separated [13], and a fundamental open question is whether the gap exhibited by the NAND tree [39] is the largest possible. What separation between deterministic and randomized trial complexities could we have in our model? This question could also be considered in the polynomial-time computation framework.

- An algorithm in our model can access two oracles, verification and computation. In this paper, we consider only the complexity that interacts with the verification oracle. It is natural to ask about the query complexity of the other oracle (the problem is of particular importance when the computational complexity of the problem itself is large).

Acknowledgments

This research was supported by the AcRF Tier 2 grant of Singapore (No. MOE2012-T2-2-071) and Research Grants Council of the Hong Kong SAR (Project No. CUHK418710, CUHK419011). We are grateful to Shang-Hua Teng, Leslie Valiant, Umesh Vazirani, and Andrew Yao for their helpful discussions and comments. We also thank Eric Allender, Graham Brightwell, Leslie Goldberg, and Kazuo Iwama for pointing out [40], [11], [12], and [34], respectively.

7. REFERENCES

[1] http://en.wikipedia.org/wiki/Trial_and_error.

[2] D. Angluin. Queries revisited. *Theoretical Computer Science*, 313(2):175–194, 2004.

[3] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[4] V. Arvind and J. Torán. Solvable group isomorphism is (almost) in NP ∩ coNP. *ACM Transactions on Computation Theory*, 2(2):1–22, 2011.

[5] M. Balcan and A. Blum. A discriminative model for semi-supervised learning. *JACM*, 57(3), 2010.

[6] A. Barto and R. Sutton. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[7] D. Beaver, J. Feigenbaum, R. Ostrovsky, and V. Shoup. Instance-hiding proof systems. Technical Report 65, DIMACS, Rutgers University, 1993.

[8] X. Bei, N. Chen, and S. Zhang. On the complexity of trial and error. *arXiv:1205.1183v2*, 2013.

[9] X. Bei, N. Chen, and S. Zhang. Solving linear programming with constraints unknown. *working paper*, 2013.

[10] G. Brightwell. Balanced pairs in partial orders. *Discrete Mathematics*, 201(1-3):25–52, 1999.

[11] G. Brightwell and P. Winkler. Counting linear extensions is #P-complete. In *STOC*, pages 175–181, 1991.

[12] R. Bubley and M. Dyer. Faster random generation of linear extensions. In *SODA*, pages 350–354, 1998.

[13] H. Buhrman and R. D. Wolf. Complexity measures and decision tree complexity: A survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

[14] X. Chen, X. Deng, and S. Teng. Settling the complexity of computing two-player Nash equilibria. *JACM*, 56(3), 2009.

[15] R. W. Cottle and G. B. Dantzig. Complementary pivot theory of mathematical programming. *Linear Algebra and its Applications*, 1:103–125, 1986.

[16] C. Daskalakis, P. Goldberg, and C. Papadimitriou. Computing a Nash equilibrium is PPAD-complete. *SIAM Journal on Computing*, 39(1):195–259, 2009.

[17] B. Davey and H. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2002.

[18] M. Dyer, A. Frieze, and R. Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. In *STOC*, pages 375–381, 1989.

[19] D. Fudenberg and D. Levine. *The Theory of Learning in Games*. The MIT Press, 1998.

[20] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.

[21] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[22] M. Grötschel, L. Lovász, and A. Schrijver. Geometric methods in combinatorial optimization. *Progress in Combinatorial Optimization*, pages 167–183, 1984.

[23] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988.

[24] J. Halpern. Beyond Nash equilibrium: Solution concepts for the 21st century. In *PODC*, pages 1–10, 2008.

[25] A. Indrayan. Elements of medical research. *Indian Journal of Medical Research*, 119:93–100, 2004.

[26] J. Kahn and J. H. Kim. Entropy and sorting. *JCSS*, 51(3):390–399, 1995.

[27] J. Kahn and M. Saks. Balancing poset extensions. *Order*, 1(2):113–126, 1984.

[28] T. Kavitha. Linear time algorithms for abelian group isomorphism and related problems. *JCSS*, 73(6):986–996, 2007.

[29] M. Kearns, M. Littman, and S. Singh. Graphical models for game theory. In *UAI*, pages 253–260, 2001.

[30] M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

[31] J. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[32] D. Montgomery. *Design and Analysis of Experiments*. Wiley, 7th edition, 2008.

[33] J. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951.

[34] C. Ng. Lower bounds for the stable marriage problem and its variants. In *FOCS*, pages 129–133, 1989.

[35] N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.

[36] C. Papadimitriou. Algorithms, games, and the internet. In *STOC*, pages 749–753, 2001.

[37] A. Roth. Deferred acceptance algorithms: History, theory, practice, and open questions. *International Journal of Game Theory*, pages 537–569, 2008.

[38] A. Roth and M. Sotomayor. *Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis*. Cambridge University Press, 1992.

[39] M. Saks and A. Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. In *FOCS*, pages 29–38, 1986.

[40] U. Schöning. Graph Isomorphism is in the low hierarchy. In *STACS*, pages 114–124, 1987.

[41] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[42] H. Young. Learning by trial and error. *Games and Economic Behavior*, 65(2):626–643, 2009.