On Learning and Testing Dynamic Environments
EXTENDED ABSTRACT

Oded Goldreich
Department of Computer Science,
Weizmann Institute of Science
Rehovot, ISRAEL
Email: `oded.goldreich@weizmann.ac.il`

Dana Ron
School of Electrical Engineering
Tel-Aviv University
Ramat-Aviv, ISRAEL
Email: `danar@eng.tau.ac.il`

***Abstract*—We initiate a study of learning and testing dynamic environments, focusing on environment that evolve according to a fixed local rule. The (proper) learning task consists of obtaining the initial configuration of the environment, whereas for non-proper learning it suffices to predict its future values. The testing task consists of checking whether the environment has indeed evolved from some initial configuration according to the known evolution rule. We focus on the temporal aspect of these computational problems, which is reflected in the requirement that only a small portion of the environment is inspected in each time slot (i.e., the time period between two consecutive applications of the evolution rule).**

We present some general observations, an extensive study of two special cases, two separation results, and a host of open problems. The two special cases that we study refer to linear rules of evolution and to rules of evolution that represent simple movement of objects. Specifically, we show that evolution according to any linear rule can be tested within a total number of queries that is sublinear in the size of the environment, and that evolution according to a simple one-dimensional movement can be tested within a total number of queries that is independent of the size of the environment.

***Keywords*-Property Testing, Learning, Multi-dimensional cellular automata.**

I. A BRIEF INTRODUCTION

We initiate a study of sublinear algorithms for testing and learning *dynamic environments that evolve according to a local rule*. That is, the content of the environment in each location and at each time is determined by the contents of the local neighborhood of that location at the previous time.

One archetypical example of such environments is that of a collection of elements that interact at a local level (i.e., each element may change its local state based on the state of its neighbors). Indeed, the model of (two-dimensional) cellular automata was invented and studied as a model for such applications, and one may view our study as a study of sublinear algorithms for testing and learning the evolution of cellular automata. Another archetypical example is that of a collection of objects that move in (three-dimensional) space such that their movements may be affected by collisions (or near collisions) with other objects. Indeed, such motion can also be represented as an evolution of a (three-dimensional) cellular automaton.

The *sublinear aspect* of our model is reflected in the postulate that the algorithm can only probe a small portion of the environment in each time slot, where the environment evolves in time (and is thus potentially different in each time slot). Yet, as stated above, the evolution of the environment is not arbitrary, but is rather based on local rules.

For a full version of this paper see [5]

II. THE BASIC MODEL

The environment is viewed as a d-dimensional grid, mainly for $d \in \{1,2,3\}$, and local rules determine the state of each location as a function of its own state and the state of its neighbors in the previous time unit. (Indeed, time is also discrete.)

The environment's *evolution in time* is captured by a $d+1$ dimensional array, denoted $\mathtt{ENV} : \mathbb{Z}^{d+1} \to \Sigma$, such that $\mathtt{ENV}_j(i_1,...,i_d) \stackrel{\rm def}{=} \mathtt{ENV}(j,i_1,..,i_d)$ represents the state of location $(i_1,...,i_d)$ at time j, and $\mathtt{ENV}_j$ is determined by $\mathtt{ENV}_{j-1}$. The set Σ is an arbitrary finite set of possible local states, and the (instantaneous) environment is viewed as an infinite d-dimensional grid. Actually, we shall restrict $\mathtt{ENV}$ to $[t] \times [n_1] \times \cdots \times [n_d]$ or rather to $[t] \times [n]^d$, and postulate that $\mathtt{ENV}$ contains neutral values outside this domain. (The notion of a neutral value is rule-dependent, but at the very least we require that a cell maintains the neutral value as long as its neighboring cells also hold this value).[1]

An *observer*, who is trying to learn or test the environment, may query its locations at any point in time, but at time $j \in [t]$ it may only obtain values of $\mathtt{ENV}_j : [n]^d \to \Sigma$. That is, the observer is modeled as an oracle machine, but this machine is restricted to make queries that are monotonically non-decreasing with respect to the time value (i.e., the value j in queries of the form $(j,i_1,...,i_d)$). This key feature of the model is captured by the following definition (where x

[1]Jumping ahead in order to exemplify this notion, we mention that zero is a neutral value for linear rules (as in Theorem 3.3) whereas an empty cell is a neutral value for rules that describe moving objects (as in Theorem 3.4).

represents some auxiliary input that the machine may be given).

Definition 2.1: Time-conforming observers. An oracle machine T is said to be time-conforming if, on input (t, n, x) and oracle access to $\texttt{ENV} : [t] \times [n]^d \to \Sigma$, it never makes a query $(j, i_1, ..., i_d)$ after making a query $(j', i'_1, ..., i'_d)$ such that $j < j'$.

In particular, any nonadaptive oracle machine is time-conforming, because its queries can be determined beforehand and made at the appropriate order (i.e., time-wise). This does not mean that time-conforming machines are necessarily nonadaptive (see Theorem 3.2).

In general, when the observer queries location $(i_1, ..., i_d) \in [n]^d$ at time $j \in [t]$, it will retrieve the visible part of the state $\texttt{ENV}_j(i_1, ..., i_d)$, rather than the entire state. That is, the model includes an auxiliary function $V : \Sigma \to \Sigma'$ (called a viewing function) such that $V(\sigma)$ is the visible part of the state σ; hence, the query $(j, i_1, ..., i_d)$ is answered by $V(\texttt{ENV}_j(i_1, ..., i_d))$. We may say that the part of σ not revealed by $V(\sigma)$ is the hidden part of σ. Without loss of generality, we may assume that $\Sigma = Q \times \Sigma'$ and $V(q, \sigma') = \sigma'$ for every $(q, \sigma') \in Q \times \Sigma'$. (In this case q is the hidden part of (q, σ').) We also consider the special case in which the state is fully visible, which is captured by the case that $|Q| = 1$. (In this case, we prefer to view V as an identity function.)

The evolution of the environment is *local* in the sense that the value of $\texttt{ENV}_j(i_1, ..., i_d)$ is determined by the value of $\texttt{ENV}_{j-1}$ in positions $\{(i_1 + s_1, ..., i_d + s_d) : s_1, ..., s_d \in \{-1, 0, 1\}\}$. The rule of determining the value, denoted $\Gamma : \Sigma^{3^d} \to \Sigma$, is known.[2] Thus, $\texttt{ENV}_j(i_1, ..., i_d)$ equals $\Gamma(z_{-1,...,-1}, .., z_{1,...,1})$, where $z_{s_1,...,s_d} = \texttt{ENV}_{j-1}(i_1 + s_1, ..., i_d + s_d)$ and the sequence of all $(s_1, ..., s_d) \in \{-1, 0, 1\}^d$ appears in some canonical order (e.g., lexicographic order, with $(0, ..., 0)$ in the middle). Indeed, *we model the evolution of the environment as a computation of a d-dimensional cellular automaton.*

The computational problems

The computational problems that we consider are (1) testing whether the observed evolution of the environment is actually consistent with a fixed known rule, and (2) learning the entire evolution of the environment (i.e., recovering the (visible part of the) states corresponding to all locations at all times). We refer to the standard notions of property testing (cf. [4], [10], [9]) and PAC learning (cf. [11], [7]), when applied with respect to the uniform distribution on the domain (of the functions in question). The symbol ϵ always denotes the relevant proximity parameter. Since the evolution is determined by the local states at the initial time (i.e., by $\texttt{ENV}_1$), *testing* is equivalent to asking whether the evolution is consistent with the known rule and some initial global state (i.e., $\texttt{ENV}_1$), whereas *proper learning*[3] calls for recovering the initial global state.

Definition 2.2: Testing consistency with Γ as viewed via V. We say that an oracle machine T tests the consistency of evolving environments with respect to $\Gamma : \Sigma^{3^d} \to \Sigma$ and $V : \Sigma \to \Sigma'$ if for every $\texttt{ENV} : [t] \times [n]^d \to \Sigma$ the following holds:

1) If ENV evolves from $\texttt{ENV}_1$ according to Γ, then $\Pr[T^{V \circ \texttt{ENV}}(t, n, \epsilon) = 1] \geq 2/3$, where $T^F(\overline{p})$ denotes the execution of T on parameters $\overline{p}$ when given access to the oracle F and $V \circ \texttt{ENV}$ denotes the composition of V and ENV *(i.e., $(V \circ \texttt{ENV})(j, i_1, ..., i_d) = V(\texttt{ENV}(j, i_1, ..., i_d))$).*
2) If ENV is ϵ-far from any environment $\texttt{ENV}'$ that evolves from the corresponding $\texttt{ENV}'_1$ according to Γ, then $\Pr[T^{V \circ \texttt{ENV}}(t, n, \epsilon) = 1] \leq 1/3$, where the distance between ENV and $\texttt{ENV}'$ equals the fraction of entries on which ENV and $\texttt{ENV}'$ disagree *(i.e., $|\{(j, i_1, ..., i_d) \in [t] \times [n]^d : \texttt{ENV}(j, i_1, ..., i_d) \neq \texttt{ENV}'(j, i_1, ..., i_d)\}|/tn^d$).*[4]

If Condition 1 holds with probability 1, then we say that T has one-sided error probability.

Note that, on top of oracle access to $V \circ \texttt{ENV} : [t] \times [n]^d \to \Sigma'$, the tester gets the (duration and size) parameters t, n and the proximity parameter ϵ as explicit inputs. The same applies to learners as defined next.

Definition 2.3: Learning evolution according to Γ via V. We say that an oracle machine learns the environment evolving according to $\Gamma : \Sigma^{3^d} \to \Sigma$ and viewed via $V : \Sigma \to \Sigma'$ if the following holds: On input (t, n, ϵ) and oracle access to $V \circ \texttt{ENV}$ such that $\texttt{ENV} : [t] \times [n]^d \to \Sigma$ evolves according to Γ, the oracle machine outputs a function $F : [t] \times [n]^d \to \Sigma'$ that is ϵ-close to $V \circ \texttt{ENV}$. The learner is said to be proper if it outputs $\texttt{ENV}'$ such that $\texttt{ENV}' : [t] \times [n]^d \to \Sigma$ is an environment that evolves according to Γ and $V \circ \texttt{ENV}'$ is ϵ-close to $V \circ \texttt{ENV}$.[5]

We seek *time-conforming* oracle machines that solve the corresponding tasks (of testing and learning). Furthermore, we seek testers and learners that solve the corresponding tasks in *sublinear query complexity*, which we interpret as *making $o(n^d)$ queries at each particular time*. In other

[2]The number of possible rules is a function of Σ and d, which is independent of the size of the environment (i.e., n^d). From this perspective (i.e., for fixed Σ and d), it is easy to learn the rule that determines a given evolution.

[3]In general, proper learning a concept class requires obtaining a description that has the same format as functions in the concept class. Indeed, here $\texttt{ENV}_1$ serves as such a description.

[4]Equivalently, we may require that if the "visible environment" captured by $F : [t] \times [n]^d \to \Sigma'$ is ϵ-far from $V \circ \texttt{ENV}'$ for any environment $\texttt{ENV}' : [t] \times [n]^d \to \Sigma$ that evolves (from the corresponding $\texttt{ENV}'_1$) according to Γ, then $\Pr[T^F(t, n, \epsilon) = 1] \leq 1/3$.

[5]Equivalently, we may require the proper learner to output (only) the corresponding $\texttt{ENV}'_1$.

words, we seek machines of sublinear temporal query complexity.

Definition 2.4: Temporal query complexity. The temporal query complexity of an oracle machine querying $\texttt{ENV} : [t] \times [n]^d \to \Sigma$ is the maximal number of queries that the machines makes to each $\texttt{ENV}_j$ *($\forall j \in [t]$).*

Definitions 2.1 and 2.4 capture the time-evolving nature of the environment and our goals, and distinguish the current testing and learning problems from the standard testing and learning problems regarding various structures (including these related to $(d+1)$-dimensional arrays). First, whenever the oracle machine does not query the entire oracle $\texttt{ENV}$, the time-conforming condition restricts its access pattern (i.e., the order in which the machine probes the various entries). Second, the temporal query complexity refers to the number of queries made at each time slot (as compared to n^d), rather than the total number of queries (as compared to $t \cdot n^d$). This requirement reflects the reality of actual observers of natural phenomena, who are not only forced to be time-conforming (since they cannot inspect the past) but are restricted in the amount of inspection they can perform at any time slot.

A natural question is whether the time-conforming requirement actually restricts the power of testers. We show that this is indeed the case (see Theorem 3.1): We demonstrate that the time-conforming requirement makes testing of evolving d-dimensional environments fundamentally different from testing properties of the corresponding $(d+1)$-dimensional array. Specifically, there exist pairs (Γ, V) for which the time-conforming requirement causes a subexponential increase in the query complexity of testers (i.e., an increase from $\text{poly}(\log n)$ to $n^{\Omega(1)}$).

Recall that *proper learning implies testing* (cf. [4, Sec. 3.1]), and note that the argument extends to our setting (i.e., when referring to time-conforming machines and their temporal query complexity).[6] Thus, we present testing results only when they improve over the best possible learning results (which typically require a total number of $\Omega(n^d)$ queries). We note that there exist evolution rules for which testing is not easier than learning, and this holds even if the state is fully visible (see Theorem 3.5).

III. A TASTE OF OUR RESULTS

In this section we provide an overview of the results presented in this work. The stated results are followed by rather laconic comments regarding their proofs. More substantial overviews of these proofs are provided in Section IV, and precise statements, together with complete proofs, can be found in the full version of this paper [5].

[6]Recall that the argument in [4, Sec. 3.1] suggests that the tester first learns a hypothesis, and then checks the hypothesis's validity by an auxiliary sample (which is uniformly distributed in the function's domain). Note that this auxiliary sample can be chosen a priori, and the adequate queries can be made in due time (even before the learning stage is completed).

We start by presenting the two separation results that were mentioned in Section II. Both results are established by using one-dimensional environments (i.e., $d = 1$). Recall that, throughout this paper, the evolving environments are represented as functions from $[t] \times [n]^d$ to Σ. For simplicity, we assume in this section that $t = \Theta(n)$.

The first result establishes the non-triviality of the notion of time-conforming observers by showing that the time-conforming requirement may cause a subexponential increase in the query complexity of testers (i.e., an increase from $\text{poly}(\log n)$ to $n^{\Omega(1)}$).

Theorem 3.1: On the time-conforming requirement. There exist a constant $c > 0$, an evolution rule $\Gamma : \Sigma^3 \to \Sigma$, and a viewing function $V : \Sigma \to \Sigma'$, such that (1) any *time-conforming* tester of evolution according to Γ via V requires $\Omega(n^c)$ queries, but (2) there exists a *(non-time-conforming)* tester of query complexity $\text{poly}(\epsilon^{-1} \log n)$ for this property.

The proof of Theorem 3.1 is based on notions and ideas of Gur and Rothblum [6] concerning sublinear, non-interactive proof-systems, which they refer to as Merlin-Arthur proofs of proximity (MAPs). Specifically, we refer to their notions of general MAPs and MAPs with proof-oblivious queries, and transform the separation between them into a separation between general testers and time-conforming ones. Towards this end, we construct an evolution rule that first reveals an object to be tested, and then deletes the object and reveals a corresponding proof (for a suitable MAP). While a general tester may invoke the corresponding MAP, a time-conforming tester can be transformed into an MAP that makes proof-oblivious queries.

The second separation result asserts that adaptivity is useful also in the context of time-conforming testers.

Theorem 3.2: On the benefits of adaptivity. There exist a constant $c > 0$, an evolution rule $\Gamma : \Sigma^3 \to \Sigma$, and a viewing function $V : \Sigma \to \Sigma'$, such that (1) any *nonadaptive* tester of evolution according to Γ via V requires $\Omega(n^c)$ queries, but (2) there exists a *(time-conforming)* tester of query complexity $O(\epsilon^{-1} \log n)$ for this property.

The proof of Theorem 3.2 is based on the observation that some separations between adaptive and nonadaptive testers that hold in the standard model can be translated to analogous results regarding testing evolving environments. Our translation requires the existence of an efficient algorithm for sampling the property that is used in the separation result (of the standard model). This sampler need not produce the uniform distribution over objects having the property, but the support of its output distribution should equal the set of all objects having this property.

Turning to the actual study of testing evolving environments, we first note that, in general (i.e., for arbitrary evolution rules $\Gamma : \Sigma^3 \to \Sigma$, even for $d = 1$), testing may require as many queries as learning (cf. Theorem 3.5) and its computational complexity may be NP-Hard (as we show in the full version of this paper [5]). We thus focus our attention on *special classes* of evolution rules. In two natural cases, we obtain testers of lower query complexity than the corresponding learners. Furthermore, these testers are efficient (i.e., their computational complexity is closely related to their query complexity). The first class of evolution rules that we consider is the class of linear rules.

Theorem 3.3: Sublinear time complexity for testing linear rules. For any $d \geq 1$ and any field Σ of prime order there exists a constant $\gamma < d$ such that the following holds. For any linear rule $\Gamma : \Sigma^{3^d} \to \Sigma$ there exists a time-conforming oracle machine of *(total)* time complexity $\mathrm{poly}(\epsilon^{-1}) \cdot n^{\gamma}$ that tests the consistency of an evolving environment with respect to $\Gamma : \Sigma^{3^d} \to \Sigma$ and the identity viewing function *(i.e., $V(\sigma) = \sigma$ for every $\sigma \in \Sigma$)*. Furthermore, the tester is nonadaptive and has one-sided error probability.

The proof of Theorem 3.3 is based on proving that, on the average, the value of a random location in the evolving environment (i.e., $\texttt{ENV} : [t] \times [n]^d \to \Sigma$) depends only on $O(n^{\gamma})$ locations in the initial configuration. We note that our upper bound on the time complexity is only mildly lower than $\mathrm{poly}(1/\epsilon) \cdot n^d$ (e.g., for $d = 1$ and $|\Sigma| = 2$ we obtain the bound $O(n^{0.8}/\epsilon)$), and we wonder whether $\mathrm{poly}(\epsilon^{-1} \log n)$ complexity is possible (for all linear rules).

The second class of evolution rules is aimed at capturing the movement of objects in a d-dimensional grid. The following result refers to the case of $d = 1$ and to objects that move in a fixed-speed but stop whenever a collision occurs (i.e., an object continues moving, one cell at a time, until it collides with another object, and when this happens the object stops).

Theorem 3.4: Testing interruptible moving objects, very loosely stated. Let $\Gamma : \Sigma^3 \to \Sigma$ be a local rule that captures the fixed-speed movement of objects in one dimension such that colliding objects stop forever. Then, there exists a time-conforming oracle machine of *(total)* time complexity $\mathrm{poly}(1/\epsilon)$ that tests the consistency of evolving environments with respect to $\Gamma : \Sigma^3 \to \Sigma$ and the identity viewing function.

The tester referred to in Theorem 3.4 makes quite a few checks, which include checking that individual objects move in fixed speed as long as they don't stop, checking that these objects do not cross each other, and checking global statistics regarding the number of moving and stopping objects within some intervals at some times. The non-triviality of this testing task is reflected in the fact that the tester has *two-sided error probability, and that this is unavoidable for testers of query complexity that is independent of n*. The latter assertion is a corollary of a theorem proved in the full version of this paper [5], which asserts that *any nonadaptive tester of one-sided error probability for this task must have query complexity $\Omega(\sqrt{n})$*, which in turn implies an $\Omega(\log n)$ bound for general testers (of one-sided error probability). We note that the tester used in the proof of Theorem 3.4 is actually nonadaptive.

As stated above, we show that, in general, testing evolving environments may be as hard as learning them. That is, in contrast to (the rules considered in) Theorems 3.3 and 3.4, there are evolution rules for which testing is not easier than learning. This hardness result holds even when the state is fully visible, which indicates that this restriction (i.e., fully visible states) does not suffice for making testing easier than learning.

Theorem 3.5: Testing may have the same query complexity as learning. There exist a constant $c > 0$ and an evolution rule $\Gamma : \Sigma^3 \to \Sigma$ such that both testing evolution according to Γ via V and *(proper)* learning evolution according to Γ via V have *(total)* query complexity $\Theta(n^c)$, where in both cases we refer to a fully visible state *(i.e., V is the identity function)* and to all sufficiently small constant values of $\epsilon > 0$.

As in the proof of Theorem 3.2, the main observation in the proof of Theorem 3.5 is that results that hold in the standard model (in this case relations between the complexity of testing and learning) can be translated to analogous results regarding testing evolving environments. However, in the current proof, we wish to carry out this translation in the context of fully visible states. Thus, we pick a property for which probing the process of the construction of the object (having the property) does not reveal more than probing the object itself.

Staying within the realm of fully visible states, the following result assert that, even in this case, the computational complexity of testing may be NP-Hard (with respect to the size of the environment), provided that the temporal query complexity is "significantly sublinear" (where $f(m)$ is significantly sublinear if $f(m) < m^{1-\Omega(1)}$).

Theorem 3.6: on the computational complexity of testing with sublinear temporal query complexity. Assuming $\mathcal{NP} \not\subseteq \mathcal{BPP}$, there exists an evolution rule $\Gamma : \Sigma^3 \to \Sigma$ such that no time-conforming probabilistic polynomial-time tester for the evolution *(of n-sized environments)* according to Γ and $V_{\equiv}$ has temporal query complexity $n^{1-\Omega(1)}$, where $V_{\equiv}$ denotes the identity mapping.

Indeed, Theorem 3.6 stands in contrast to Theorems 3.3 and 3.4, which refer to specific evolution rules (and a fully visible state). We mention that in the case of a viewing function that hides part of the state, NP-Hardness of testing holds regardless of the query complexity.

IV. More on the ideas underlying our proofs

In this section we attempt to give a flavour of the ideas underlying the proofs of our results, going beyond the laconic comments provided in the prior section.

A. On the proofs of our separation and hardness results

Our separation and hardness results (i.e., Theorems 3.1, 3.2, 3.5, and 3.6) are all based on the fact that (1-dimensional) cellular automata can emulate (1-tape) Turing machines (while preserving the time complexity), and thus can generate objects that are hard to test in some sense. The proofs differ by issues such as which computation is emulated and under which circumstances it is emulated.

The basic approach: As a warm-up, let us consider the following claim, which we'll refer to as Claim (*): *Testing the consistency of an evolution with respect to a fixed rule* $\Gamma : \Sigma^3 \to \Sigma$ *may be NP-hard, regardless of the query complexity.* (A precise statement appears as Theorem 2.1 in the full version of this paper [5]). The basic idea is to design a rule Γ such that evolutions that are consistent with Γ reveal (via the viewing function) an error-corrected encoding of a string if and only if the string is in the NP-set. Hence, membership in the NP-set S can be decided by invoking the tester, while answering its queries (and relying on the fact that strings that are not in S have encodings that are far from the encoding of any string in S). Specifically, we let Γ emulate the verification procedure associated with an NP-witness relation of S, and output (an encoding of) the main input (but not the NP-witness) if the procedure accepts (and output an empty string otherwise). The emulation is carried out using the hidden part of the state, and the output is revealed through the visible part of the state. (The output is maintained for a sufficient number of steps so evolving environments that output encodings of different strings are far apart (i.e., at constant relative distance of one another).)

On input x, the decision procedure for S invokes the tester, while answering queries that correspond to the emulated computation by an empty symbol (which matches the expected output of the viewing function) and answers the other queries by using bits in the encoding of x. The key observation is that an evolving environment that "outputs" an encoding of x is legal if and only if there exists an NP-witness w for x (i.e., iff $x \in S$). The distance between evolving environments that output encodings of different strings guarantees that evolving environments that output a string not in S are far from any legal evolution.

The simple argument outlined above relies in an essential manner on the use of a viewing function that hides information. This function reveals the encoding of x, but it does not reveal the emulation of the computation that maps (x, w) to this encoding. Moreover, this simple argument does not allow to separate different query models as asserted in Theorems 3.1 and 3.2. Still, the underlying principle of generating an object that is hard to test in some sense will be pivotal to all subsequent proofs. But the generation process in these proofs will be less straightforward than in the above case.

Handling the case of a fully visible state: Note that Claim (*) cannot possibly hold when the state is fully visible, since in this case testing consistency of evolutions with a fixed rule is trivial when there are no restrictions on the query complexity. Indeed, Theorem 3.6 refers to testers of limited query complexity (i.e., sublinear temporal query complexity), and its proof capitalizes on this limitation. In the corresponding cellular automaton, many emulations of the above type take place in parallel, but the temporal query complexity imposed on the tester does not allow it to probe the "informative time" (i.e., the time when the NP-witness is visible) of almost all of these emulations. Specifically, the evolution rule Γ partitions $[n]$ into many blocks (i.e., more blocks than the temporal query complexity), emulates the verification of an NP-witness in each block, identifies the first non-empty output obtained in some block (if such exists), and copies its contents (which is a codeword) to all other blocks. (Otherwise, the evolution maintains an empty output in all blocks.)

We consider evolutions in which a single informative emulation (i.e., an emulation of the verification of a real NP-witness) takes place in one block, selected at random, while dummy emulations (which emulate verification with dummy values) take place in all other blocks. The point is that a time-conforming tester is unlikely to hit the informative block at the informative (emulation) time, whereas if the informative block produces an encoding of a string, then this encoding will be propagated to all other blocks. Hence, the tester should determine whether an evolution that repeats an encoding of x is legal or not, which means that this tester decides membership of x in an NP-set, whereas the tester is highly unlikely to probe the process in which the encoding of x was generated by verifying a valid NP-witness.

Specifically, on input x, the decision procedure invokes the tester, while answering queries that correspond to the emulated computations (in the various blocks) according to the verification of a dummy NP-witness for x (which indeed leads the verification procedure to reject in each block), and answers the other queries by using bits in the encoding of x (which is being copied from a random block that was not probed by the tester during the verification stage).[7] That is, the decision procedure emulates a seemingly illegal evolution; yet, if x is in the NP-set (and the tester did not probe the informative block at the informative time), then the tester's view is consistent with a legal evolution that outputs (the encoding of) x. On the other hand, if x is not in the set, then an evolution that outputs the encoding of x is far from being legal.

The forgoing arguments relied on the ability of an ordinary procedure (i.e., the decision procedure we obtain for

[7]Actually, the decision procedure selects the informative block at random, and aborts the emulation if the tester probes this block during the verification stage. Otherwise, it continues the emulation as if only this block has output the encoding of x.

NP-sets) to answer queries to various fictitious evolutions (or to evolutions taken from a restricted set of possible evolutions). That is, the decision procedure partially emulates the execution of hypothetical testers while providing them with oracle access to fictitious (or restricted) evolutions. This strategy is not open to us when we want to distinguish different types of machines that query a real evolving environment (i.e., different types of testers or testers versus learners). In this case, we must construct oracle machines that actually probe a real evolving environment, and not merely foil hypothetical machines by presenting them with fictitious (or restricted) evolutions that we construct.

Consider, for example, the assertion of Theorem 3.5 by which testing requires asymptotically as many queries as (proper) learning. Here we have to present a lower bound on the complexity of testing and match it with an upper bound on the complexity of learning. Furthermore, we claim these bounds for the case of fully visible states. The basic idea is to consider a cellular automaton that computes the inner product (mod 2) of two binary vectors, while placing the vectors and the result in an error correcting form (so as to generate a distance between different input-output pairs). Using the communication complexity method of [1] (see also [3]), one may show that testing such outcomes requires linear query complexity. But, since we deal with fully visible states, we need to establish this lower bound also with respect to testers that may query the process of computing the outcome. We first note that the process of encoding the individual vectors poses no additional difficulties, because probing this process only provides values that are functions of one of the vectors (rather than functions of both vectors).[8] Hence, we focus on the process of computing the inner product (mod 2) of the two vectors, and show that this process leaks no additional information when we reduce from the communication complexity of (Unique) `Disjointness` (rather than from `Inner Product`). Specifically, we use that fact that if a pair of vectors has no common 1-entry, then all partial inner products are zero (whereas if the pair has a single common 1-entry, then its inner product (mod 2) is 1).

On separation results regarding various types of testers: We now turn to the separation results (i.e., Theorems 3.1 and 3.2), which are proved using a viewing function that hides parts of the state. We start with Theorem 3.2, which asserts a separation between nonadaptive testers and (time-conforming) adaptive testers. The idea here is to present a cellular automaton that can generate (and maintain) instances of some property that is easy to test adaptively but hard to test nonadaptively. We need a process that can generate each string that has the property, but never generates a string that does not have the property. We do *not* need this process to generate these strings with uniform distribution (over the property); any distribution that assigns non-zero weight to each string that has the property will do. For example, we can use a process that generates a random 4-partition of $[v]$ (or rather maps the set of all $v \log_2 v$-bit strings to the set of all such partitions), and consider a 3-regular graph that consists of the corresponding isolated 4-cliques. This property is easy to test in the bounded-degree graph model (i.e., a constant number of queries suffice), but it cannot be tested by a nonadaptive machine that makes $o(\sqrt{v})$ queries [8].

Turning to the proof of Theorem 3.1, recall that this result asserts a gap between the power of (adaptive) time-conforming testers and the power of testers that are not time-conforming. The idea here is to rely on notions and ideas of Gur and Rothblum [6], which we re-interpret in terms of "order of events". Specifically, we refer to their notions of general MAPs and MAPs with proof-oblivious queries, where MAPs are testers that obtain (short) auxiliary proofs. We view proof-oblivious MAPs as testers that are restricted to first query the object and only later read the auxiliary proof (without having further access to the object). In contrast, w.l.o.g., general MAPs first read the (entire) auxiliary proof, and then query the object based on the proof just read. Hence, in the proof-oblivious case the queries are made before the proof is read, whereas in the general case the queries are made after the proof is read. Wishing to capitalize on the gap (established in [6]) between the power of the two models, we present a cellular automaton that first makes visible a string to be tested, and then hides (or deletes) this string and reveals the corresponding short proof. The point is that a time-conforming tester is restricted in a manner analogous to a proof-oblivious MAP, whereas a general tester (which is not time-conforming) can first read the proof and then inspect the object (just as a general MAP).

We implement the above strategy by presenting a cellular automaton that proceeds as follows. On input a string x and an index $i \in [|x|]$, which are not visible, the automaton first reveals an error correcting form of x. Next, it computes the i^{th} bit of x (i.e., x_i), and an error correcting version of i (in a secret manner), and deletes all trace of x. Finally, it reveals the encoding of i and the bit x_i, and repeats them for enough time. The legality of an evolution is easy to test by a logarithmic number of queries that violate the time-conforming condition. To show that a time-conforming tester requires an exponentially larger query complexity, we use a reduction from `Disjointness` (again via the method of [1]). Here, we reduce a standard testing problem that refers to a pair of strings (presented in error correcting format) to two instances of testing the consistency of evolving environments with the foregoing evolution rule. Actually, we reduce testing via proof-oblivious MAPs of a property related to `Disjointness` to testing two evolutions; specifically, the property is the set of all pairs of codewords that

[8]Such a situation is easy to handle when using the formulation of [3] (rather than that of [1]).

encode a pair of strings that has a common 1-entry.

Establishing all results for $t = O(n)$: All the above constructions can be implemented when the evolution time (i.e., t) is polynomial in the size of the environment (i.e., n), since all of them are based on evolutions that emulate polynomial-time computations of 1-tape Turing machines. In order to obtain results that refer to the case that $t = O(n)$, we emulate many computations as above in parallel such that each computation takes place on a block of length ℓ, where the emulation time on ℓ-bit inputs is $\Theta(n)$. The details involve performing some simple manipulations in linear time (on a one-dimensional cellular automata); for example, the n cells of the automaton can be partitioned into ℓ-cell blocks in $O(n)$ time, and an ℓ-bit string can be copied to the neighboring block in $O(\ell)$ steps of such an automaton.

B. On testers for special cases

We now turn to our positive testing results, which correspond to two classes of evolution rules.

Testers for linear rules: The first result (i.e., Theorem 3.3) refers to the class of all linear rules (over a finite field of prime cardinality). The tester operates by picking a random location $i \in [n]^d$ at a random time $j \in [t]$ and comparing the value of $\mathtt{ENV}_j(i)$ to a predetermined linear combination of the values of the initial configuration (i.e., $\mathtt{ENV}_1(\cdot)$). The crucial fact is that this linear combination is typically sparse, which implies that this tester has sublinear query complexity (i.e., it makes $(n^d)^{1-\Omega(1)}$ queries). Specifically, we prove that on the average, over all times $j \in [t]$, the contents of a cell depends on a sublinear number of locations in the initial configuration (i.e., $(n^d)^{1-\eta}$ locations, for some $\eta > 0$). This is proved by first showing that each location $i \in [n]^d$ at time $j \in [t]$ depends only on 3^d locations at time $j - p^e$, where p is the cardinality of the field and $e \in \mathbb{N}$. Furthermore, each of these locations has the form $i + p^e \cdot \delta$, where $\delta \in \{-1, 0, 1\}^d$ (i.e., in the case $d = 1$, these locations are $i - p^e$, i and $i + p^e$). Using this fact, we upper bound the number of locations at times $j - j'$ for $j' \in [p^e]$ that influence the fixed location at time j, by employing a careful accounting of all influences.

We comment that using a similar accounting, one can show that on the average, over all times $j \in [t]$, the contents of a cell does depend on $(n^d)^{\eta'}$ locations in the initial configuration, for some $\eta' > 0$, provided that $t = \Omega(n)$ and the linear rule is not degenerated. This means that, except for linear rules that depend on at most one variable, the above tester has query complexity at least $n^{\Omega(1)}$.

Testers for environments of moving objects: The second result (i.e., Theorem 3.4) refers to a rule that describes the interruptible fixed-speed movement of objects in one dimension. The result asserts a two-sided error tester of complexity that does not depend on the size of the environment. This is complemented by a negative result that asserts that such a complexity is impossible in the case of one-sided error testers. Indeed, our tester performs several checks, and one of these checks refers to a global statistics (and not to the consistency of a partial view with the rule of movement). The core of the analysis boils down to showing that an evolution that passes all checks with high probability must be close to a legal one. This is proved by a sequence of modifications such that each set of modifications relies on a different check, showing that if the environment passes this check with high probability then it is close to one that satisfies the corresponding sub-property. Details follow.

We decouple the property of legal evolutions of environments according to the foregoing rule into the following four conditions:[9] (1) the movement and standing behavior of each object is consecutive; (2) each object appears as standing just after it stopped moving; (3) objects do not cross each other paths; and (4) objects stop due to an object that occupies the neighboring cell (in their direction of movement). Note that Condition (1) only says that each object moves in some time interval $[1, j]$ and/or stand in some time interval $[j', t]$, but it does not say that if an object stopped moving in location i at time j then it will appear as standing at location i in time $[j + 1, t]$. As indicated by the one-sided error lower bound, it is infeasible to check the latter ("matching") condition for individual objects. Instead, the checking of Condition (2) is performed "globally"; that is, by comparing the statistics regarding the movement-stopping times and the standing-start times of all objects. Checks that correspond to Conditions (1) and (3) are performed in a rather straightforward manner, and checking Condition (4) relies on a characterization of the intervals that must be filled with standing objects, given a specific pattern of movement stopping.

It is rather easy to see that a legal evolution passes all checks with high probability, whereas the small error probability is due to the statistical checking of Condition (2). The difficult part is proving that if an evolution passed all checks with high probability then it is close to a legal one. This is shown by a sequence of four modifications, which correspond to the above four checks. For example, it is quite easy to see that if the evolution passes the first check with high probability, then it must be close to one that satisfies Condition (1). Proving an analogous claim for the other three checks (and corresponding conditions) is more complex. In particular, we should make sure that the modifications that we perform in later stages do not violate the conditions established in prior stages. For example, the matching of movement and standing should be performed while maintaining Condition (1) that asserts that each such behavior occurs in a consecutive interval of time.

[9]Another condition (i.e., "spaced beginning") was omitted in the current (high-level) description. In fact, the evolution rule does not allow moving objects to start at neighboring locations, and this condition is checked too. The reason for this augmentation is discussed in the full version of this paper [5].

Acknowledgments

We were inspired by a short presentation of Bernard Chazelle in the Property Testing Workshop that took place in January 2010 at Tsinghua University (Beijing).[10] Specifically, Bernard suggested attempting to provide a sublinear time analysis of dynamic systems, which may consist of selecting few objects and tracing their movement in time. This suggestion sounded very appealing to us, and it was the trigger for the model presented here.

We are grateful to Benny Applebaum for collaboration in early stages of this research. We wish to thank a few anonymous readers for their comments.

This research was partially supported by the Israel Science Foundation (grant No. 671/13).

References

[1] E. Blais, J. Brody, and K. Matulef. Property testing lower bounds via communication complexity. *Computational Complexity*, 21(2):311–358, 2012.

[2] O. Goldreich, editor. *Property Testing: Current Research and Surveys*. Springer, 2010. LNCS 6390.

[3] O. Goldreich. On the communication complexity methodology for proving lower bounds on the query complexity of property testing. Technical Report TR13-073, Electronic colloquium on computational complexity (ECCC), 2013.

[4] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.

[5] O. Goldreich and D. Ron. On learning and testing dynamic environments. Technical Report TR14-029, Electronic colloquium on computational complexity (ECCC), 2014.

[6] T. Gur and R. Rothblum. Non-interactive proofs of proximity. Technical Report TR13-078, Electronic colloquium on computational complexity (ECCC), 2013.

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

[8] S. Raskhodnikova and A. Smith. A note on adaptivity in testing properties of bounded degree graphs. Technical Report TR06-089, Electronic Colloquium on Computational Complexity (ECCC), 2006.

[9] D. Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, 5:73–205, 2010.

[10] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

[11] L. G. Valiant. A theory of the learnable. *CACM*, 27(11):1134–1142, November 1984.

[10]A related collection of extended abstracts and surveys has appeared as [2].