# Nonlinear System Identification Using Coevolution of Models and Tests

Josh C. Bongard and Hod Lipson

***Abstract*—We present a coevolutionary algorithm for inferring the topology and parameters of a wide range of hidden nonlinear systems with a minimum of experimentation on the target system. The algorithm synthesizes an explicit model directly from the observed data produced by intelligently generated tests. The algorithm is composed of two coevolving populations. One population evolves candidate models that estimate the structure of the hidden system. The second population evolves informative tests that either extract new information from the hidden system or elicit desirable behavior from it. The fitness of candidate models is their ability to explain behavior of the target system observed in response to all tests carried out so far; the fitness of candidate tests is their ability to make the models disagree in their predictions. We demonstrate the generality of this *estimation-exploration algorithm* by applying it to four different problems—grammar induction, gene network inference, evolutionary robotics, and robot damage recovery—and discuss how it overcomes several of the pathologies commonly found in other coevolutionary algorithms. We show that the algorithm is able to successfully infer and/or manipulate highly nonlinear hidden systems using very few tests, and that the benefit of this approach increases as the hidden systems possess more degrees of freedom, or become more biased or unobservable. The algorithm provides a systematic method for posing synthesis or analysis tasks to a coevolutionary system.**

***Index Terms*—Coevolution, evolutionary robotics, gene network inference, grammar induction, nonlinear topological system identification.**

## I. Introduction

SYSTEM identification is a ubiquitous tool in both science and engineering. Given some partially hidden system, the experimenter applies a series of intelligently formulated experiments to the system in order to learn more about it [44] or make it produce a desired output [27]. The experiment is often described as a set of input to the system, and the resulting behavior of the system given this input is described by a set of output values. The result of the system identification process is a model that describes the salient features of the system itself.

There are two ways to approach the model inference process: A batch (offline) approach and an iterative (online) approach.

Manuscript received June 16, 2004; revised November 9, 2004. This work was supported in part by the U.S. Department of Energy under Grant DE-FG02-01ER45902 and in part by the NASA Program for Research in Intelligent Systems under Grant NNA04CL10A. This research was conducted using the resources of the Cornell Theory Center, which receives funding from Cornell University, New York State, federal agencies, foundations, and corporate partners.

The authors are with the Computational Synthesis Laboratory, Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14853 USA (e-mail: JB382@cornell.edu; HL274@cornell.edu).

Digital Object Identifier 10.1109/TEVC.2005.850293

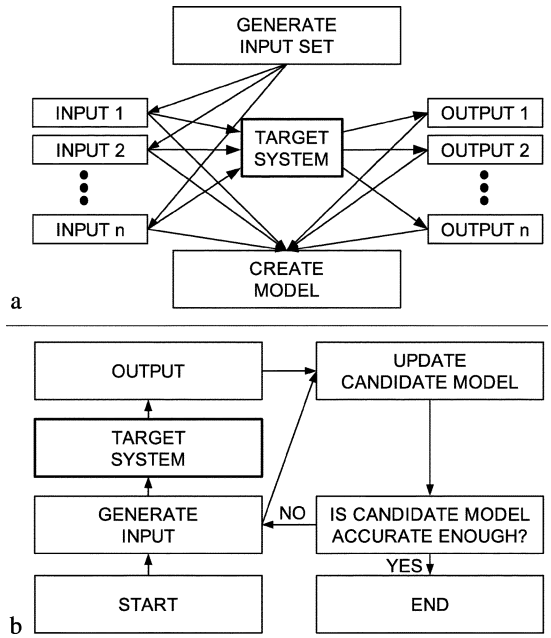


Fig. 1. Two approaches to system identification. (a) A batch approach: a set of input data is generated before identification begins; the hidden system generates output for each input; and the resulting corpus of input/output pairs is used to construct a model of the hidden system. (b) An iterative approach, in which a continual model of the hidden system is refined based on each new input/output data pair.

The batch approach [shown in Fig. 1(a)] involves first generating a set of input vectors and obtaining the corresponding set of output vectors, and then generating a model that correctly predicts the observed outputs for the given inputs. Many data-intensive machine-learning methods operate in this way; it is mostly suitable when data is freely available or can easily be collected, but cannot be controlled. Examples include the evolution of models to explain stock market data, or models of gene coactivation from microarray data.

In many application areas, however, data is critically limited, expensive, or may be biased in unknown ways. In such cases, it is worthwhile to intelligently formulate new experiments to deliberately collect useful data. In the iterative approach [shown in Fig. 1(b)], the inference process generates an input vector, obtains the resulting output vector, and uses the input/output vector pair to improve the original model. This process is iterated until a sufficiently accurate model is obtained. Importantly, in the iterative approach the current model can be used to guide the selection of new input vectors that will extract the most new information about the hidden system given what is already known.

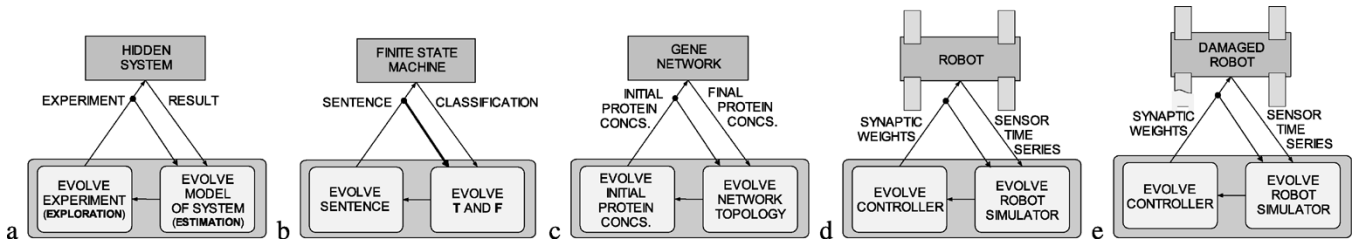


Fig. 2. Applying the algorithm to different problem domains. (a) The general organization of the estimation-exploration algorithm. (b) Applying the algorithm to the problem of grammar induction. (c) Applying the algorithm to the problem of gene network inference. (d) Applying the algorithm to evolutionary robotics. (e) Applying the algorithm for automated recovery after unanticipated robot malfunction.

The field of identification for control [27] involves the generation of a possibly incomplete model solely for the purpose of control, i.e., the correctness of the model itself is important only insofar as it enables the derivation of a useful controller. However, in this field, as in system identification, data is passively collected from the target system; the model is not used to determine which experiment to perform next. Also, it has been noted [27] that no methods exist yet for the automated creation of a controller, given a set of approximate models of the target system. This paper presents an advance toward such automation.

By intelligently choosing input vectors, the number of experiments that need to be performed on the target system in order to create a sufficiently accurate model of it can be greatly reduced. Similarly, any biases present in batch-generated data can be actively compensated for by deliberately asking for new tests. This is important in domains in which it is expensive, risky or time-consuming to perform experiments, or in which experiments alter the structure of the hidden system, thus complicating the inference process.

The method described here is a form of *active learning*. This machine-learning approach actively seeks out tests that will improve the generalization ability of a classifier or learner [16]. However, in typical active learning methods, the "model" of the hidden system is simply a mapping that best translates the input data into the observed output data: It does not mirror the internal structure of the hidden system. The algorithm presented here synthesizes an explicit model of the hidden system using intelligently selected tests. This explicit model can then be used to learn about possible causalities in the hidden system, to localize some change in the system (such as a broken part), or to use that localized information to generate desired behavior (such as recovery of function by circumventing the broken part).

In this paper, we introduce a coevolutionary algorithm, which we call the *estimation-exploration algorithm*, that automates both model inference and the generation of useful experiments. We use an iterative (online) approach, and maintain two coevolving populations: One that evolves candidate models, and one that evolves tests. The fitness of candidate models is their ability to explain the behavior of the target system observed in response to all tests carried out so far; the fitness of candidate tests is their ability to make the models disagree in their predictions.

The estimation-exploration algorithm can be implemented in a variety of ways: The two populations can coevolve in parallel (in steady state) or in a two-phase cycle; models and tests can be evaluated against current populations or against their histories, and models and tests can be evolved not just to infer a system but also to improve its performance. Here, we describe a number of implementations of the algorithm and its application to four different problems: inferring finite-state automata; inferring genetic regulatory networks; improving behavior transferal from a simulated to a target robot; and allowing robots to automatically diagnose and recover from unanticipated malfunctions.

The next section introduces the estimation-exploration algorithm. The following four sections describe each application in turn. The last section provides discusses general properties of the algorithm and avenues for further study.

## II. Estimation-Exploration Algorithm

The estimation-exploration algorithm is comprised of two populations: The estimation population, which evolves improvements to models of the hidden system, given pairs of input/output data obtained from the system; and the exploration population, which evolves intelligent tests to perform on the hidden target system using the best models so far. A cyclical implementation of the algorithm comprises two phases: The estimation phase and the exploration phase. The algorithmic flow of the algorithm is given in Fig. 3.

The estimation phase begins with an initial population of candidate models, which can be random, blank, or seeded with some prior knowledge about the target system. The exploration phase evolves tests that will serve one or possibly two functions: First, tests should extract as much information about the internal structure of the system as possible, given what is already known about the system by the candidate models. Second, tests should elicit some useful behavior from the target system. Once such a test is evolved, it is applied to the target hidden system and some output is obtained. Using this new input/output data pair, plus any additional input/output pairs obtained during previous tests, the estimation phase evolves a better model that can better reproduce the observed output data, given the input data. The new candidate models are then passed to the exploration phase. The cycle continues until either a sufficiently accurate model of the hidden system is obtained, or a test eventually elicits some desirable behavior from the target system. A schematic representation of the algorithm is shown in Fig. 2(a).

### A. Coevolution

There has been much interest in coevolutionary algorithms within the evolutionary computation community, as evidenced by the recent literature (e.g., [12], [33], [49], [52], [54], and

1) **Characterization of the target system (Preliminary Steps)**
   a) Define a representation, variation operators and similarity metric for the space of systems.
   b) Define a representation and variation operators for the space of inputs (tests).
   c) Define a representation and similarity metric for the space of outputs (results).
2) **Initialization**
   a) If an approximate model is available, goto 3.
   b) Otherwise create a random test, evaluate it on the target system, store the results, add it to the (currently empty) test suite and goto 4.
3) **Exploration Phase**
   a) Create a random population of tests.
   b) Evolve them against either best model(s).
   c) The fitness of a test is either its ability to induce disagreement among the models, or to elicit desirable behavior from the model.
   d) Carry out the best test on the target system, store the results, add it to the test suite, and goto 4.
4) **Estimation Phase**
   a) Evolve candidate models; encourage diversity.
   b) Fitness of a model is its ability to explain all input-output data seen so far.
   c) When evolution finishes, send the best model to 6. for validation, and send the best model(s) to 3.
   d) If the most recent test is not sufficiently explained by the models, remove the test from the test suite, and store for later use when the models are more accurate.
5) **Termination**
   a) Cycle through the algorithm until the population of models converges on a sufficiently accurate solution, the target system exhibits some desired behavior, or a fixed number of physical or model evaluations have been conducted.
   b) If no model is found, the search space may be inappropriate, or the target system may be inconsistent.
   c) If no good test is found, the target system may be partially unobservable.
6) **Validation**
   a) Validate the best model(s) using unseen inputs.

Fig. 3. Estimation-exploration algorithm outline.

[58]), starting with the seminal work of Hillis [29] on sorting networks. Contrary to conventional evolutionary systems, coevolutionary systems consist of one or more populations, where individuals may influence the *relative ranking* of others individuals [12]. For example, whether individual $A$ is inferior or superior to individual $B$ may depend on a third individual $C$ rather than on some external fitness metric that provides an absolute ranking. There are a number of different forms of coevolution: Antagonistic coevolution (e.g., predator–prey), cooperative coevolution (e.g., symbiosis) or nonsymmetric systems (e.g., host–parasite or teacher–learner [22]).

In the discussion of coevolutionary systems, it is important to distinguish between the notion of *objective* fitness versus *subjective* fitness. Objective fitness is the well-defined absolute fitness metric used in classical evolutionary algorithms. Subjective fitness is the fitness as defined by coevolving individuals, which may be only weakly correlated with the objective fitness and may sometimes even be misleading. A coevolving individual only knows its subjective fitness. In the examples presented in this paper, we show absolute fitness only for benchmarking purposes: the algorithm itself has no access to the absolute fitness, as in a realistic application, we do not know how close the model *really* is to the hidden target system; rather it only has indirect evidence by comparing input–output sets.

Implementations of coevolution are notoriously difficult, and have been plagued with a number of pathologies arising from complex coevolutionary dynamics [34], [62]. Much of the focus of current research has been to address these drawbacks [15], [33], [54], [58]. However, hybrid coevolutionary algorithms, such as the one proposed in this paper, are less affected by these pathologies due to the anchoring effect of the stable target system.

*Pathology I: The Red Queen Effect:* One related set of pathologies derives from the purely subjective measure of fitness in pure coevolutionary systems, and is known as the Red Queen effect: Two populations continuously adapt to each other and their subjective fitness improves, but they fail to make any consistent progress along the objective metric. Conversely, they do make progress along an objective fitness but their subjective fitness does not reflect this and falsely indicates lack of progress. Both of these effects may initially occur in our system: A series of difficult tests may decrease the subjective fitness of models while in fact they are improving in their objective fitness. Alternatively, a series of biased tests may give rise to overspecialized models which seem to be doing well at explaining the available data (high subjective fitness) but in fact are departing from the true model (lowering their objective fitness).

Both of these effects are transient in our method because of the anchoring provided by the fixed target system. As the true objective correctness of models improves, they are ultimately able to describe test data and, therefore, their subjective fitness also increases. Conversely, arbitrary overspecialization is removed because it is a source of disagreement among models and is, therefore, challenged by new tests.

*Pathology II: Cycling and Transitive Dominance:* Because the subjective fitness criteria is changing over time, individuals may "forget" previous abilities, and then rediscover them later, yielding a cycling performance. Similarly, an individual may be superior to a second individual who is at the same time superior to it according to a different subjective metric. Cycling and transitive dominance problems are eliminated by the fixed target system, because all models need to explain all data so far in addition to any new data.

*Pathology III: Disengagement:* Disengagement occurs when one population is entirely superior to another population. The subjective fitnesses of both populations then become constant and all selection pressure is lost, resulting in drift. In system identification disengagement may occur when a test may be presented which is too difficult for the model population to explain, and all models get an equally low subjective fitness. Although disengagement was not observed in the four applications presented here, it has occurred in more recent experiments. We have recently [6], [63] proposed several mechanisms that successfully prevent disengagement from occurring in the estimation-exploration algorithm. The first mechanism is the "test bank" [6] in which difficult tests are withdrawn from the test suite and are only reintroduced when models become accurate enough to explain them. The second mechanism [63] involves searching explicitly for lower difficulty tests by looking for tests which create a less disagreement among models, and reducing the disagreement until the population reengage. Although both these methods have been shown to be empirically useful, Ficici [64] presents a useful theoretical framework for determining under which conditions monotonically increasing performance can be ensured in any coevolutionary system which attains a monotonically increasing knowledge of the search space.

In our algorithm, the evolution of a better model in the estimation phase allows the exploration phase to evolve a better test. In other words, if little is known about the target system, tests must be suggested at random. However, if something is known about the system, tests can be evolved that cause the system to exhibit some behavior it has not shown before, generating more information about the system. Conversely, the best previously evolved model may fail to replicate a new input/output data pair obtained from the system, thus causing new selection pressure to produce a new model that explains all previous input/output data pairs, as well as the new one.

The power of our algorithm is threefold. First, by evolving intelligent tests, it is possible to reduce the amount of testing required on the target system. Second, the algorithm is problem domain independent: the outline of the algorithm given above does not presuppose any particularities about the hidden system or the type of experiment to be performed on it. Third, the algorithm produces an explicit model of the hidden system. Fig. 2(b)–(e) sketches the application of the algorithm to the four problems described in the next four sections.

### B. Algorithm Outline

Six steps must be followed to apply the algorithm to a given problem, given in Fig. 3.

*1) Characterization of the Target System*: This involves defining the target system itself, specifying what aspects of the target system are known and which aspects must be inferred, and establishing representations for the space of models, inputs, and outputs. Variation operators need to be defined to search the space of models and space of inputs (tests). A similarity metric comparing two models needs to be established to assess convergence. A similarity metric comparing output also needs to be defined in order to quantify disagreement in model predictions.

Defining representations for inputs and outputs is usually a simple matter, as they are typically vectors or matrices which indicate the values of variables to be fed into the system or obtained as output. Similarity metrics for outputs are often some form of normalized error function, but occasionally more sophisticated metrics may be required as shown in some of our applications described later. The choices of representation, variation and comparison of systems usually involve domain-specific considerations.

*2) Initialization*: Initial populations of models and tests need to be created. In the absence of any prior information, these models and tests may be set at random or left blank; if some prior knowledge exists, it may be used to bias the initial populations. In all four of the applications reported in this paper, the initial populations are seeded with random models and tests.

*3) Exploration Phase*: Useful tests (inputs) are evolved with their fitness proportional to their ability to create *disagreement* among the successful candidate models. Since successful models are already compatible with all prior input/output sets, creating disagreement among their predictions focuses the tests on targeting any remaining uncertainties in the model. This is the approach taken in active learning methods [16]. Note, however, that disagreement among models can only be measured insofar as candidate models are different; in absence of sufficient model diversity, it may be necessary to seek a diversity of outputs compared with tests performed on the target system in previous cycles. Either way, we elicit some previously unobserved behavior from the model, and by extension, from the target system itself.

If the overall objective of the entire process is not just to infer a system but also to make it behave in a particular way, then the fitness of inputs needs to also capture a measure of its ability to elicit the desired output. In this case, a form of multiobjective search may be necessary (e.g., by alternating, weighting, or Pareto-selecting tests), though often these objectives coincide. Pareto optimization has already been investigated in the context of coevolutionary algorithms [22], and may be useful in evolving tests that extract desirable behavior *and* information about internal structure from the target system.

We have found that in order to facilitate evolution of new tests, it is often useful to evolve tests from scratch, rather than seed them with tests from the previous cycles. Once successful tests have been found, the best test is carried out on the target system and the output measured. The input/output set is recorded with all previous input/output sets.

*4) Estimation phase*: Useful candidate models are evolved with their fitness proportional to their compatibility with *all* input/output sets collected from the target system so far. Assuming the target system is consistent, all input/output sets are equally important and should be equally weighted.

When formulating a compatibility error for a specific application of the algorithm, the error often takes the form

$$m = \frac{\sum_{i=1}^{n} |t_i - m_i|}{n}$$

where $n$ indicates the number of experiments performed so far on the target system, $t_i$ is the output obtained from the target system during the $i$th test, and $m_i$ is the output obtained from

TABLE I
OVERVIEW OF THE FOUR APPLICATIONS

| | Grammar Induction | Gene Network Inference | Evolutionary Robotics | Damage Recovery |
|---|---|---|---|---|
| **Hidden Target System** | Finite State Machine | Genetic regulatory network | Robot | Damaged robot |
| **Termination Criterion** | Perfect model or $3 \times 10^8$ model evaluations | $1 \times 10^7$ model evaluations | 20 Iterations | 5 Iterations |
| **The Exploration Phase** | | | | |
| **Structure to Evolve** | Sentence | Initial protein concentrations | NN controller | NN controller |
| **Behavior of System** | Classification of the sentence | Changing concentrations | Sensor-based movement | Sensor-based movement |
| **Output of System** | Binary classification | New concentrations | Sensor time series | Sensor time series |
| **Quality Metric** | Greatest classification variation returned by the current best set of hypotheses. | Least number of zero or saturated concentrations. | Distance traveled over a fixed time period. | Distance traveled over a fixed time period. |
| **Population Size** | 2 populations of 100 genomes each | 200 | 100 | 200 |
| **Generations / Iteration** | 50 | 30 | 30 | 40 |
| **The Estimation Phase** | | | | |
| **Fixed Structures** | Number of states, alphabet size | Number of genes | Default simulator | Default simulator |
| **Structure to Evolve** | Transition function, accept states | Regulatory connections | Simulator modifications | Simulator modifications |
| **Resulting Hypothesis** | Finite State Machine | Model of the regulatory network | Simulation of the robot | Simulation of the damaged robot |
| **Quality metric** | Differences between the hypothesis' classifications and the target system's resulting classifications. | Differences between the hypothesis' resulting concentrations and the target system's resulting concentrations. | Differences between the simulated robot's sensor time series and the target robot's sensor time series. | Differences between the simulated robot's sensor time series and the target robot's sensor time series. |
| **Population Size** | 200 | 1000 | 100 | 200 |
| **Generations / Iteration** | 50 | 30 | 30 | 40 |

the model for the same test. An encoded model that correctly describes the target system will produce the same output as the target system for all tests, and its error will be low. We call this the *subjective quality* of a model, since it is estimated based only on known, possibly biased test data; the *absolute quality* of a model is not known to the algorithm, but the objective of the exploration phase is to create a suite of tests that make the subjective quality approximate the absolute quality. Note that diversity maintenance is important since assessment of tests is based on creating disagreement among models, and this is only possible insofar as models are different.

In some cases, models may fail to sufficiently explain the most recent test; in other words, the error $m$ of the best model at the end of the current pass through the estimation phase will be higher than $m$ of the best model from the previous pass. This can indicate the beginning of disengagement, which is one of the three pathologies that coevolutionary algorithms may experience [15], [33], [54], [58], [62]. Disengagement occurs when one population poses too much of a challenge to the other population; the dominated population then loses its fitness gradient, because all individuals perform equally badly against the individuals from the dominating population. In recent papers [6], [63], we have proposed several mechanisms for combating disengagement. The most useful mechanism has proven to be the test bank, in which difficult tests are removed from the test suite and only returned to the test suite when models become accurate enough to explain them. However, in the four applications presented here, disengagement was not detected, so these mechanisms are not used.

*5) Termination*: There are four criteria for terminating the algorithm, either 1) a sufficiently accurate model of the target system has been obtained; 2) an evolved test has caused the target system to exhibit the desired behavior; 3) a maximum number of target or model evaluations have been performed; or 4) the algorithm failed. The algorithm fails when the estimation phase fails to find any model that is compatible with all observed data, or when the exploration phase fails to find a test that causes different models to disagree. In the former case, this may indicate that the search space or variation operators do not span the target system, or that the target system is behaving inconsistently. In the latter case, this may indicate that there is some unobservable aspect of the target system that the test space cannot elucidate. In either of these cases, the representation, operators, or similarity metrics need to be reconsidered. In practice, it may be difficult or impossible to measure the actual accuracy of a model: some external kind of validation (see below) may then be required. However, even if validation is not possible in a practical situation, the algorithm can still be run for a fixed budget of physical trials, depending on the expense of performing a single physical trial.

For the grammar induction application, the algorithm is terminated if either criteria 1) or 2) is met: either a perfect model is discovered, or $3 \times 10^8$ model evaluations have been performed. For the gene network inference application, criterion 3) is used: the algorithm terminates when $1 \times 10^7$ model evaluations have been performed. For the final two robot applications, the algorithm terminates when a fixed number of target evaluations have been performed [criterion 3)].

*6) Validation*: In inference applications, a cross validation step helps assess the significance of the resulting successful model. Cross validation is achieved by performing a previously unseen test on the target system and comparing its outputs to the prediction proposed by the model. If the validation step is unsuccessful, the new input/output set is added to all previous input/output sets, and the algorithm resumes at the estimation phase.

These six steps are applied, in turn, to each of the four problems described in this paper. An overview of the applications is given in Table I. In the next section, we describe the application of the algorithm to the problem of grammar induction.

## III. Application 1: Grammar Induction

Grammar induction is a special case of the larger problem domain of inductive learning [5], which aims to construct systems based on sets of positive and negative classifications (for an overview, see [50]). In grammar induction, the system is a finite-state machine, or FSM, that takes strings of symbols as input, and produces a binary output, indicating whether that string, or sentence, is a part of the FSM's encoded language. Grammar induction is an attempt to model the internal structure of the hidden FSM, based only on pairs of sentences and classifications. The most successful grammar induction algorithms produced so far are heuristic in nature (see [14] for an overview), and there have been several attempts to evolve FSMs from sample data [25], [45], [46], but all approaches so far assume that a representative sample of sentences have already been classified by the target FSM. In other words they follow the batch-oriented approach to system identification [see Fig. 1(a)]: a pregenerated set of input/output data is fed into some induction algorithm, and a model FSM is produced.

Moreover, the pregenerated set of sentences in these approaches are usually assumed to be balanced: that is, there is an equal number of positively and negatively classified sentences. However, if an FSM is unbalanced (the FSM classifies a majority of sentences either positively or negatively), then collecting a balanced set of classified data for induction may be prohibitively expensive: a large number of sentences would have to be classified before a sufficient number of sentences attaining the minority classification could be collected. The application of our algorithm to the problem of grammar induction does not presuppose a set of input/output data, but rather uses an evolved set of models of the hidden FSM to generate a new sentence which, when fed into the hidden FSM, will produce a new data point that will help improve the model. It will be shown in this section how the estimation-exploration algorithm dramatically reduces the number of sentence classifications required by the target FSM, compared with a batch approach to grammar induction.

A deterministic finite automaton, or DFA, is a type of FSM that can be represented using the five-tuple $(n, \Sigma, T, s, F)$, where $n$ is the number of states, $\Sigma$ is the alphabet of the encoded language, $T$ is a transition function, $s$ is the start state, and $F$ is a set of final states. Then, given some sentence made up of $i$ symbols taken from the alphabet $\Sigma$, and beginning at the start state $s$, the first symbol is extracted from the sentence, and based on that symbol the sentence transitions to a new state as indicated by $T$. The next symbol is then extracted from the sentence, and based on $T$ the sentence transitions to a new state. This process is continued until all symbols in the sentence have been exhausted. If the last state visited is a member of $F$, then the sentence receives a positive classification (the sentence belongs to the language); otherwise, a negative classification is assigned (the sentence does not belong to the language). Given this formulation, we can now apply the estimation-exploration algorithm to this problem.

*1) Characterization of the Target System*: We assume that the target system is a DFA, and that we know the number of states, which is given as $n$. For simplicity, we here choose a binary alphabet. We then can represent $T$ as a $2 \times S$ matrix with values in $[0, n-1]$, where each column in $T$ represents a state, and the values in a given column indicate which new state to transition to, given the current symbol of a sentence. Finally, we can represent the set of final states $F$ using a binary string of length $n$: if $f_i = 1$, then state $i$ is a final state; otherwise, it is not. For simplicity, we assume that the first column of $T$ represents the start state. For the example three-state DFA shown in Fig. 4, the sentence 010 receives a positive classification, but the second sentence 011 receives a negative classification.

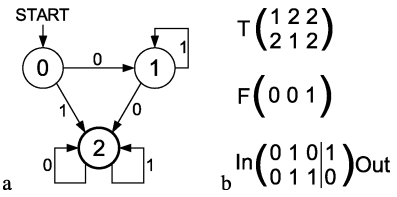


Fig. 4. Example of deterministic finite automaton. The DFA is composed of three states ($n = 3$), assumes a binary alphabet, and the third state is a final state. The first of the two sentences receives a positive classification because it ends at state 2; the second sentence receives a negative classification because it ends at state 1.

A simple grammar induction problem is then to model the hidden DFA by finding a $T'$ and an $F'$ such that for all sentences, the model produces the same classifications as those produced by some target DFA. Therefore, from among $(n, \Sigma, T, s, F)$, $n$, $\Sigma$, and $s$ are assumed to be known, but $T$ and $F$ are assumed to be unknown. Note that a model with $T' \neq T$ and/or $F' \neq F$ may still reproduce all of the same classifications as the target DFA, if it accurately captures the classification structure of the target DFA using a different $T$ and $F$, or if only a few input/output pairs have been tested.

In the estimation phase, each genome represents a candidate model of the hidden DFA. Since the number of states $n$ is known, each genome encodes a $2 \times n$ integer matrix $T$ with values in $[0, n-1]$ and a binary vector $F$ of length $n$. If $i$ sentences have already been classified by the hidden DFA, then the quality of a candidate model—its *subjective error*—is given by

$$m_{\mathrm{DFA}} = \frac{\sum_{j=1}^{i} |t_j - m_j|}{i} \tag{1}$$

where $t_j$ is the binary classification generated for the $j$th sentence by the hidden target DFA, and $m_j$ is the classification generated by the model DFA for the same sentence. Then, a genome that obtains $m_{\mathrm{DFA}} = 0$ perfectly reproduces all of the classifications produced by the hidden DFA so far, and genomes with higher values of $m_{\mathrm{DFA}}$ reproduce less of the correct classifications.

The space of inputs (tests) that can be applied to any given DFA is the space of binary sentences. For simplicity, we only evolve sentences with length $n$, the number of states. Sentences of variable length could also be used. Thus, each genome in the exploration phase is a binary string of length $n$.

The quality of a sentence is determined to be the ability of the given sentence to cause the greatest variation in classifications produced by the algorithm's current best set of candidate

models. In order to evolve more than one accurate model during the estimation phase, variation must be maintained in the model population until the estimation phase terminates. The simplest way to ensure this is to evolve $k$ isolated subpopulations, so that each subpopulation will evolve an accurate and unique model. So for this application, the population of candidate models is partitioned into $k$ subpopulations during each pass through the estimation phase. At the end of each pass, there are then $k$ candidate models that explain the hidden DFA: the best model from each subpopulation obtained during that pass through the estimation phase. The quality metric of a candidate sentence (experiment) is then given by

$$e_{\mathrm{DFA}} = 2\left(\left|0.5 - \frac{\sum_{j=1}^{k} c_j}{k}\right|\right) \tag{2}$$

where $c_j$ is the classification of the candidate sentence by candidate model $j$. Sentences that do not differentiate between the candidate models—all models produce the same classification—obtain $e_{\mathrm{DFA}} = 1$ (poorest quality); sentences that produce the maximum classification variance obtain $e_{\mathrm{DFA}} = 0$ (best quality). When a sentence is evolved that induces high classification variance, and that sentence is classified by the hidden DFA, then the resulting classification will usually lend support to $k/2$ candidate models during the next pass through the estimation phase, and provide evidence against the remaining half. This idea is borrowed from the coevolutionary literature, in which it has been shown that the fitness of a test should be proportional to its ability to induce a learning gradient in the competing population of learners [12], [22], [33] (in our framework, a learner is a model DFA). The value of causing disagreement between models is discussed in more detail in Section VII.

*2) Initialization*: Since we do not assume an initial model of a given hidden DFA, the algorithm initially generates a random binary sentence, which is classified by the hidden DFA. This single sentence/classification pair is then fed into the estimation phase in order to generate an initial model.

*3) Exploration Phase*: The exploration phase begins each pass with a population of 200 randomly generated genomes, and the population evolves for 50 generations. At the end of each generation, when all of the genomes have been evaluated, 150 pairs of genomes are selected randomly and sequentially (some genomes may be selected more than once), and for each pair the genome with a lower value of $e_{\mathrm{DFA}}$ is copied and replaces the genome with a higher value of $e_{\mathrm{DFA}}$.

The copied genome is then mutated: one randomly selected element from the candidate sentence is chosen with a uniform distribution,[1] and is replaced with a new random value. Crossover is currently not used, but may be implemented in future improvements to the algorithm. A total of 150 replacements are performed after each generation. Note that a given genome may undergo more than one mutation if it is selected, copied, mutated, and then selected again. Once a set of selections, replacements, and mutations have occurred, all of the new genomes in the population are evaluated.

[1]As are all the other random values used in the experiments described in this paper.

When all 50 generations have been evaluated, the sentence with the lowest value of $e_{\mathrm{DFA}}$ is output and supplied to the target system.

*4) Estimation Phase*: The estimation phase, like the exploration phase, evolves a population of 200 initially random genomes for 50 generations. However, the population of the estimation phase is partitioned into two genetically isolated subpopulations with 100 genomes each.

Also, like the exploration phase, when all genomes in each subpopulation have been evaluated, selection and mutation occurs. In each subpopulation, 150 pairs of genomes are selected at random, and for each pair the genome with the lower value of $m_{\mathrm{DFA}}$ is copied and replaces the genome with the higher value of $m_{\mathrm{DFA}}$. The copied genome then undergoes mutation: first either the encoded $T'$ or $F'$ is selected with equal probability; then a value within the selected matrix or vector is randomly selected and replaced with a new random value. Crossover is also not implemented in this phase. The newly created genomes are evaluated, and evolution continues. When this phase terminates, two competing candidate models are output to the exploration phase: the model with the lowest value of $m_{\mathrm{DFA}}$ from each subpopulation.

Unlike the exploration phase, which begins each pass with a population of random genomes, on the second and subsequent passes through the estimation phase each subpopulation of random genomes is seeded with the best genome it evolved during the previous pass. This allows the algorithm to continue improving its previous best candidate models, and for the results of the newly evolved test to lend support or provide evidence against these models.

*5) Termination*: After each pass through the estimation phase, the algorithm is terminated if either of two criteria are met: the most fit model achieves an absolute error of 0, or $3 \times 10^8$ model evaluations have been performed. In this paper, hidden DFAs with ten states $(n = 10)$ were inferred. If a model DFA classifies all possible sentences correctly, then it obtains an absolute error of 0. The set of all possible sentences in this context is considered to be all binary strings with lengths equal to the number of states in the target DFA. So for the inference of DFAs with ten states, there are $2^{10} = 1024$ possible sentences. The number of model evaluations is tallied as followed: during the $i$th pass through the estimation phase, each model requires $i$ evaluations in order to determine its fitness; during the exploration phase, each test requires $k$ model evaluations in order to determine its fitness, where $k$ is the number of model subpopulations.

*6) Validation*: The calculation of the absolute error of a model DFA is considered to be validation, because it uses sentences unseen during model evolution. Thus, from the viewpoint of validation, the algorithm can be said to have been successful if the absolute error of the model DFA output after termination is below some sufficiently small error threshold.

### A. Grammar Induction Results and Discussion

The second column in Table I provides a summary of the application of the estimation-exploration algorithm to this particular instance of grammar induction. The proposed algorithm was run against a number of target DFAs with ten states, along with two control algorithms. The first control algorithm was a

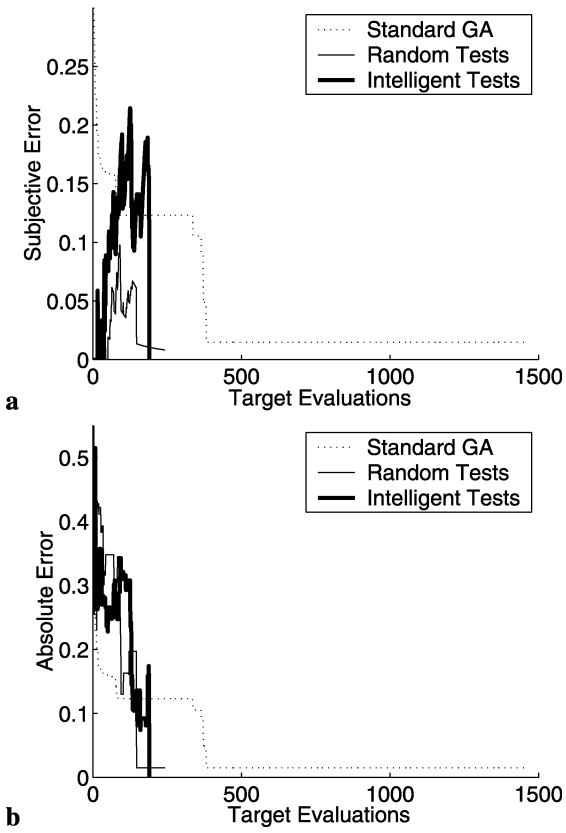


Fig. 5. Model accuracy results from a single run. (a) Subjective errors of the most fit model DFAs produced by the two control algorithms (a standard GA and random testing) and the proposed algorithm (intelligent testing). (b) The absolute errors for the sample model DFAs. The most fit model DFA at the end of each generation, for all three algorithms, was extracted and its subjective and absolute error calculated.

standard genetic algorithm: all 1024 binary sentences of length 10 were first classified by the target DFA, and then a random population of 200 model DFAs were evolved until either a perfect model is discovered (one that achieves an absolute error of 0) or $3 \times 10^8$ model evaluations are performed. Genetic encoding, selection and mutation are the same as described in the previous section. This control algorithm, thus, follows the batch approach to system identification depicted in Fig. 1(a). The second control algorithm is identical to the estimation-exploration algorithm, except that the exploration phase is disabled: when the estimation phase terminates, a randomly generated binary sentence of length 10 is sent to the target DFA for classification.

Thirty target DFAs with ten states each were randomly generated for inference. Each of the three algorithms then performed 40 independent runs against each of the 30 target DFAs, requiring a total of $3 \times 40 \times 30 = 3600$ independent runs.

Fig. 5(a) reports the subjective errors [see (1)] of the best model DFA during the course of each algorithm's execution against the same target DFA. For the standard genetic algorithm (GA), the subjective error of the most fit model DFA after each generation was recorded; for the latter two algorithms, the subjective error of the most fit model DFA after each generation of the estimation phase terminates was recorded. Note that because the standard GA uses all possible sentences for model evaluation, subjective error is equal to absolute error for the model

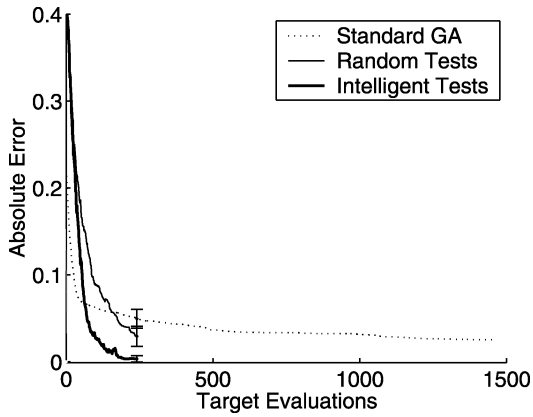


Fig. 6. Mean performance of the three algorithms against a single target DFA. Forty independent runs were performed using each algorithm against the same target DFA composed of ten states. Absolute errors were computed for the best model DFAs produced at the end of each generation; the errors are averaged over the 40 runs.

DFAs in that algorithm. Fig. 5(b) reports the absolute errors for the same model DFAs.

As can be seen, only the estimation-exploration algorithm achieves a perfect model, after about 190 target evaluations; the other two control algorithms terminate without ever finding a perfect model. Also, Fig. 5(a) shows that the standard GA begins with a very high subjective error, while the latter two algorithms begin with zero subjective errors, which then climb and later reapproach zero. This is because it is much easier for the model DFAs to correctly classify the few sentences labeled by the target DFA than later on, when many sentences have been labeled. Fig. 5(b) shows that the standard GA exhibits the fastest drop in absolute error, which is to be expected because early in the algorithm the standard GA has much more observed data available for model evolution than the latter two algorithms do. Finally, Fig. 5(a) shows that intelligent testing induces much greater subjective errors in the model DFAs compared with random testing. This is indirect evidence that the proposed algorithm is discovering informative tests: tests that expose unexplained aspects of the target system.

Fig. 6 reports the mean absolute errors of the most fit model DFAs produced by all three algorithms for one of the target DFAs. The absolute errors for the model DFAs were averaged over the 40 independent runs. The error bars in the figure (as well as the error bars in all subsequent plots in this paper) indicate standard error with a 95% confidence interval. As in the previous plot, the average accuracies of the model DFAs produced by the standard GA early on are much better than the latter two algorithms, due to the scarcity of observed data available to the iterative methods. However, the proposed algorithm attains a significantly better mean performance than the other two algorithms after less than 100 target evaluations have been performed.

Fig. 7 reports the comparative performance of the three algorithms against the 30 target DFAs. Fig. 7(a) reports the mean number of model evaluations required by each algorithm to produce a perfect model: a model DFA with an absolute error of 0. Note that in most cases, the proposed algorithm is able to consistently discover a perfect model with fewer model evaluations

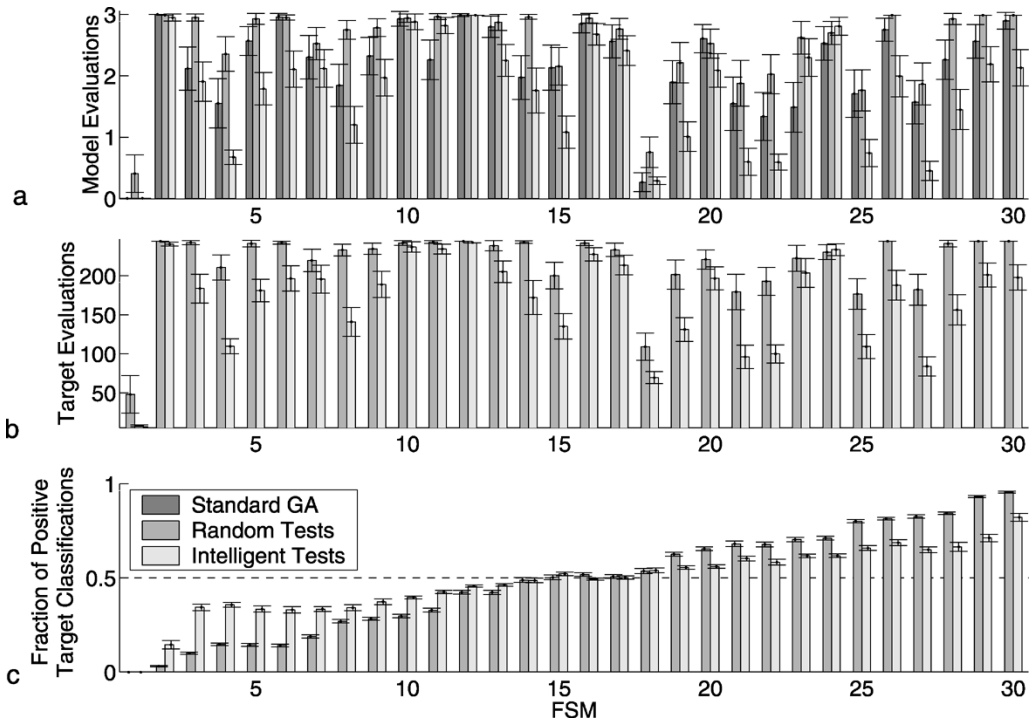


Fig. 7. Comparison of algorithm performance against 30 random target DFAs. (a) The mean number of model evaluations required until a perfect model is found. (b) The mean number of target evaluations required until a perfect model is found. (c) The mean fraction of positive classifications acquired by the sentences proposed to the target DFA.

than the standard GA, even though the standard GA has access to much more information about the target system (compare the heights of the dark gray bars against the light gray bars). Conversely, the standard GA always outperforms iterative, random testing (compare the dark gray bars against the medium gray bars), indicating the performance gain of the estimation-exploration algorithm comes not from performing iterative inference, but from sending informative tests to the target DFA.

Fig. 7(b) compares the number of target trials required by the second control and proposed algorithms to find a perfect model. The standard GA is not shown in this plot because each standard GA requires exactly 1024 target trials before the algorithm commences. For most of the 30 target DFAs, intelligent testing requires significantly fewer target trials in order to discover a perfect model. Even with roughly 1/5 the number of target trials (about 200 trials compared with 1024 trials), the proposed algorithm can discover a perfect model more quickly (i.e. with fewer model evaluations) than the standard GA can.

Fig. 7(c) compares the fraction of positive classifications obtained from the target DFA for the tests proposed by the second control and proposed algorithms. Because the second control algorithm proposes random tests, the fraction of positive classifications obtained by this algorithm gives an approximation of the balance of the target DFA. The balance of a DFA is considered to be the probability that it will produce a positive classification given any arbitrary sentence; DFAs with probabilities of 0.5 are considered balanced, and those with probabilities far from 0.5 are considered imbalanced. As can be seen, the target DFAs are aligned in Fig. 7 in order of their balance. [The target DFAs in Fig. 7(a) and (b) are aligned with the ordering in Fig. 7(c); for example, target DFA 5 in Fig. 7(a) and (b) is the same target DFA 5 shown in Fig. 7(c)]. It can be seen that intelligent testing tends to consistently elicit more balanced classifications from very imbalanced target DFAs: in other words intelligent testing creates a sufficiently accurate model that more of the underrepresented class of sentences can be discovered and proposed to the target DFA (indicated by the relatively taller light gray bars on the left side of Fig. 7 and the relatively shorter light gray bars on the right side.) This indicates that evolving tests that cause disagreement among models leads to the uncovering of less observable components of the target system: in this applications, this involves traversal to final states that produce minority classifications.

However, it can be seen that intelligent testing also outperforms random testing on many balanced DFAs (note the significant performance advantages of the proposed algorithm on the horizontally centered target DFAs). This indicates that a target DFA may have less observable components even if it is balanced: for example, some states may be visited by random sentences much more often than other states if the state transition graph has regions of dense connectivity and other regions have sparse connectivity.

In this section, we have described the application of the estimation-exploration algorithm to the problem of grammar induction.We have shown that it is worthwhile to evolve experiments (sentences) for this problem, and that the quality of a test lies in its ability to distinguish between two candidate models. Furthermore, we have shown that intelligent testing can discover a perfect model using fewer model and target tests than either a similar iterative algorithm that employs random testing, as well as a standard genetic algorithm that assumes a large amount of preclassified sample data. Finally, we have shown that intelligent

testing is most useful when the hidden DFA has low observability. In the next section, we describe the application of the algorithm to the problem of gene network inference.

## IV. Application 2: Gene Network Inference

Systems biology [39] is concerned with the synthesis of large amounts of biological detail in order to infer the structure of complex structures with many interacting parts. In this way, systems biology can be viewed as another example of system identification. The field of gene network inference is a rapidly burgeoning subfield in systems biology [39], and is concerned with inferring genetic regulatory networks based on the results of a set of tests performed on the network in question.

Many different models of the underlying genetic network have been used, usually classified based on the amount of biological detail inherent in the model (see [17] and [18] for an overview). In addition to the type of model, several methods have been used to infer genetic networks, including clustering algorithms (see [18] for an overview), correlation metrics [3], linear algebra [13], simulated annealing [48], and genetic algorithms [31], [38].

A number of input and output data pairs are required in order to obtain enough information about the target network to infer its structure correctly. As pointed out in [18], it is desirable to reduce the number of input/output pairs required as much as possible, so that a minimum of experiments have to be conducted. Also, the *type* of experiment required should be as cheap as possible in terms of experimental difficulty and accuracy of acquired output data. Iba and Mimura [31] showed that by using a multipopulation evolutionary algorithm to not only infer the hidden network, but also to propose additional experiments that would most help to refine the current best evolved network hypothesis. However, their model requires the experimenter to perform costly knockout or lesion experiments in order to supply the algorithm with an actual subset of the regulatory network.

Our approach does not require such invasive experiments, but rather assumes that the input data to a biological experiment is a set of chemical concentrations. These chemicals can be viewed as either initial concentrations of the gene products themselves, or some other media such as signalling proteins or glucose that externally affect the cell. This approach obviates the need for invasive testing, leading to a simpler experiment. The input is assumed to trigger gene regulation, which produces a set of output gene product concentrations over time, as obtained using microarray tools. We can now apply the estimation-exploration algorithm to infer the structure of a hidden gene network, given sets of initial chemical concentrations, and resulting product concentrations, as outlined in Fig. 2(c).

*1) Characterization of the Target System*: The target system is assumed to be a gene network in which the number of genes $n$ is known, but how one gene contributes to the regulation of another is assumed to be unknown.

The steps for applying the estimation-exploration algorithm to this particular problem are only summarized below: please refer to [9] for more details regarding this application.

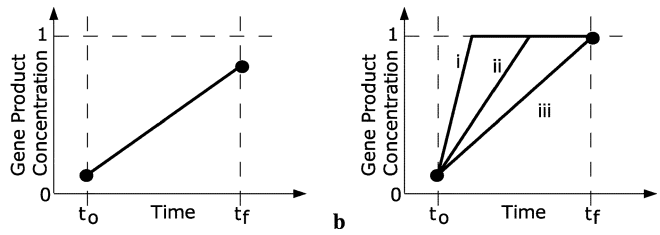


Fig. 8. Observability of gene regulation. In the gene network model used here, it is assumed that gene product concentration is measured after some time period has elapsed ($t_f$). (a) If the final concentration of some gene product is intermediate, there is only one possible rate for the gene product concentration change between $t_0$ and $t_f$ (we assume in this paper that the change during this time interval is linear). (b) If the final which attempts to evolve an accurate model using a set of observed gene product concentration is either zero or completely saturated, there are several hypotheses for how this gene is regulated (three possible hypotheses are indicated by trajectories *i*, *ii*, and *iii*). Therefore, the regulation of the gene in (b) is less observable than the gene in (a).

Both the target gene network and models of it are represented using an $n \times n$ matrix $\mathbf{R}$ with entries $r_{ij}$ in $[-1, 1]$. The new concentration of gene $j$ after some time period is then given by

$$g_j^{(t+1)} = \min\left(\max\left(0, g_j^{(t)} + \sum_{i=1}^{n} r_{ij} g_i^{(t)}\right), 1\right). \quad (3)$$

Since the $g_j$ variables indicate concentration, the $min$ and $max$ functions bound the value between 0 (for no concentration) and 1 (for concentration saturation). Genomes in the estimation phase are then represented as $n \times n$ matrices. The *subjective error* of a given network model $\mathbf{R}'$ is simply the mean squared error between the concentrations output by the hidden network and the candidate model so far. For this application, absolute error is simply the distance of a model gene network from the actual gene network

$$\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\left|r_{ij} - r'_{ij}\right|}{n^2} \quad (4)$$

where $r_{ij}$ represents gene $i$'s regulation of gene $j$ in the target system $\mathbf{R}$, and $r'_{ij}$ represents gene $i$'s regulation of gene $j$ in the candidate model $\mathbf{R}'$.

The test for a hidden gene network is determined as a vector of $n$ floating-point values in [0, 1] that represent initial gene product concentrations; thus, genomes in the exploration phase are vectors of $n$ floating-point values. The output from the target system or model is a vector of $n$ floating-point values indicating eventual gene product concentrations after some time period has elapsed. Unlike in the grammar induction application in which test quality is given by its ability to cause the candidate models to disagree, the test quality in this application is to increase the observability of the system; in other words, to minimize the number of extremal concentrations, which indicate less about the internal structure of the network than intermediate concentrations. Fig. 8 shows why final gene product concentrations that are either zero or saturated reveal less information about the underlying network than gene product concentrations that fall between these extremes.

*2) Initialization*: The algorithm begins by generating some initial random input vector $\mathbf{g}^{(t)}$. This is then applied against the hidden target network $\mathbf{R}$. The resulting $\mathbf{g}^{(t)}/\mathbf{g}^{(t+1)}$ input/output

vector pair is fed into the estimation phase, and the algorithm commences.

*3) Exploration Phase*: The exploration phase begins with a population of 1000 random genomes, and the population is evolved against the single candidate model $\mathbf{R}'$ output by the estimation phase. Selection and mutation then proceed similarly to the previous application, with the mutation operators altered to handle the floating-point valued genomes. The population is evolved for 30 generations, at which point the genome with the best quality is output to the target network $\mathbf{R}$.

*4) Estimation Phase*: Evolutionary search in the estimation phase begins with a population of 200 random genomes, each encoding some $\mathbf{R}'$. The subjective error of each genome is computed, and 150 replacements are performed: for each genome pair, the genome with lower subjective error replaces the genome with higher subjective error. Mutation but not crossover is applied to the copied genomes. The population is evolved for 30 generations, at which point the genome with the lowest subjective error is passed to the exploration phase. During the second and subsequent passes through this phase, the initial random population is seeded with the best model evolved during the previous pass.

*5) Termination*: As in the grammar induction, the algorithm was run until either a perfect model was produced, or a maximum number of model evaluations (for this application $1\times10^7$) had been performed. Because the model in this application consists of continuous values (indicated by the strength and kind of gene $i$'s regulation of gene $j$), there is a vanishingly small probability that a perfect model—one that achieves an absolute error of 0—will be discovered, so all runs for this application terminate when $1 \times 10^7$ model evaluations have been performed.

*6) Validation*: Validation of a candidate model is interpreted as its absolute error.

### A. Gene Network Inference Results and Discussion

In order to test the algorithm against this task, two control algorithms were formulated as in the previous application: the first control algorithm acts as a standard GA which attempts to evolve an accurate model using a set of observed data from the target system; the second control algorithm is identical to the estimation-exploration algorithm, except that the exploration phase is disabled: each pass through the exploration simply outputs a random set of gene product concentrations.

Because the tests in this application are composed of continuous values (instead of binary values in the grammar induction application), there are an infinite number of possible tests, so we can not provide the standard GA with an exhaustive set of sample data already processed by the target system. Rather, we generate a set of 100 random $1 \times n$ vectors of initial gene product concentrations, with the values of each vector sampled from [0, 1]. By choosing a maximal number of model evaluations $(1 \times 10^7)$, and using a population size of 200 for all three algorithms, and running both the estimation and exploration phases for 30 generations, the second control algorithms performs a total of 56 target evaluations, and the proposed algorithm performs 55 target evaluations (because the proposed algorithm incurs slightly more model evaluations per iteration than the second algorithm due to the active exploration phase, it only gets through 55 iterations before the $1 \times 10^7$ model evaluations have been exhausted). The standard GA was provided with 100 sets of sample data so that it requires almost twice as many target evaluations as the other two algorithms.

All three algorithms were run against a total of 80 randomly generated target gene networks. The networks varied in the number of genes in the network $(n)$, and the network's connectivity $(k)$. For each network, each gene was regulated by a total of $k$ other randomly chosen genes (with the possibility of itself being one of the genes); the other $n - k$ genes do not directly regulate that gene $(r_{ij} = 0)$. The first 20 target networks were generated with $n = 5$ and $k = 2$; the following 20 with $n = 5$ and $k = 5$; the following 20 with $n = 10$ and $k = 2$; and last 20 networks with $n = 10$ and $k = 10$. For each target network, each algorithm performed a total of 30 independent runs in an attempt to infer the hidden regulatory network.

Fig. 9 reports the average performance of the three algorithms against these 80 target networks. Because a perfect model is never found, each algorithm performs $1 \times 10^7$ model evaluations; the standard GA performs 100 target trials; the iterative algorithm with random testing performs 56 target trials; and the estimation-exploration algorithm performs 55 target trials. The mean performance of each algorithm is calculated to be the mean of the absolute errors of the 30 best model gene networks output at the termination of the 30 independent runs.

First, it can be noted that for all classes of target network, the standard GA performs significantly worse than the other two algorithms. Individual runs (data not shown) indicate that the standard GA tends to converge on very poor local minima after only a few generations, and never make any subsequent progress. Surprisingly, the iterative algorithm employing random testing performs significantly better than the standard GA, using half as many target trials. This indicates that the iterative approach for this problem domain provides an evolutionary advantage: each successive pass through the estimation phase presents the evolving models with a fitness landscape different from the previous pass, and that this difference is induced by the new test obtained from the target system.

Second, the proposed algorithm only outperforms random testing on the largest and most densely interconnected gene networks ($n = 10$, $k = 10$). This occurs for two reasons. Because the tests are composed of continuous rather than binary or integer values, there is a very low probability that the same test will be proposed more than once during random testing, and a set of unique tests is more informative than a set of tests in which several tests appear more than once. Also, large, densely interconnected networks tend to lead to more extremal gene product concentrations than smaller, sparse gene networks: a gene regulated by many other genes will exhibit a large change in its gene product concentration, compared with a gene which is only regulated by a few genes. Thus, as in the previous application, our algorithm is most valuable for inferring large target systems with low observability.

This claim that extremal gene product concentrations are more likely in large, dense gene networks is supported by Fig. 10: 1000 random networks were generated using values of $n$ selected from [2, 30] with a uniform distribution, and values of $k$ selected from $[1, n' - 1]$, where $n'$ is an already

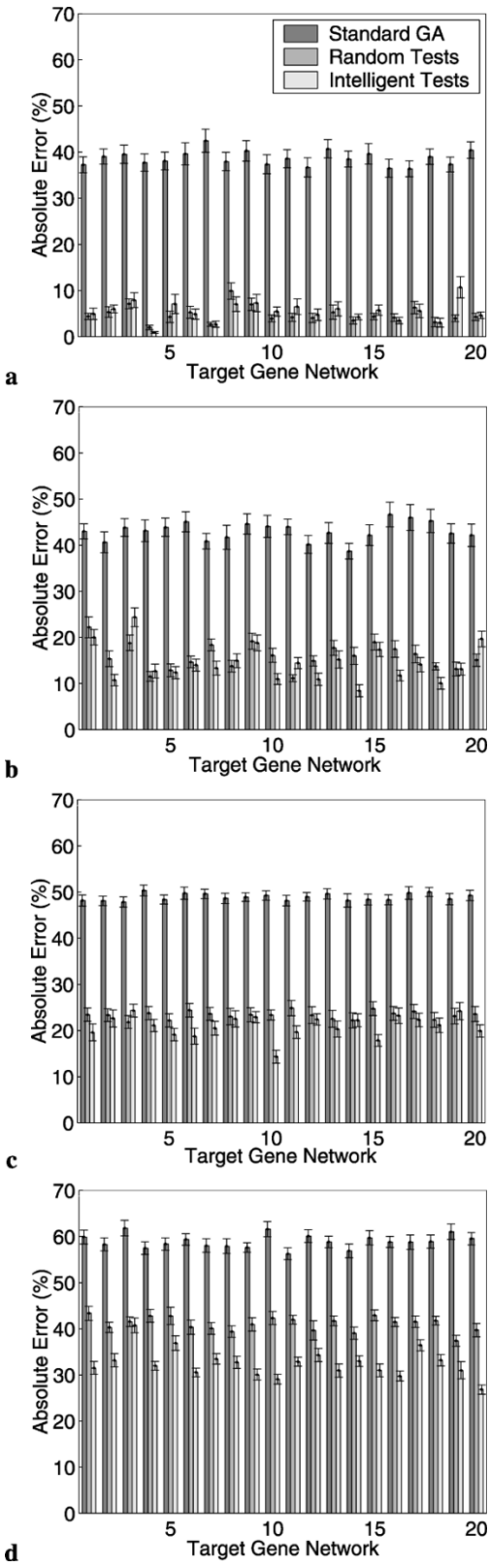


Fig. 9. Mean performances of the three algorithms against the four classes of target gene networks. (a) Mean performance of all three algorithms against the 20 target gene networks with $n = 5$ genes and $k = 2$ connectivity, (b) the 20 target networks with $n = 5$ and $k = 5$, (c) the 20 target networks with $n = 10$ and $k = 2$, and (d) the 20 target networks with $n = 10$ and $k = 10$.

randomly selected value for $n$. For each of the 1000 random networks, ten input vectors were randomly constructed, and

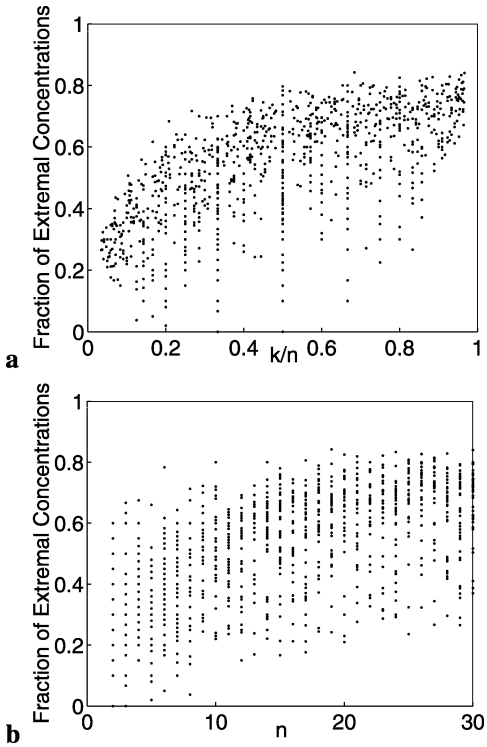


Fig. 10. Observability of various gene network types. 1000 gene networks with different values of $n$ and $k$ were generated randomly. For each network, 100 random input vectors were supplied, and the average fraction of resulting extremal output concentrations was calculated. (a) The relationship between the number of genes $(n)$ and the fraction of extremal values. (b) The relationship between connection density $(n/k)$ and the fraction of extremal values for the same 1000 networks.

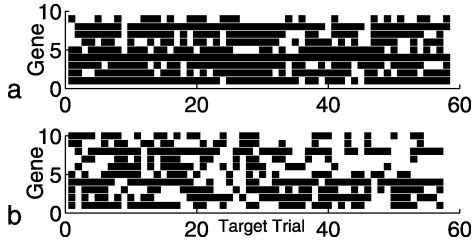


Fig. 11. Testing results from a sample run using random and intelligent testing. Each column indicates the resulting gene product concentrations obtained from a test proposed by either (a) the iterative algorithm using random testing or (b) the proposed algorithm. Black squares indicate extremal gene product concentrations and white areas indicate nonextremal concentrations.

the corresponding ten output vectors were calculated using (3). The average fraction of extremal output concentrations was computed for each network, and is plotted in Fig. 10(a) as a function of the number of genes in the network $(n)$, and in Fig. 10(b) as a function of connectivity $(k/n)$. Clearly, the fraction of noninformative output elements increases both with the number of genes, and with connectivity. Thus, it becomes increasingly valuable not only to evolve models, but also to evolve informative experiments for the inference of larger and more complex gene networks.

Fig. 11 substantiates this claim by showing the resulting gene product concentrations from tests proposed by a sample run of the iterative algorithm with random testing and the proposed algorithm. The sample runs were both drawn from the set performed against one of the target gene networks with $n = 10$ and

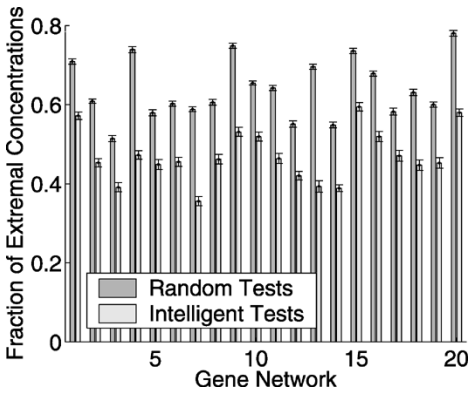


Fig. 12. Mean fraction of extremal gene product concentrations produced by random and intelligent testing. Results are reported for both algorithms against the 20 target gene networks with $n = 10$ and $k = 10$.

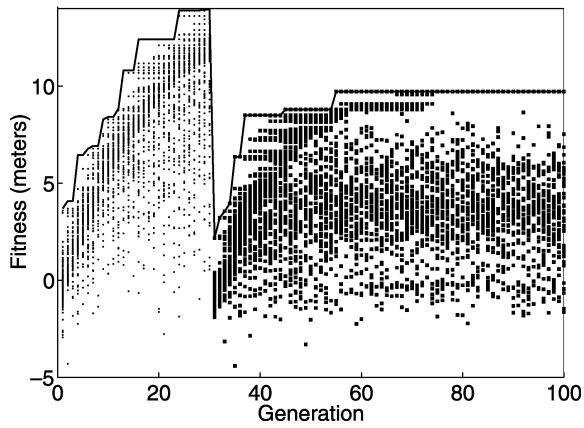


Fig. 13. Evolutionary progress of a traditional evolutionary robotics regime. Thirty generations of a genetic algorithm that evolves a neural network controller for a robot so that it performs a forward locomotion task (the robot and its controller are shown in Fig. 14). Fitness is the distance traveled by the robot. Each dot indicates a simulated robot evaluation. The best controller is then transferred to a target robot, which has some hidden morphological difference (one of the robot's lower legs has broken off). Evolution then continues on the target robot, using the same genetic algorithm, for another 70 generations. Each square indicates an evaluation on the target robot. The solid line indicates the best fitness achieved at each generation.

$k = 10$. As can be seen, the proposed algorithm tends to propose tests that produce many fewer extremal gene product concentrations [indicated by the fewer black squares in Fig. 11(b) compared with Fig. 11(a)].

Fig. 12 reports the mean fraction of extremal gene product concentrations produced by the tests proposed by the iterative algorithm with random testing and the estimation-exploration algorithm, while inferring the 20 target networks with $n = 10$ and $k = 10$. It is clear that for all 20 target networks, intelligent testing achieves significantly fewer extremal concentrations than random testing. This indicates that evolving tests that produce as many nonextremal gene product concentrations as possible allows for the evolution of more accurate tests when the target network is large and has low observability.

Interestingly, some of the genes in a given hidden network are less observable than others. For example the experimental results obtained by the proposed algorithm in Fig. 11(b) fail to consistently achieve nonextremal concentrations for the fourth gene (indicated by the long black bars for that gene over the course of the run), while it seems relatively easy to obtain nonextremal concentrations for the sixth and seventh gene (evidenced by the long white bars for those genes near the end of the run). This indicates that this gene is very strongly positively or negatively regulated and, thus, has low observability because it often reaches extremal concentrations. We plan to improve the quality metric for experiments in future work such that genes with low observability (as indicated by previous experiments) automatically come under increasing scrutiny.

Thus, the estimation-exploration algorithm has three benefits for gene network inference. First, it does not require invasive, expensive, slow and disruptive experiments such as knockout or lesion studies. Rather, the exploration phase carefully evolves a low-cost experiment (a change in the initial gene product concentrations) that yields a large amount of information about the target system. Second, the number of experiments performed on the target system is reduced, because each proposed experiment is carefully chosen. Finally, the automated evolution of informative experiments becomes increasingly valuable as the hidden networks become larger and more densely interconnected, since large, dense networks often produce information-poor output data in response to random testing.

The following two sections describe the application of our algorithm to two important problems in robotics research: automating behavior generation, and automating damage diagnosis and recovery.

## V. Application 3: Evolutionary Robotics

An evolutionary robotics experiment requires an evolutionary algorithm to optimize aspects of a simulated or physical robot in order to generate some desired behavior. Because it is difficult and time-consuming to perform the thousands of fitness evaluations required by an evolutionary algorithm on a physical robot, most or all of evolutionary robotics experiments are performed in simulation.

Evolution in simulation raises a major challenge: The transferal of evolved controllers from simulated to physical robots, or "crossing the reality gap" [32]. Because there are always differences between the simulated robot and its physical instantiation, controllers evolved in simulation do not always allow for the same behavior to arise in the target robot when transferred. Fig. 13 shows the evolutionary progress for an evolutionary robotics experiment. A neural network controller was evolved for a simulated legged robot for 30 generations, and then transferred into a different robot (the target robot, which in this case was also simulated) that is morphologically different from the simulated robot: the target robot is missing one of its lower legs. The transferal causes a large drop in fitness, and subsequent evolution, involving about 3000 evaluations on the target robot, only recovers about 70% functionality.

There are several approaches to the challenge of controller transferal, including adding noise to the simulated robot's sensors [32]; adding generic safety margins to the simulated objects comprising the physical system [26]; evolving directly on the physical system ([23], [47] and [59]); evolving first in simulation followed by further adaptation on the physical robot ([47], [51]); or implementing some neural plasticity that allows the

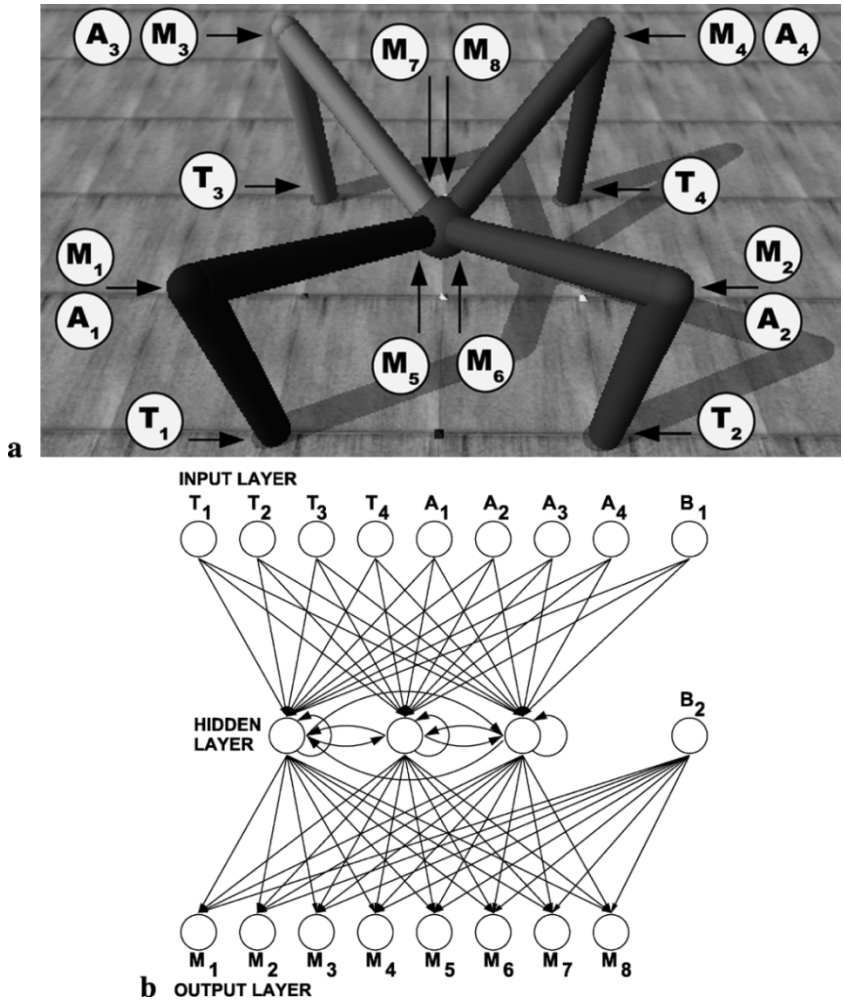


Fig. 14. Robot and its controller used for the evolutionary robotics experiments. (a) Quadrupedal robot. $T_i$ indicates touch sensors; $A_i$ indicates angle sensors; $M_i$ indicates motorized joints. (b) Neural network controller. Sensors are arranged on the input layer, and motors are arranged on the output layer. $B_i$ indicates bias neurons that output a value of 1 at each time step.

physical robot to adapt during its lifetime to novel environments ([19], [24], [60]).

Another approach that can be used in lieu of, or in addition to the above-mentioned approaches, is to treat the problem as a system identification task in which there are hidden differences between the target robot and the robot simulator, which must be automatically uncovered and included into the simulation. In this section, we document the application of our algorithm to this problem, such that the exploration phase evolves controllers for a sensor-driven target robot to make it behave, and the resulting behavior of the target robot is used to refine the robot simulator. The goal is to automatically refine the simulator sufficiently that controllers evolving in it cause the target robot to produce similar behavior to that seen in simulation. Thus, the target robot serves as the target system; the controller is the input vector that elicits behavior from the system; the resulting sensor time series are treated as the output from the system; and the robot simulator serves as the model of the target system. Fig. 2(d) outlines the algorithmic flow for this application.

Most evolutionary robotics experiments evolve controller parameters for a robot with a fixed controller topology and fixed morphology (examples include [23] and [53]), whether the evolution is performed in simulation or the real world. Other approaches have widened evolution's control over the design process by subjugating the controller topology and/or the robot's morphology to modification as well (e.g., [2], [7], [30], [41], [43], and [55]) with the aid of simulation. In this application, we evolve the simulator itself: this may involve virtual modifications to the simulated robot's body, its sensor/motor apparatus, its virtual environment, or the physical parameters of the simulation itself. The ability to evolve a simulated robot's morphology or its environment, in addition to its controller, has become much easier recently due to the advent and availability of physics-based simulators, which allow for faster than real-time, three-dimensional dynamic evaluation of different physical systems (see [21] for an overview). Next, we describe the preparatory steps for applying our algorithm to this problem.

*1) Characterization of the Target System*: The target system here is a quadrupedal robot with an articulated body, a set of sensors and motors, and a neural network controller that connects sensors to motors. In this work, the target robot is not a physical robot out in the world, but a separate, simulated robot which is identical to the default simulated robot except for some

unknown morphological differences. The task of the algorithm is to indirectly infer these differences, and modify the default simulator to reflect them accurately. In future work, we plan to apply our algorithm to a physical robot. The layout of the target robot is shown in Fig. 14(a), and the topology of its controller is shown in Fig. 14(b). The simulation environment, the simulated robot and its neural network controller are described in more detail in [8].

The experimenter must then choose those morphological or environmental characteristics that he thinks may differ between the target robot and the simulation and that will affect the successful transferal of behavior. Currently, this selection is done ad hoc, but it is intuitive: for example the color between the robots will not matter, but mass distribution and sensor behavior will most likely not be modeled accurately at the outset, and both affect behavior greatly.

For the work reported here, we induced a difference in mass distribution and sensor time lags between the target robot and the initial default robot simulator. For these initial experiments, we assume that the target robot only differs from the default simulation in those characteristics that we have chosen to place under evolutionary control. From similar experiments [8], we have found that in some cases even an approximate simulator that does not refine all the differing physical characteristics between the simulated and target robot allows for adequate transfer of behavior.

Thus, for this application the algorithm must automatically infer the differences in body part masses and sensor time lags only using sensor feedback from the target robot. Therefore, a candidate model is a set of 17 floating-point parameters (masses of the nine body parts and the time lags for the eight sensors), which are collected into genomes and evolved. The parameters are used to modify the default robot simulator, producing a candidate robot simulator.

The quality of a candidate simulator is given by the ability of the simulated robot to mimic the observed behavior of the target robot. More specifically, given $i$ previously evolved controllers tested on the target robot, the simulated robot should mimic as closely as possible the $i$ sets of sensor time series produced by the target robot. However, unlike the previous two applications, we can not directly compare model and target output to gauge model accuracy. Quantitatively comparing sensor data from two highly coupled, highly nonlinear machines like the robot used here is very difficult: slight differences between the two machines rapidly leads to uncorrelated signals. To address this, we have formulated a comparison metric called the *rolling mean metric*, which is described in detail in [8].

In this application, a test is a set of 68 floating-point values used to label the robot's neural network controller. Therefore, the genomes in the exploration phase are vectors of 68 floating-point values in $[-1, 1]$. The quality of a test is how far the robot, using the labeled controller, moves forward during 1000 time steps of the simulation. It is implicitly assumed that there are many possible gaits for the target robot and, therefore, different evolved controllers that make it move in various ways. Therefore, as long as separate passes through the exploration phase begin with different random populations of controllers, different useful controllers will be output to the target robot, and additional information will be extracted from the robot automatically. This obviates the need to explicitly include a term in the fitness function to evolve controllers that distinguish between competing models (as in the grammar induction case) or expose some unobservable part of the target system (as in the gene network inference case). However, such terms could be formulated and included in the fitness function in future.

*2) Initialization*: Unlike the previous two applications, here the algorithm begins in the exploration phase, evolving neural network controllers using a default robot simulator.

*3) Exploration Phase*: Each pass through this phase begins with a random population of 100 genomes. Each controller in turn is evaluated on the simulated robot, and the fitness (forward displacement) is recorded. Selection and mutation is implemented differently from the previous two applications.

Once all of the genomes in the population have been evaluated, they are sorted in order of forward displacement, and the 50 genomes with the least displacement are deleted from the population. Fifty new genomes are selected to replace them from the remaining 50, using tournament selection, with a tournament size of 3. Each floating-point value of a copied genome has a 1% chance of undergoing a point mutation (replacement of the evolved value with a new random value). Of the 50 copied and mutated genomes, 12 pairs are randomly selected and undergo one-point crossover. The population is evolved for 30 generations. When this phase terminates, the controller with the best fitness is output and downloaded to the target robot, and the resulting sensor values are recorded. Both the evolved controller and resulting sensor time series are passed into the estimation phase.

*4) Estimation Phase*: The genetic algorithm operating in the estimation phase is similar to the one in the exploration phase. During the first pass through the estimation phase, a population of 100 random genomes is generated; during subsequent passes through this phase, the initial population contains 99 random genomes, and a copy of the best genome evolved during the previous pass. Each genome is evaluated in turn: the default simulation is modified according to the genome; the simulated robot is evaluated using the $i$ previously evolved controllers for 1000 time steps; $i$ sets of sensor time series are obtained; and the genome's rolling mean is calculated. Selection and mutation for the estimation phase is similar to that for the exploration phase. The population of the estimation phase is evolved for 30 generations as well, and outputs the most accurate simulator to the exploration phase, which begins again with a random population of controllers.

*5) Termination*: The algorithm iterates through the cycle shown in Fig. 2(d) 20 times, starting at the exploration phase with the default simulator, and terminating after the 20th pass through the estimation phase. This produces 20 evolved controllers and 20 simulator modifications.

*6) Validation*: There is no validation phase necessary for this example since in this case we know the *absolute error*. This error is not available to the algorithm.

### A. Evolutionary Robotics Results and Discussion

Fifty independent runs of the algorithm were conducted against the target robot. Fig. 15 shows the convergence toward

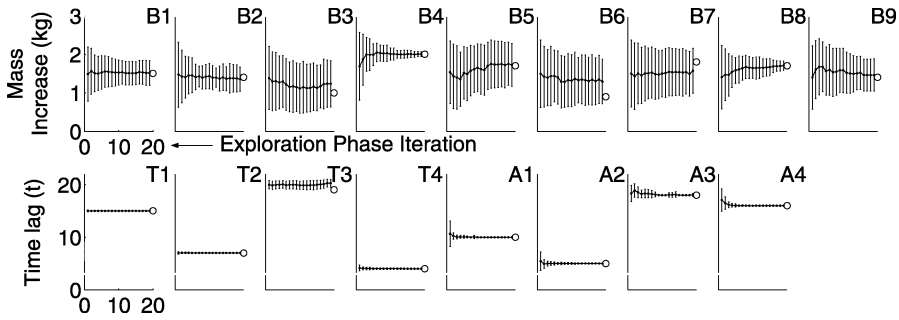


Fig. 15. Convergence toward the physical characteristics of the target robot. Each pass through the estimation phase produces a set of mass changes for each of the nine body parts of the robot (top row) and a set of time lags for each of the eight sensors of the robot (bottom row). The trajectories indicate the mean and standard deviation of the best guesses output by the estimation phase from the 50 independent runs. The each bar corresponds to the best guesses from that pass through the estimation phase. The open circles indicate the actual differences between the target robot and the starting default simulated robot (for example, the first body part of the target robot is 1.5 kg heavier than the corresponding body part of the default simulated robot).

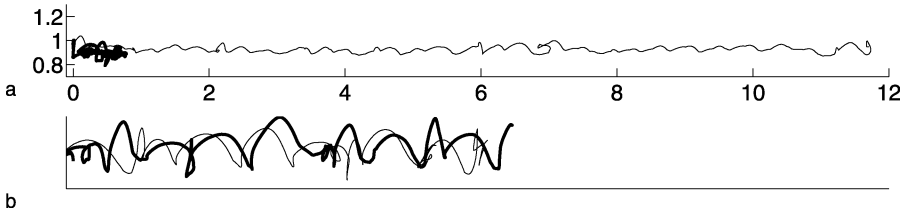


Fig. 16. Behavior recovery after controller transferal. After the first pass through the exploration phase, the best evolved controller caused the simulated robot to move as shown in (a) the trajectory of its center of mass is given by the thin line. The same controller was then supplied to the target robot, and the resulting trajectory of its motion is given by the thick line. The movement of the updated simulated robot after the 20th pass through the exploration phase (using the new best evolved controller) is given by the thin line in (b). The motion of the target robot using the same controller is given by the thick line in (b). The horizontal axis indicates forward distance, and the vertical axis indicates height (both are in meters).

the actual mass distribution and sensor time lags of the target robot. The figure makes clear that for all 50 runs, the algorithm was better able to infer the time lags of the eight sensors than the mass increases of the nine body parts (indicated by the convergence of the means and the negligible standard deviations toward the actual time lags on the bottom row compared with the large standard deviations on the top row). In the inference of the time lags, the algorithm only had difficulty with sensor 3, and tended to overestimate its particular lag. Also, the algorithm tended to have difficulty converging on the correct mass increases of most of the body parts, but did a relatively good job inferring the mass changes of body parts 4 and 8.

The fact that the algorithm had more difficulty with some parts of the robot and not others is not surprising: the asymmetric mass distribution of the robot causes some aspects of the robot's morphology to be more observable than others. For example, the time lag of a touch sensor, attached to a very heavy leg that is only ever dragged along the ground plane (such that the sensor always fires), is much more difficult to infer than another touch sensor attached to a leg that repeatedly touches the ground and lifts off again. Also, a sensor with a shorter time lag is probably easier to infer than one that has a long time lag.

Another reason that it was easier for the algorithm to infer sensor rather than body part characteristics is that the sensors themselves provide feedback about the robot. In other words, the algorithm automatically, and after only a few target trials, deduces the correct time lags of the target robot's sensors, but is less successful at indirectly inferring the masses of the body parts using the sensor data. As discussed in the previous two sections, often some of the components in a target system (in this case, an asymmetric robot) are less observable than other components.

One of the runs was selected at random, and the gait of the simulated robot was compared against the gait of the target robot, when both used the same evolved controller. Fig. 16(a) indicates the change in behaviors when the first evolved controller was transferred, and Fig. 16(b) shows the behavior change when the 20th evolved controller was transferred, during the last iteration through the algorithm's cycle.

This figure shows that even with an approximate description of the robot's mass distribution, the simulator is improved enough to allow smooth transfer of controllers from simulation to the target robot. Using the default, approximate simulation, there is a complete failure of transferal, as indicated by Fig. 16(a): the target robot simply moves randomly, and achieves no appreciable forward locomotion.

After 20 iterations through the algorithm, an improved simulator is available to the exploration phase, which evolves a controller that allows the simulated robot to move forward, although not as far as the original simulated robot [indicated by the shorter trajectory in Fig. 16(b) compared with Fig. 16(a)]. This is most likely due to the fact that the simulator is now accurately modeling the heavier mass of the target robot. Also, the new gait causes the robot to hop [indicated by the large vertical curves of the robot's center of mass in Fig. 16(b)] instead of walk [indicated by the steady trajectory of Fig. 16(a)]. In contrast to the first pass, the target robot exhibits very similar behavior to the simulated robot when it uses the same controller: both travel a similar distance (about 6.5 m), and both move in

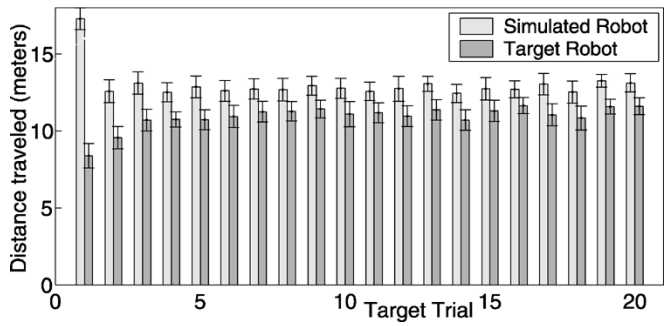


Fig. 17. Average transferal success after each target trial. The light gray bars indicate the average distance traveled by the simulated robot using the best evolved controller output by that pass through the exploration phase, over all 50 runs. The dark gray bars indicate the average distance traveled by the target robot using the same controller, during each target trial. Error bars indicate two units of standard deviation.

the same way (both exhibit a hopping gait that produces trajectories with similar frequencies and amplitudes).

Finally, for each pass through the exploration phase, the distance traveled by the simulated robot using the best evolved controller was averaged over all the 50 runs. Similarly, the distance traveled by the target robot using the same controller was averaged over the 50 runs. The results are shown in Fig. 17, which indicates that improvement in behavior transferal success, even using an approximate simulator, is a general phenomenon. On average, over the 50 independent runs, there is a drop by 50% in the distance traveled by the target robot, compared with the default simulated robot. After about five iterations through the algorithm's cycle there is only a statistically insignificant decrease in distance traveled between the two robots. Although not shown in Fig. 17, this similar distance is matched in all of the cases viewed by a qualitative similarity in gait patterns, as shown for a single run in Fig. 16(b).

In this section, we have shown that the estimation-exploration algorithm can be used to automatically evolve an accurate robot simulator. Even if the simulator does not describe the target robot perfectly, it does allow for automated transferal of evolved controllers from simulation to the target robot. Moreover, successful transferal becomes possible after only three or four target trials. In the next section, we extend this approach to automate the process of diagnosis and recovery for robots that undergo some unanticipated malfunction.

## VI. Application 4: Damaged Robot Recovery

For a robot to function for long periods of time in a hostile, unknown or remote environment, it must be able to deal autonomously with uncertainty: specifically, unanticipated internal damage or external environmental change. The recent difficulties with Jet Propulsion Laboratory's two Mars rovers provide a dramatic example: both robots suffered different, unanticipated partial failures [42]. Automatic recovery is most acute in such instances where human operators cannot manually repair or provide compensation for failure. In this work, we are concerned with catastrophic, highly nonlinear robot faults that require recovery controllers qualitatively different from the original controller.

Some work has been done on employing evolutionary algorithms to restore functionality after some unanticipated damage has occurred, but all of this work relies on massive numbers of hardware trials: robot recovery has been demonstrated in [4] and [47], and for electronic circuits in [10] and [37]. However, repeated generate-and-test algorithms for robotics is not desirable for several reasons: repeated trials may exacerbate damage and drain limited energy; long periods of time are required for repeated hardware trials; damage may require rapid compensation (e.g., power drain due to coverage of solar panels); and repeated trials continuously change the state of the robot, making damage diagnosis difficult.

Grefenstette and Ramsey [28] propose an algorithm for continual behavioral learning in a robot context which does not require large numbers of target trials. However, their system assumes that changes in the robot's environment (in their case, speed change of a prey robot) can be directly inferred using sensor data and provided to the behavior learning component. In their case, it is assumed that the prey robot may change speed, and their described algorithm assumes that speed change can be detected by the predator robot; the observed change in then fed directly into the predator's behavior learning component. However, the more difficult aspect of damage diagnosis and recovery is usually diagnosis: in the estimation-exploration algorithm it is assumed that robot or environmental changes are not known in advance, and must be inferred indirectly using the sensors that also drive behavior. Grefenstette and Ramsey state that indirect sensing of environmental change could be included into their system, but they do not provide any description as to how this could be accomplished.

As mentioned in the previous section, several types of plastic neural network controllers have been proposed that allow for rapid, lifetime adaptation to external perturbation (e.g., [19], [24] and [60]). Furthermore, Keymeulen *et al.* [36] have formulated an algorithm that continuously updates an internal model of sensor input/world response data obtained from a wheeled robot, and uses this model to evolve and download controllers to the robot during task execution. They have demonstrated that their algorithm greatly reduces the number of required hardware trials, compared with a similar model-free algorithm. However, none of these approaches generate a hypothesis describing what particular change the robot or its environment has experienced. Second, the evolved controllers have not been shown capable of fundamental reorganization when faced with unanticipated, catastrophic failure (such as the separation of a limb), as is demonstrated here for our proposed algorithm.

Srinivas [57] was one of the first researchers to study error diagnosis and recovery, but his approach, along with subsequent approaches ([1], [11], [20], [35], [61]), require online operation (repeated testing on the physical robot), and cannot handle unanticipated errors. Baydar and Saitou [4] proposed the first offline error diagnostic and recovery system, which relies on Bayesian inference for error diagnosis, and genetic programming [40] for error recovery. However, their algorithm also only handles prespecified error types.

Mahdavi and Bentley [47] recently demonstrated an online evolutionary algorithm that automatically recovers behavior for a physical robot. However, after damage the physical robot required 400 hardware trials and nearly seven hours to recover

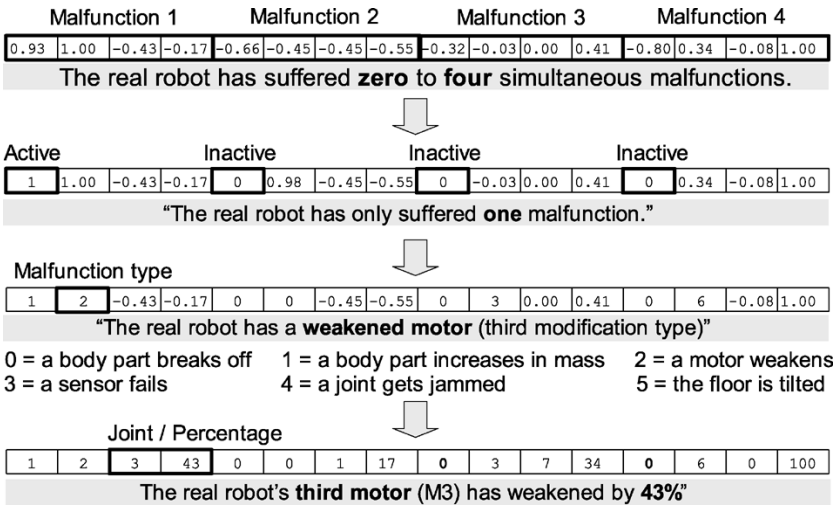


Fig. 18. Parsing of an example genome in the estimation phase. This particular genome has one active and three inactive genes. The resulting single modification (a weakening of the third motor by 43%) is applied to the default simulator.

72% of its original functionality. Their approach is similar to that portrayed in Fig. 13: evolution is continued on the target robot after transferal.

Here, we describe the application of the estimation-exploration algorithm to the problem of damaged robot recovery, which, for the results presented here, requires only ten target trials (on average) to restore functionality. Like the previous application, the algorithm automatically evolves two separate structures: a robot simulator that explains unanticipated internal damage or external environmental change suffered by the target robot; and a compensatory neural network controller that restores functionality to the target robot, given the evolved robot simulator. Below, we outline the application of the first five steps of the proposed algorithm to the problem of automated damage diagnosis and recovery; validation, as in the previous applications, is not used here.

*1) Characterization of the Target System*: Like the previous application, the target robot is an independent simulated robot that possesses some unknown differences in contrast to an initial, approximate simulator. We use the same quadrupedal robot as before (Fig. 14), but the target robot no longer has a larger mass and time-delayed sensors, but does suffer some unexpected malfunction or environmental change. The algorithm must then diagnose this malfunction using sensory feedback as before, and update the simulator to reflect this damage: if the target robot has lost a leg, then a good simulator must simulate a robot with the correct leg missing.

The model in this application is a genome that can encode zero to four simultaneous malfunctions. Each encoded malfunction is parameterized as to: which body part, joint, sensor, or motor it applies; what has happened (the joint has broken or the sensor has failed); and how severe the damage is (expressed as a percentage). Genomes are comprised of 16 floating-point values, and each genome is divided into four genes, each comprised of four values; each gene describes a possible malfunction. These values are parsed into a set of malfunctions to apply to the default simulated robot. Fig. 18 outlines a sample genome being parsed; refer to [8] for more details regarding the specifics of the genetic encoding. Thus, each genome produces some modification of the simulated robot and its environment; this simulation is treated as the candidate model. The quality of a model is again the ability of the given simulated robot to mimic the behavior of the crippled target robot.

This encoding gives a search space of $200^{16} = 6.5 \times 10^{36}$ different genomes. As the values are discretized, there are at most $4\ \text{genes} \times 2\ \text{active/inactive gene} \times 6\ \text{situations} \times 9\ \text{body parts} \times 200\ \text{percentages} = 86\,400$ possible sets of simulator modifications, although some of these are not unique because ordering is irrelevant, multiple modifications can be additive (two weakenings of the same motor by 10% is equivalent to one weakening of the same motor by 20%) and modifications encoded by inactive genes are not applied.

A test in this application is the same as for the previous application: genomes encode a vector of 68 synaptic weights for the controller. The quality of a labeled controller is how far it enables the crippled robot to move forward during 1000 time steps of the simulation.

*2) Initialization*: As in the previous application, we begin with a default simulation of the quadrupedal robot and its environment, and initiate the exploration phase to evolve a neural network controller for it. The best evolved controller is then downloaded onto the damaged target robot, and the resulting sensory data, along with the evolved controller, are fed into the estimation phase in order to improve the simulation.

*3) Exploration Phase*: During each pass through the exploration phase a population of 200 genomes is evolved for 40 generations. Before each evaluation, the default simulator is modified (i.e., the simulated robot is damaged) according to the evolved set of modifications supplied by the estimation phase. Selection and mutation are implemented the same way as in the previous application. At the end of this phase, the best controller is output to the target robot.

*4) Estimation Phase*: Each pass through the estimation phase begins with a random population of 200 genomes; on the second and subsequent passes through this pass, the random population is seeded with the best simulator modification evolved during the previous pass. Each genome in turn is parsed; the resulting

TABLE II
UNANTICIPATED SITUATIONS TESTED

| Case | Explanation | Perfect Runs | Total Runs |
|---|---|---|---|
| 1 | One motor weakens by 50%. | 10 | 12 |
| 2 | One body part increases in mass by 200%. | 10 | 12 |
| 3 | One of the lower legs breaks off. | 11 | 12 |
| 4 | One of the lower legs breaks off, and a sensor fails by 50%. | 3 | 12 |
| 5 | The second touch sensor fails by 50%. | 11 | 12 |
| 6 | One of the joints jams by 50%. | 9 | 12 |
| 7 | One of the lower legs breaks off, and one of the joints jams by 50%. | 1 | 12 |
| 8 | One of the lower legs breaks off, one of the joints jams by 50%, and one of the sensors breaks by 50%. | 0 | 12 |
| 9 | Nothing breaks. | 12 | 12 |
| 10 | Robot on a 30 degree horizontal slope. | 11 | 12 |
| 11 | One of the hidden neurons fails by 50%. | 0 | 12 |
| 12 | Two motor neurons output the same value. | 0 | 12 |
| 13 | A body part decreases in mass by 50%. | 0 | 12 |

modifications (if any) are applied to the default simulator; the modified simulated robot is evaluated using the $i$ controllers already tested on the target robot; and both the sensor logs from the damaged target robot and those obtained from the simulation are compared using the rolling mean metric [8] to assess the accuracy of the current genome. Once all of the genomes in the population have been evaluated, the population is sorted according to accuracy, and selection, mutation and crossover is applied. Each pass through the estimation phase evolves the population for 40 generations. At the end of the pass, the best set of simulator modifications is passed to the exploration phase.

*5) Termination*: Starting at the exploration phase, the algorithm iterates through the cycle shown in Fig. 2(e) ten times, leading to ten evolved controllers, ten target tests, and ten evolved simulations. See Table I for a summary of the application of the estimation-exploration algorithm to this problem.

*6) Validation*: There is no validation phase necessary for this example since in this case we know the *absolute error*. This error is not available to the algorithm.

### A. Damaged Robot Recovery Results and Discussion

Thirteen different target robots were tested: each suffered a different unanticipated situation. Table II describes the situations. The algorithm was then run 20 times against each of the 13 target robots. The estimation phase attempts to evolve simulator modifications that will describe this situation: for each of the 13 situations, there are only a few genomes (or none, in the case of situations 11–13) that will describe it perfectly, out of the 86 400 possible situations. Each run requires about 4 h of computation on a 1 GHz PC.

Fig. 19 documents the performance of a single run of the control algorithm against a target robot that has suffered a 50% failure of one of its touch sensors (situation 5 in Table II). The algorithm evolves an approximate description of the damage after the first target test, and evolves a perfect description after the second test. This indicates that as the algorithm accumulates more data from the target robot by causing it to behave in different ways, it is better able to model the target robot's situation. Also, Fig. 19(a) shows that, armed with an accurate model after

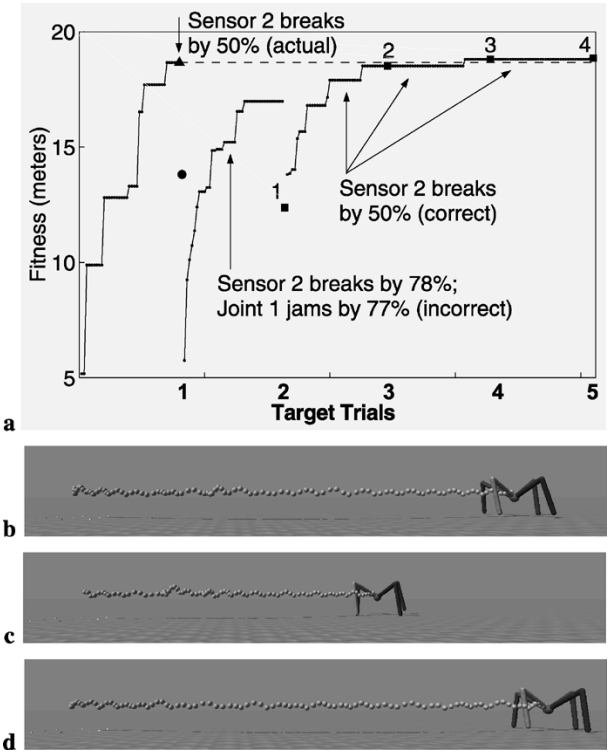


Fig. 19. (a) The evolutionary progress of a sample run against situation 5: one of the angle sensors breaks by 50%. The dotted lines indicate the progress of the first five passes through exploration phase. The captions indicate the best simulator modifications evolved by four passes through the estimation phase. The triangle shows the original distance traveled by the target robot. The circle indicates distance traveled after suffering the failure. The squares indicate the distance traveled by the damaged target robot during each of the four trials. (b)–(d) Images of the target robot's motion: (b) before the unanticipated situation; (c) after encountering the unanticipated situation; and (d) during the fifth target trial.

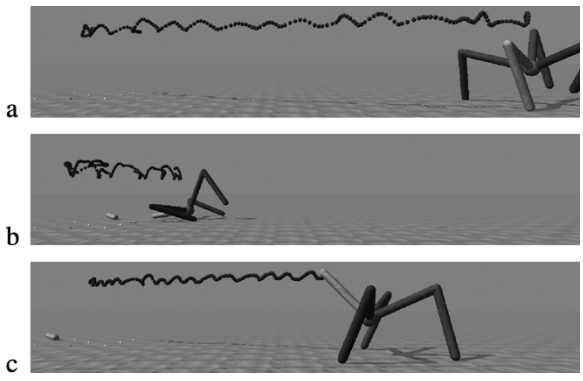


Fig. 20. Automated recovery after a catastrophic failure using the proposed algorithm. (a) The behavior of the default simulated robot using the best controller evolved during the first pass through the exploration phase. (b) The target robot has lost part of a leg through joint separation, and the robot attempts to move using the first evolved controller. (c) The motion of the damaged target robot during the second target trial.

the second test, the algorithm is able to evolve a compensatory controller for the damaged robot that restores functionality.

Fig. 20 indicates that the proposed algorithm is even able to provide compensatory controllers when the target robot suffers catastrophic failure, such as a change in the robot's topology (one of the lower legs has broken off, represented by situation 3 from Table II). Here, the algorithm correctly identifies the

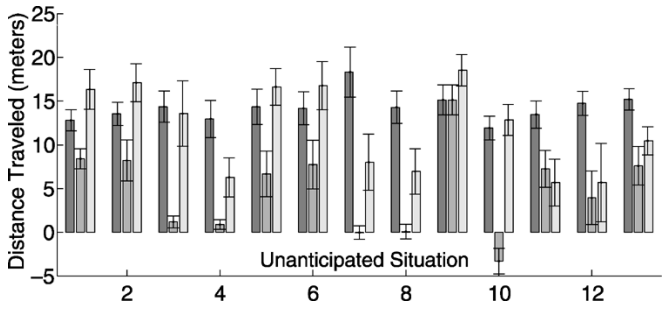


Fig. 21. The average performance of the proposed algorithm against all 13 damaged robots. The algorithm was run ten times against each situation. The white bars indicate the average distance traveled by the default simulated robot using the first evolved controller. The light gray bars indicate the average distance traveled by the target robot during the first target trial. The dark gray bars indicate the average distance traveled by the target robot during the tenth target trial.

damage after the second pass through the estimation phase, and the exploration phase uses this accurate simulator to evolve a new controller that allows the damaged robot to move using only three legs. Note the difference in trajectories shown in Fig. 20(a) and (c): the original gait caused the robot to move using long hops [indicated by the long loops in Fig. 20(a)], but the damaged robot uses a compensatory controller that causes it to move in short, hesitant hops] indicated by the short loops in Fig. 20(b)]. Thus, the algorithm *automatically* diagnoses mild or severe malfunction, and either modifies the gait slightly over subsequent trials or evolves a qualitatively different gait, depending on the damage incurred.

Fig. 21 reports the ability of the proposed algorithm to recover function for damaged robots. The algorithm was able to restore functionality (regarded as a statistically significant increase in mean performance between the first and tenth hardware trial) for all but damage scenarios 6, 11, and 13. Even in these three cases, many of the runs produce a significant increase in performance. Note that even in the catastrophic cases such as 3, 4, 7, and 10 (indicated by almost complete lack of forward motion during the first hardware trial) some functionality was restored, indicating that our algorithm may be useful for recovering some function from severely damaged robots.

Note also that both algorithms were able to successfully recover when faced with an unanticipated environmental change: the horizontal canting of the ground plane (situation 10). This was probably due to the drastic effect on all sensors induced by this change. It is important to note that the proposed algorithm can distinguish between internal damage and external environmental change, and reconstruct the change in simulation, based solely on indirect sensory information.

Most encouragingly, some functionality was restored in cases 11 and 12, in which there was no genome that could perfectly describe the malfunction. Further study is required in order to understand how truly unanticipated situations can best be approximated by a set of simulator modifications.

## VII. Discussion

The previous four sections have described the application of the estimation-exploration algorithm to four separate problems: grammar induction, gene network inference, the evolution of an accurate robot simulator, and automated diagnosis and recovery from unanticipated robot malfunction. These four applications have not only demonstrated that our approach is problem domain independent, but have also highlighted its various properties and benefits. Before we summarize the results from these applications, however, it is worthwhile to go into more detail about the differences between the proposed algorithm and other coevolutionary algorithms.

### A. Differences From Pure Coevolutionary Algorithms

The main difference between the estimation-exploration algorithm and other coevolutionary systems proposed so far is that our algorithm is a hybrid system: the fitness of both the model and test populations is influenced by the opposing population, but also by the behavior of an additional, third component—the target system. In other coevolutionary algorithms, the fitness of an individual is influenced by one or more individuals from another (or the same) population. For this reason, the fitness of a coevolving individual in such systems is purely subjective; it is calculated based only on other individuals. However, in our system, there is a static component to search, which is the ability of a model to mimic the observed behavior of a target system: as long as the target system does not alter its internal structure over time, there is an objective aspect to the search for better models. For this reason, we view previous coevolutionary systems as pure coevolutionary systems, and the estimation-exploration algorithm as a hybrid coevolutionary system.

### B. Application of the Algorithm

In the grammar induction problem, it was shown that an intelligent test is often one that distinguishes between alternative candidate models, a concept related to that propounded in the competitive coevolutionary literature: the quality of an individual in one population is proportional to its ability to induce a performance gradient in the subset of individuals challenging it from the competing population [12], [22], [33]. There are several reasons cited as to why such a test is desirable, but there is a specific advantage in the context of the estimation-exploration algorithm, described next.

If a test does *not* induce disagreement between models, and the test is passed to the target DFA for classification, then on the subsequent pass through the estimation phase there will be an absolute but not a relative change in the fitnesses of the best models: the fitnesses of the $k$ best models from each subpopulation will either all go up by one (if they match the classification of the target DFA) or down by one (if they disagree with the target DFA). If the best models all improve in fitness, there is little chance for evolutionary improvement: new models can only classify any previous sentences not yet correctly classified by the current best models, if there are any. If the best models all experience a drop in fitness, then there is sufficient room for improvement: a new model may correctly classify all the sentences that the best model(s) do, as well as the new sentence that they do not.

However, as the inference process proceeds, and the models in the estimation phase improve, there is an increasing likelihood that the best models will correctly classify any incoming sentence. Therefore, there will be a diminishing return over the

inference process of proposing sentences for classification that the best models already agree on. However, if a sentence is proposed for classification by the target DFA that causes disagreement among the best models, there is a guarantee that $k/2$ of the best models will incorrectly classify the new sentence, thereby ensuring the possibility of further model improvement.

Tests that cause model disagreement have an added benefit: it helps to uncover less observable components of the target system. For example, in the grammar induction application, intelligent testing tended to elicit much more balanced classifications from unbalanced target DFAs than random testing was able to [see Fig. 7(c)]. For many of these unbalanced DFAs, the proposed algorithm was able to extract enough of the minority classifications using intelligent testing to evolve a much more accurate model, on average, than the same algorithm but with random testing. In such cases when the hidden system has low observability, tests must be formulated that produce more of the underrepresented output in order to infer the internal structure of the system: in the estimation-exploration algorithm this occurs as a natural result of evolving tests that cause disagreement among models.

For the gene network inference problem, we have shown that it is possible to automatically evolve the regulatory connections between a set of genes given a set of input and output gene product concentrations. Moreover, our approach does not require invasive, expensive, slow, and disruptive experiments such as knockout or lesion studies. Rather, the exploration phase carefully evolves a low-cost experiment (a set of initial chemical concentrations that trigger gene regulation) that yields a large amount of information about the physical system.

Like the grammar induction problem, it was found that evolving useful tests, instead of proposing random ones, speeds the discovery of accurate gene network models, thus requiring a minimum of physical experimentation. Moreover, it was found that the usefulness of evolving tests increases as problem difficulty increases: it was found that evolving tests is increasingly useful as the hidden gene networks increase in size and connectivity. This finding bodes well for the scalability of our approach, suggesting that our algorithm may prove very useful for genetic network inference in particular, and for systems biology research in general.

In Section V, we described the application of the estimation-exploration algorithm to the problem of automatically generating controllers for robots. In that application the hidden target system is a robot with unknown morphological differences compared with an initial default simulated robot. The algorithm evolves controllers for the robot in simulation, which are then tested on the target robot. The resulting sensor data, along with the evolved controller, serve as input/output data pairs which are used to evolve improvements to the robot simulator. It was shown that this approach automatically produces accurate enough simulators for the successful transferal of controllers from simulation to the target robot, using a minimum of trials on the target robot. This approach is appealing because: repeated testing on a physical robot is time-consuming and potentially damaging; the sensors already being used to drive behavior also act as diagnostic sensors, returning indirect information about the target robot's morphological characteristics; and our approach could be combined with other approaches that have already been suggested for automating the process of evolved controller transferal.

In the last application, we employed the estimation-exploration algorithm to automatically diagnose unanticipated malfunctions suffered by the target robot, and automatically evolve compensatory controllers for it. As in the previous three applications, recovery was achieved with a small number of target trials. In this application a maximum of ten target trials were sufficient to restore functionality in the face of: minor, catastrophic, or compound damages; no damage at all; changes in the robot's environment; or malfunctions that could not be perfectly expressed in simulation. Most importantly, the algorithm was able to automatically distinguish and describe these diverse situations using only sensory feedback and the rolling mean metric, which quantitatively compares behaviors between highly coupled, highly nonlinear dynamic robots. Once a correct diagnosis is discovered the algorithm is often able to evolve a compensatory controller, often evolving qualitatively different gaits in response to fundamental changes experienced by the target robot.

### C. Model Accuracy Versus Structural Accuracy

When performing system identification, it is possible to generate a model that successfully mimics the input–output behavior of the target system, which we refer to as *model accuracy*, but fails to reflect the actual internal structure of the target system, which we call *structural accuracy*. Whether our proposed algorithm actually captures structural as well as model accuracy depends on the specific application.

For grammar induction, there are often several DFAs that produce identical classifications. So for this application, it is impossible to determine, using any algorithm, whether a model DFA has captured the actual internal structure of the target DFA: absolute error of a candidate model is simply approximated using a large set of previously unseen sentences.

In the gene network application, our algorithm does capture the internal structure of the target gene networks: absolute error is considered to be the distance between the internal structures of the model and target networks. As can be seen in Fig. 9(d), the internal structures of the best model gene networks produced by the estimation-exploration algorithm are closer to those of the target networks than those models produced by the other two algorithms (indicated by the final absolute errors of the best models). However, this correlation between model and structural accuracy depends on the way in which we have chosen to model gene networks: validation methods for determining whether a given gene network model actually captures the regulation inherent in an actual biological gene network is currently being investigated.

In the case of the first robot application, Fig. 15 indicates that all of the runs tend to converge on the actual internal structure of the target system: the mass distribution and sensor time lags of the target robot. In the case of the second robot application, Table II reports that for many of the damage scenarios, the estimation-exploration algorithm actually discovers the perfect description of the damage (see column 3). However, for these two applications we are most interested in determining how well the

TABLE III
ALGORITHM INFERENCE ABILITY

| | Target System DOFs | Standard GA | Random Testing | Intelligent Testing |
|---|---|---|---|---|
| **Grammar Induction** | | | | |
| Least Target Evaluations | 30 | 0 / 30 targets | 0/30 targets | **21**/30 targets |
| Least Model Evaluations | 30 | 2 / 30 targets | 0/30 targets | **14**/30 targets |
| **Gene Network Inference** | | | | |
| Target Evaluations | | 100 | 56 | 55 |
| Best Model Accuracy | $5 \times 5 = 25$ (for $n = 5$, (for $k = 2$) | 0/20 targets | 1/20 targets | 1/20 targets |
| Best Model Accuracy | 25 (for $n = 5$, (for $k = 5$) | 0/20 targets | 3/20 targets | **7**/20 targets |
| Best Model Accuracy | $10 \times 10 = 100$ (for $n = 10$, (for $k = 2$) | 0/20 targets | 0/20 targets | **10**/20 targets |
| Best Model Accuracy | 100 (for $n = 10$, (for $k = 10$) | 0/20 targets | 1/20 targets | **19**/20 targets |

algorithm can approximate an indescribable structural change to the robot or its environment and recover from it.

### D. Comparison of Performance

Table III summarizes the inference ability of the algorithm for the grammar induction and gene network inference applications. The first row reports that for 21 of the 30 target DFAs, the estimation-exploration algorithm was able to find a perfect model using fewer target evaluations than the other two control algorithms [see Fig. 7(b)]. The remaining nine targets showed no statistical improvement using intelligent testing. Moreover, for 14 of the 30 target DFAs, the proposed algorithm discovered a perfect model using significantly fewer model evaluations [see Fig. 7(a)]. In the case of gene network inference, all three algorithms performed $1 \times 10^7$ model evaluations, the standard GA performed 100 target evaluations, the iterative algorithm with random testing performed 56 target evaluations and the estimation-exploration algorithm performed 55 target evaluations. As can be seen from Table III, the proposed algorithm consistently produces more accurate models for large and more densely connected gene networks (see Fig. 9). This indicates that the estimation-exploration algorithm is scalable: intelligent testing becomes increasingly useful as the size of the target system increases and its observability decreases.

Table IV summarizes the recovery ability of the proposed algorithm for the latter two robot applications. In the third application, the target robot had a different mass distribution and significant sensor time lags, compared with its corresponding robot simulator. As a control, a neural controller was evolved directly on this target robot using the genetic algorithm described in Section V. Ten independent runs of this control were performed. After 600 evaluations in each run, the best controller was extracted, and the mean distance achieved by the robot using these ten networks in turn was found to be 5.3 m. The intervals in this table represent standard deviation, indicating the distribution of distances traveled by the best robot from each run, for both algorithms. Both the means and the standard deviations

TABLE IV
ALGORITHM RECOVERY ABILITY

| Physical Tests | Mean Fitness | Total Runs | Physical Tests | Mean Fitness | Total Runs |
|---|---|---|---|---|---|
| Algorithm | | | Control | | |
| **Evolutionary Robotics** | | | | | |
| 20 | 5.3m ($\pm$1.8) | 50 | 600 | 5.3m ($\pm$1.6) | 10 |
| **Damaged Robot Recovery** | | | | | |
| 10 (Lower leg separates) | 14.0m ($\pm$6.5) | 10 | 1550 (Lower leg separates) | 7.3m ($\pm$1.7) | 10 |

indicate that in this case, the estimation-exploration algorithm caused the target robot to achieve the same performance as the control algorithm, in which all evolution is performed directly on the target robot, but using only 20 physical tests, compared with 600. Thus, there was an order of magnitude improvement in extracting a desired behavior from the target robot.

A similar control was devised for the damaged robot recovery application. Again a genetic algorithm was used to evolve neural controllers directly on the target robot, which in this case was a damaged quadrupedal robot (one of the lower legs separates completely from the body). In this case, even after 1550 evaluations (evolving a population of 100 controllers for 30 generations), only half of the mean distance was achieved compared with the mean distance traveled by the target robot after ten physical tests using the proposed algorithm. Although there is a slight overlap in the standard deviations, it is clear that the proposed algorithm performs better than simply evolving controllers directly on the target robot, using two orders of magnitude fewer target trials (1500 versus 10).

## VIII. CONCLUSION

We have introduced the estimation-exploration algorithm, a coevolutionary approach to system identification. We have shown how our algorithm differs from other pure coevolutionary systems because it does not so much introduce a new advance in coevolutionary theory, but rather introduces a systematic, domain independent method for performing synthesis or analysis using coevolution.

Unlike other coevolutionary systems, the estimation-exploration algorithm evolves candidate models that are influenced not only by the competing population of tests, but also by the target system. This removes two potential pathologies from the algorithm (cycling and overspecialization) which are often suffered by other coevolutionary algorithms. The third pathology, disengagement, has been observed to occur in our algorithm, but not for the four applications shown here. We have formulated two mechanisms that combat disengagement, and are currently investigating additional mechanisms.

We have documented its application to four very different problems, thus demonstrating its problem domain independence. The algorithm has several other benefits that have been highlighted in the various applications: coupled automation of both models and intelligent tests; the active minimization of target testing; increased usefulness as target systems increase in size and complexity; increased usefulness for target systems

with low observability; and automating the extraction of useful behavior from a hidden system.

Thus, our algorithm performs well at *inference*: it can automatically construct a hidden system that explains the gathered data. Active learning is also concerned with the intelligent selection of tests [16], but in that domain a classifier is used to describe input/output relations; no explicit model of the target system is generated. This is an important distinction to make, because by building explicit models to explain the data rather than learning an approximation of the input/output relationships, we obtain several benefits:

1) we can use the resultant model to learn more about the hidden system (we can use it to test hypotheses about the actual system);
2) we can uncover possible causalities rather than just correlations ("gene $x$ regulates gene $y$," rather than "the expressions of genes $x$ and $y$ are correlated");
3) we can obtain a comprehensible description of some change in the hidden system (such as what damage has incurred, and where);
4) we can generate some useful behavior for it (continuation of function by circumventing the damage).

The third and fourth property allow us to *recover* function from the hidden system using a minimum number of tests. Second, unlike active learning, tests can be constructed, allowing for the exploration of a possibly infinite space of tests, rather than selected from some fixed space of tests with equal dimensionality.

Furthermore, using evolution allows us to synthesize models and tests for systems about which little is known. This is important in nonlinear black-box system identification tasks for which not even the underlying topology of the hidden system is known [56]. Artificial evolution could be used to build explicit models directly from observed data in such instances, a challenge that has not yet been addressed in the system identification literature, to the best of the authors' knowledge. For example the algorithm could be used in a remote robotics application to synthesize a model of a novel environment based only on the robot's sensor data: this is an attractive avenue for future study. In future work, we also plan to apply the algorithm to a range of actual physical systems, including physical robots and real biological networks.

## REFERENCES


[1] M. G. Abu-Hamdan and A. S. El-Gizawy, "Computer aided monitoring system for flexible assembly operations," *Comput. Ind.*, vol. 34, pp. 1–10, 1997.

[2] A. Adamatzky, M. Komosinski, and S. Ulatowski, "Software review: Framsticks," *Kybernetes: Int. J. Syst. Cybern.*, vol. 29, pp. 1344–1351, 2000.

[3] A. Arkin, P. Shen, and J. Ross, "A test case for correlation metric construction of a reaction pathway from measurements," *Science*, vol. 277, pp. 1275–1279, 1997.

[4] C. M. Baydar and K. Saitou, "Off-line error prediction, diagnosis and recovery using virtual assembly systems," in *Proc. IEEE Int. Conf. Robotics Autom.*, 2001, pp. 818–823.

[5] F. Bergadano and D. Gunetti, *Inductive Logic Programming: From Machine Learning to Software Engineering*. Cambridge, MA: MIT Press, 1995.

[6] J. Bongard and H. Lipson, "'Managed challenge' alleviates disengagement in coevolutionary system identification," in *Proc. Genetic Evol. Comput. Conf.*, 2005, pp. 531–538.

[7] J. Bongard and R. Pfeifer, "Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny," in *Proc. Genetic Evol. Comput. Conf.*, 2001, pp. 829–836.

[8] J. C. Bongard and H. Lipson, "Automated robot function recovery after unanticipated failure or environmental change using a minimum of hardware trials," in *Proc. NASA/DoD Conf. Evolvable Hardware*, Seattle, WA, 2004, pp. 169–176.

[9] J. C. Bongard and H. Lipson, "Automating genetic network inference with minimal physical experimentation using coevolution," in *Proc. Genetic Evol. Comput. Conf.*, Seattle, WA, 2004, pp. 333–345.

[10] D. W. Bradley and A. M. Tyrrell, "Immunotronics: Novel finite-state-machine architectures with built-in self-test using self-nonself differentiation," *IEEE Trans. Evol. Comput.*, vol. 6, no. 3, pp. 227–238, Jun. 2002.

[11] S. Brnyjolfsson and A. Arnstrom, "Error detection and recovery in flexible assembly systems," *Int. J Adv Manuf. Technol.*, vol. 5, pp. 112–125, 1990.

[12] A. Bucci, J. B. Pollack, and E. D. De Jong, "Automated extraction of problem structure," in *Proc. Genetic Evol. Comput. Conf.*, 2004, pp. 501–512.

[13] T. Chen, H. L. He, and G. M. Church, "Modeling gene expression with differential equations," in *Proc. Pacific Symp. Biocomput.*, vol. 4, 1999, pp. 29–40.

[14] O. Cicchello and S. C. Kremer, "Inducing grammars from sparse data sets: A survey of algorithms and results," *J. Mach. Learn. Res.*, vol. 4, pp. 603–632, 2003.

[15] D. Cliff and G. F. Miller, "Tracking the red queen: Measurements of adaptive progress in coevolutionary simulations," in *Proc. Eur. Conf. Artif. Life*, 1995, pp. 200–218.

[16] D. Cohn, L. Atlas, and R. Ladner, "Improving generalization with active learning," *Mach. Learn.*, vol. 15, no. 2, pp. 201–221, 1994.

[17] H. de Jong, "Modeling and simulation of genetic regulatory systems: A literature review," *J. Comput. Biol.*, vol. 9, no. 1, pp. 69–105, 2002.

[18] P. D'haeseleer, S. Liang, and R. Somogyi, "From co-expression clustering to reverse engineering," *Bioinformatics*, vol. 16, no. 8, pp. 707–726, 2000.

[19] E. A. DiPaolo, "Homeostatic adaptation to inversion of the visual field and other sensorimotor disruptions," in *From Animals to Animats 6*, J. A. Meyer, A. Berthoz, D. Floreano, H. L. Roitblat, and S. W. Wilson, Eds. Cambridge, MA: MIT Press, 2000, pp. 440–449.

[20] E. Z. Evans and S. G. Lee, "Automatic generation of error recovery knowledge through learned activity," in *Proc. IEEE Int. Conf. Robotics Autom.*, 1994, pp. 2915–2920.

[21] R. Featherstone and D. E. Orin, "Robot dynamics: Equations and algorithms," in *Proc. IEEE Int. Conf. Robotics Autom.*, 2000, pp. 826–834.

[22] S. G. Ficici and J. B. Pollack, "Pareto optimality in coevolutionary learning," in *Proc. 6th Eur. Conf. Adv. Artif. Life:*, 2001, pp. 316–327.

[23] D. Floreano and F. Mondada, "Evolutionary neurocontrollers for autonomous mobile robots," *Neural Netw.*, vol. 11, pp. 1461–1478, 1998.

[24] D. Floreano and J. Urzelai, "Neural morphogenesis, synaptic plasticity, and evolution," *Theory Biosci.*, vol. 120, pp. 225–240, 2001.

[25] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence Through Simulated Evolution*. New York: Wiley, 1966.

[26] P. Funes and J. Pollack, "Computer evolution of buildable objects," in *Evolutionary Design by Computer*, P. Bentley, Ed. San Francisco, CA: Morgan Kauffman, 1999, pp. 387–403.

[27] M. Gevers, "A decade of progress in iterative control design: From theory to practice," *J. Process Control*, vol. 12, no. 4, pp. 519–531, May 2002.

[28] J. J. Grefenstette and C. L. Ramsey, "An approach to anytime learning," in *Proc. 9th Int. Workshop Mach. Learn.*, 1992, pp. 189–195.

[29] W. D. Hillis, "Coevolving parasites improve simulated evolution as an optimization procedure," *Physica D*, vol. 42, pp. 228–234, 1990.

[30] G. S. Hornby and J. B. Pollack, "Creating high-level components with a generative representation for body-brain evolution," *Artif. Life*, vol. 8, no. 3, pp. 223–246, 2002.

[31] H. Iba and A. Mimura, "Inference of a gene regulatory network by means of interactive evolutionary computing," *Inf. Sci.*, vol. 145, pp. 225–236, 2002.

[32] N. Jakobi, "Evolutionary robotics and the radical envelope of noise hypothesis," *Adaptive Behav.*, vol. 6, no. 1, pp. 131–174, 1997.

[33] E. D. De Jong and J. B. Pollack, "Ideal evaluation from coevolution," *Evol. Comput.*, vol. 12, no. 2, pp. 159–192, 2004.

[34] H. Juillé, "Methods for statistical inference: Extending the evolutionary computation paradigm," Ph.D. dissertation, Brandeis Univ., Waltham, MA, May 1999.

[35] J. F. Kao, "Optimal recovery strategies for manufacturing systems," *Eur. J. Oper. Res.*, vol. 80, pp. 252–263, 1995.
[36] D. Keymeulen, M. Iwata, Y. Kuniyoshi, and T. Higuchi, "Online evolution for a self-adapting robotics navigation system using evolvable hardware," *Artif. Life*, vol. 4, pp. 359–393, 1998.
[37] D. Keymeulen, A. Stoica, and R. Zebulum, "Fault-tolerant evolvable hardware using field programmable transistor arrays," *IEEE Trans. Reliability (Special Issue on Fault-Tolerant VLSI Syst.)*, vol. 49, pp. 305–316, Sep. 2000.
[38] S. Kikuchi, D. Tominaga, M. Arita, and M. Tomita, "Pathway finding from given time-courses using genetic algorithm," *Genome Informatics*, vol. 12, pp. 304–305, 2001.
[39] H. Kitano, *Foundations of Systems Biology*. Cambridge, MA: MIT Press, 2001.
[40] J. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.
[41] S. Kumar and P. Bentley, *On Growth, Form and Computers*. New York: Academic, 2003.
[42] NASA Jet Propulsion Laboratory. (2004) Mars exploration rovers. [Online]. Available: http: //www.jpl.nasa.gov/mer2004/
[43] H. Lipson and J. B. Pollack, "Automatic design and manufacture of artificial lifeforms," *Nature*, vol. 406, pp. 974–978, 2000.
[44] L. Ljung, *System Identification: Theory for the User*. Englewood Cliffs, NJ: Prentice-Hall, 1999.
[45] S. M. Lucas and T. J. Reynolds, "Learning DFA: Evolution versus evidence driven state merging," in *Proc. Congr. Evol. Comput.*, 2003, pp. 351–358.
[46] S. Luke, S. Hamahashi, and H. Kitano, "'Genetic' programming," in *Proc. Genetic Evol. Comput. Conf.*, 1999, pp. 1098–1105.
[47] S. H. Mahdavi and P. J. Bentley, "An evolutionary approach to damage recovery of robot motion with muscles," in *Proc. 7th Eur. Conf. Artif. Life*, 2003, pp. 248–255.
[48] E. Mjolsness, D. H. Sharp, and J. Reinitz, "A connectionist model of development," *J. Theor. Biol.*, vol. 152, pp. 429–454, 1991.
[49] B. Olsson, "Coevolutionary search in asymmetric spaces," *Inf. Sci.*, vol. 133, pp. 103–125, 2001.
[50] R. Parekh and V. Honavar, "Automata induction, grammar inference, and language acquisition," in *The Handbook of Natural Language Processing*, R. Dale, H. Moisl, and H. Somers, Eds. New York: Marcel Dekker, 2000, pp. 727–764.
[51] J. B. Pollack, H. Lipson, S. Ficici, P. Funes, G. Hornby, and R. Watson, "Evolutionary techniques in physical robotics," in *Evolvable Systems: From Biology to Hardware*, J. Miller, Ed. Berlin, Germany: Springer-Verlag, 2000, pp. 175–186.
[52] M. A. Potter and K. A. De Jong, "Cooperative coevolution: An architecture for evolving coadapted subcomponents," *Evol. Comput.*, vol. 8, no. 1, pp. 1–29, 2000.
[53] T. Reil and P. Husbands, "Evolution of central pattern generators for bipedal walking in a real-time physics environment," *IEEE Tran. Evol. Comput.*, vol. 6, no. 2, pp. 159–168, Apr. 2002.
[54] C. D. Rosin and R. K. Belew, "New methods for competitive coevolution," *Evol. Comput.*, vol. 5, no. 1, pp. 1–29, 1997.
[55] K. Sims, "Evolving 3D morphology and behavior by competition," *Artif. Life IV*, pp. 28–39, 1994.
[56] J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, B. Delyon, P.-Y. Glorennec, H. Hjalmarsson, and A. Juditsky, "Nonlinear black-box modeling in system identification: A unified overview," *Automatica*, vol. 31, no. 12, pp. 1691–1724, 1995.
[57] S. Srinivas, "Error recovery in robot systems," Ph.D. dissertation, Calif. Inst. Technol., Pasadena, CA, 1977.
[58] K. O. Stanley and R. Miikkulainen, "Competitive coevolution through evolutionary complexification," *J. Artif. Intell. Res.*, vol. 21, pp. 63–100, 2004.
[59] A. Thompson, "Artificial evolution in the physical world," in *Evolutionary Robotics: From Intelligent Robots to Artificial Life (ER'97)*, T. Gomi, Ed. Ottawa, ON, Canada: AAI Books, 1997, pp. 101–125.
[60] S. Tokura, A. Ishiguro, H. Kawai, and P. Eggenberger, "The effect of neuromodulations on the adaptability of evolved neurocontrollers," in *Proc. 6th Eur. Conf. Artif. Life*, J. Kelemen and P. Sosik, Eds., 2001, pp. 292–295.
[61] M. L. Visinsky, J. R. Cavallaro, and I. D. Walker, "Expert system framework for fault detection and fault tolerance in robotics," *Comput. Elect. Eng.*, vol. 20, pp. 421–435, 1994.
[62] R. A. Watson and J. B. Pollack *et al.*, "Coevolutionary dynamics in a minimal substrate," in *Proc. Genetic Evolutionary Comput. Conf.*, L. Spector and E. D. Goodman, Eds., 2001, pp. 702–709.
[63] V. Zykov, J. Bongard, and H. Lipson, "Coevolutionary variance guides physical experimentation in evolutionary system identification," in *Proc. NASA/DoD Conf. Evolvable Hardware*, 2005, pp. 213–220.
[64] S. G. Ficici, "Solution concepts in coevolutionary algorithms," Ph.D. dissertation, Comput. Sci. Dept., Brandeis Univ., Waltham, MA, 2004. Tech. Rep. CS-03-243.

**Josh C. Bongard** received the B.Sc. degree (Honors) in computer science from McMaster University, Hamilton, ON, Canada, in 1997, the M.Sc. degree in evolutionary and adaptive systems from the School of Cognitive and Computing Sciences, University of Sussex, Brighton, U.K. in 1999, and the Ph.D. degree from the Artificial Intelligence Laboratory, University of Sussex, for research in the field of evolutionary robotics.

He is currently working as a Postdoctoral Researcher in the Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY.

**Hod Lipson** received the B.Sc. and Ph.D. degrees in mechanical engineering and computer-aided design from the Technion—Israel Institute of Technology, Haifa.

He joined the faculty of the Departments of Mechanical and Aerospace Engineering and Computing and Information Science, Cornell University, Ithaca, NY, in 2001. Prior to this appointment, he was a Postdoctoral Researcher in the Computer Science Department, Brandeis University and a Lecturer in the Mechanical Engineering Department, Massachusetts Institute of Technology (MIT), Cambridge. Before joining academia, he spent several years as a research engineer in the mechanical, electronic, and software industries. His interests are in the area of design automation and evolutionary robotics.