Action-Selection and Crossover Strategies for Self-Modeling Machines

Josh Bongard
Department of Computer Science
University of Vermont
33 Colchester Ave., Burlington VT 05405
josh.bongard@uvm.edu

ABSTRACT

In previous work [7] a computational framework was demonstrated that employs evolutionary algorithms to automatically model a given system. This is accomplished by alternating the evolution of models with the evolutionary search for new training data. Theory predicts [23] that the best new training data is that which induces maximum disagreement across the current model set. Here it is demonstrated that in a robot application this is not the case, and alternative fitness functions are developed that seek other, better training data. Also, it is shown that although crossover successfully reduces the mean error of the model set, it compromises the ability of the framework to find new, informative training data. This has implications for how to create adaptive, self-modeling machines, and suggests how competitive processes in the brain underlie the generation of intelligent behavior.

Categories and Subject Descriptors

I.2.9 [**Computing Methodologies**]: Artificial Intelligence—*Robotics*

General Terms

Algorithms

Keywords

Evolutionary robotics, self-modeling, artificial intelligence

1. INTRODUCTION

Industrial robots have permeated and revolutionized every aspect of heavy industry because they can execute preprogrammed actions in fixed, indoor industrial environments. Robots would be equally useful in outdoor or home environments, but creating devices that can continuously adapt and autonomously cope with the constantly changing aspects of such environments has had limited success. Rather than continuously having to re-program new controllers for a robot once it or its environment changes, evolutionary robotics [21] is a field that uses evolutionary computation to autonomously generate behaviors for robots. There are three main approaches to evolutionary robotics: controllers are either evolved directly on the physical device, requiring thousands of evaluations [10, 11]; controllers are adapted from an existing, hand-designed controller [24]; or a hand-designed simulator is used to evolve controllers in simulation before transferal to the physical device [14, 22]. The first approach is infeasible for continuous, rapid adaptation; the second approach requires a human to create the starting behavior; and the third approach requires a human to craft a simulation of the robot.

In previous work [6, 7] we introduced a fourth method that overcomes these obstacles by allowing the robot to evolve simulations of itself and its local surroundings, and then use the best of the evolved simulations to internally rehearse behaviors before attempting them in reality. Rather than most evolutionary computation-based modeling approaches in which a set of training data is generated first and then models are evolved to explain that data (eg. [18, 1, 12]), the framework developed in [6, 7] uses an active learning [2] approach: modeling alternates with a search for new training data, based on the current state of the models.

This raises the question of how to search for new training data. Seung *et al.* [23] showed that in theory, the optimal choice for the next training data is the one which causes the current set of models to disagree in their predictions. Once this training data is evaluated by the system being modeled and added to the training set, now only some of the models, not all, will agree with the results from the system, because the models disagree about the new training data. Further modeling can then replace these recently-revealed erroneous models with new models that explain all the old data, plus the new training data. Iterating this process is therefore shown to converge most rapidly on good models of the target system. However, finding such training data that induces maximum model disagreement is not trivial, if the space of possible training data is very large. In previous work we introduced the estimation-exploration algorithm [6], or EEA, which uses an evolutionary algorithm to search for these informative training samples: a fitness function rewards candidate training data for how much model disagreement it causes. A second evolutionary algorithm optimizes a set of models against the current set of training data. We have applied the EEA to problems in machine learning [4], gene network identification [9], damage localization in truss structures [17], and to robotics [7].

GECCO'07, July 7-11, 2007, London, England, United Kingdom

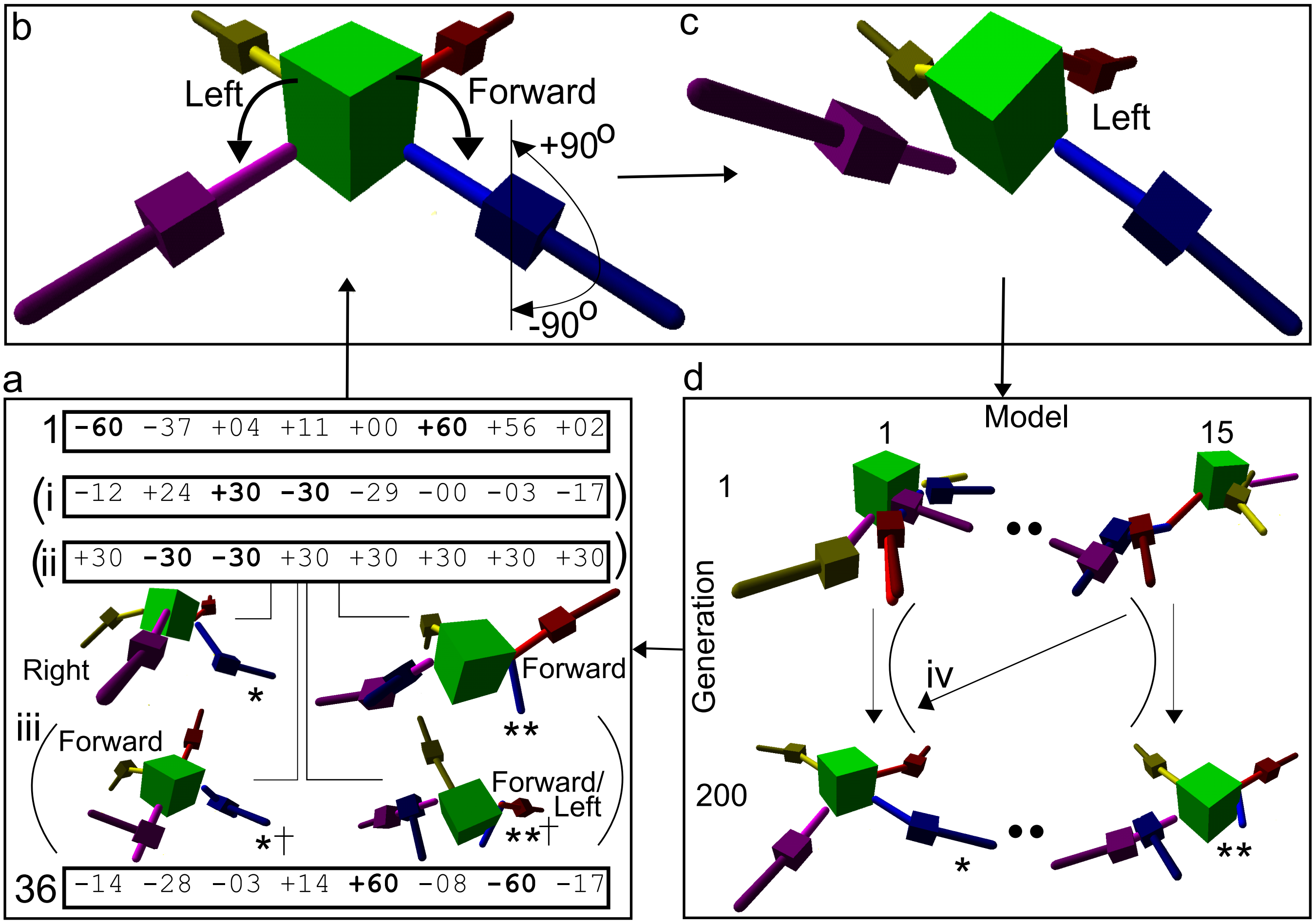

Figure 1: The EEA applied to a quadrupedal robot. The algorithm begins by supplying a random action (1. in a) to the target robot (b). The action causes the robot to move, and its main body tilts (c). The tilt information is bundled with the action that caused it, and supplied to the modeling phase. The robot meanwhile relaxes back to the default flat position. In the modeling phase (d), a set of 15 models are evolved to explain all the tilt information extracted from the target robot so far. These evolved models are then used to search for a new action (a). This process is repeated for a set number of cycles.

In this paper it is shown that, in practice, selecting training data to induce maximal model disagreement is *not* optimal. Instead it is demonstrated that a series of improvements lead the algorithm to find better training data, in a robotics application. In the next section I describe the basic algorithmic framework, as well as a set of algorithm variants that improve the modeling process. Finally, it is shown that introducing crossover into the model evolution phase, while reducing mean model error over the population, it also reduces the variation in the model population and therefore frustrates the algorithm's ability to find new, useful training data. I conclude with some general discussion of the repercussions of this finding for automated science (should computers seek new experiments to disprove hypotheses in the same way that human scientists do?) and human cognition (how do our brains model our bodies?).

2. METHOD

The EEA, applied to a robotics application, is outlined in Fig. 1. In this application the algorithm attempts to discover the correct topology of a simulated, four-legged robot, without direct information about how the robot's body parts are connected together. Previously [7] we showed that this framework allows a physical robot of the same topology to automatically construct a simulation of itself, use that simulation to internally rehearse behaviors, and automatically diagnose and recover from damage. The framework is composed of three components: the robot to be modeled (Fig. 1b,c), a set of self-models (Fig. 1d), and a set of candidate actions that can be executed on the robot (Fig. 1a).

2.1 The robot

The **target robot** used here is a simulated, four-legged robot (Fig. 1b). The robot is composed of nine body parts: the main body, and four upper and lower legs (Fig. 2a). The eight leg parts are motorized, and can rotate through the vertical plane that lies parallel to the leg. Positive rotations cause the part to rotate upward, and negative rotations downward. The robot is simulated within a three-dimensional, physics-based environment[1]. A training data in this application is an **action**: a set of eight desired mo-

[1] ode.org

tor rotations, expressed in degrees (Fig. 1a), that causes all eight motors to rotate, sending the robot to a fixed position (Fig. 1c). In this position, two tilt sensors record how much the robot's main body tilts to the left or right (-90^o to $+90^o$), and how much forward or backward (-90^o to $+90^o$). These two angles together form the **result** generated by the target robot. The robot then rotates the motors back to 0^o, causing the robot to lie flat again. The result is bundled with the action that caused them, and sent to the modeling phase (Fig 1d).

2.2 The models

In the modeling phase (Fig. 1d), a set of 15 **model robots** are evolved using a parallel hill climber to infer the way in which the robot's body parts are attached together. This setup has been used to a demonstration that a physical robot can autonomously synthesize a model of its own body and its immediate surroundings [8, 6, 7]. The training set is composed of the set of action/result pairs that have been obtained from the target robot so far. On the first cycle through the algorithm, the modeling phase has one pair; on the second cycle through it has two action/result pairs; and so on. The algorithm is assumed to know: how many motorized parts there are (eight); the mass and geometry of each part; parts are attached perpendicularly to each other; each body part is horizontal; and that actuating a body part with a positive angle will cause it rotate upward by that amount, and a negative angle downward by that amount. In future work these constraints will gradually be removed. The algorithm must indirectly infer how the parts are connected using only the tilt information in the training data.

A model is encoded as a 8×2 genotype G with floating-point values in the range $[0.0, 1.0]$, and dictates how the nine known body parts (Fig. 2a) should be connected to one another to produce a phenotype. The phenotype is a three-dimensional, physically realistic simulated robot, like those shown in Fig. 1d. The genotype is translated into a phenotype as follows. Each row in the matrix corresponds to one of the eight body parts. For each of the $i = 1, 2, \ldots, 8$ body parts, entry $G(i, 1)$ is scaled to an integer value in $[0, i-1]$. This indicates to which body part the current part attaches to. A value of 0 indicates the part attaches to the main body; a value of 1 indicates it attaches to motorized body part 1; and so on. In the example shown in Fig. 2b, part 1 attaches to part 0 ($G(1, 1) = 0$), and part 5 to part 1 ($G(5, 1) = 1$). The second value in the row, $G(i, 2)$, indicates where on the periphery of the parental body part the current part should be attached. A value of $G(i, 2) = 0$ indicates that part i should connect to the upper left of the parent body part; larger values attach the part at further positions around the periphery of the parent part, proceeding in a clockwise direction. In the example shown in Fig. 2b, body part 1 attaches to the upper-right of the main body (G(1,2)=0.25), and then body part 5 attaches in turn to the upper-right of body part 1 (G(5,2)=0.25).

Once a model robot is formed, it is actuated with each of the actions in the training data that have already been executed on the target robot. For each action, the resulting tilt of the model robot's main body is recorded. The **subjective error** of a model is then given by

$$m_e = \frac{\sum_{i=1}^{k}(|t_{lr}^{(i)} - m_{lr}^{(i)}| + |t_{fb}^{(i)} - m_{fb}^{(i)}|)}{2k}, \qquad (1)$$

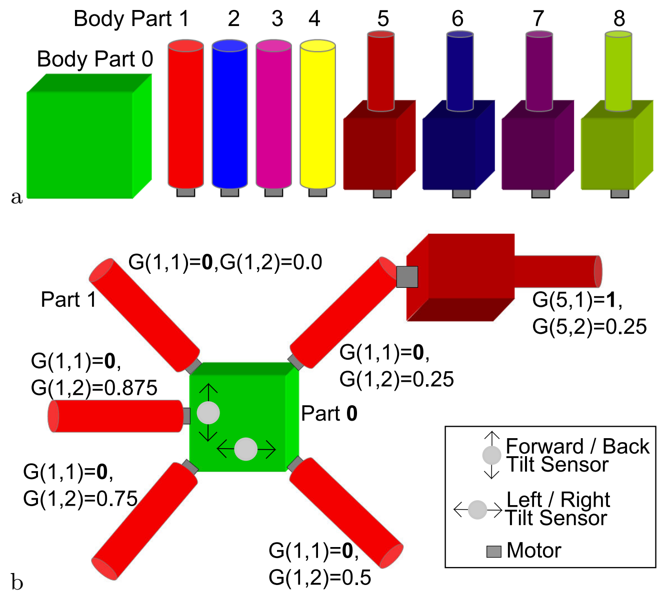

Figure 2: Genotype to phenotype translation for the self-models. The algorithm begins by knowing how many motorized parts the robot is composed of (a). A model genotype G encodes information for connecting the parts together. $G(i, 1)$ indicates to which body part i should attach. $G(i, 2)$ indicates where on the periphery it should attach.

where k is the total number of actions that have been performed by the target robot so far; $t_{lr}^{(i)}$ is the amount the target robot tilted to the left or right when it executed action i; $m_{lr}^{(i)}$ is the amount the model robot tilted to the left or right when it executed action i; $t_{fb}^{(i)}$ is the amount the target robot tilted forward or backward when it executed action i; and $m_{fb}^{(i)}$ is the amount the model robot tilted forward or backward when it executed action i. In short, the accuracy of a model is how well it reproduces the behaviors of the target robot when supplied with the same actions.

A hill climber then optimizes each of the 15 models in an attempt to minimize m_e. Once a model has been evaluated, its genotype is copied, a single value in the matrix is chosen at random, and Gaussian mutation is applied. A new model is created from this matrix, and evaluated. If the new model achieves a lower error than the parent model, the parent genotype is discarded; otherwise, the child is discarded. This process is continued for 200 generations, for each of the 15 models. These optimized models are then passed to the testing phase (Fig. 1a) for finding a new action. On the second and subsequent cycles through the modeling phase, hillclimbing begins with the best models from the previous cycle, but the models are re-evaluated against the larger training set, which contains the original action/result pairs plus the new pair just obtained from the target robot.

2.3 The actions

The testing phase attempts to find a new action that, when executed by the target robot, will provide more information about the robot's topology. At the outset of a run, 36 random actions are generated (Fig. 1a) but not yet executed on the target robot. Initially, one of the 36 actions is

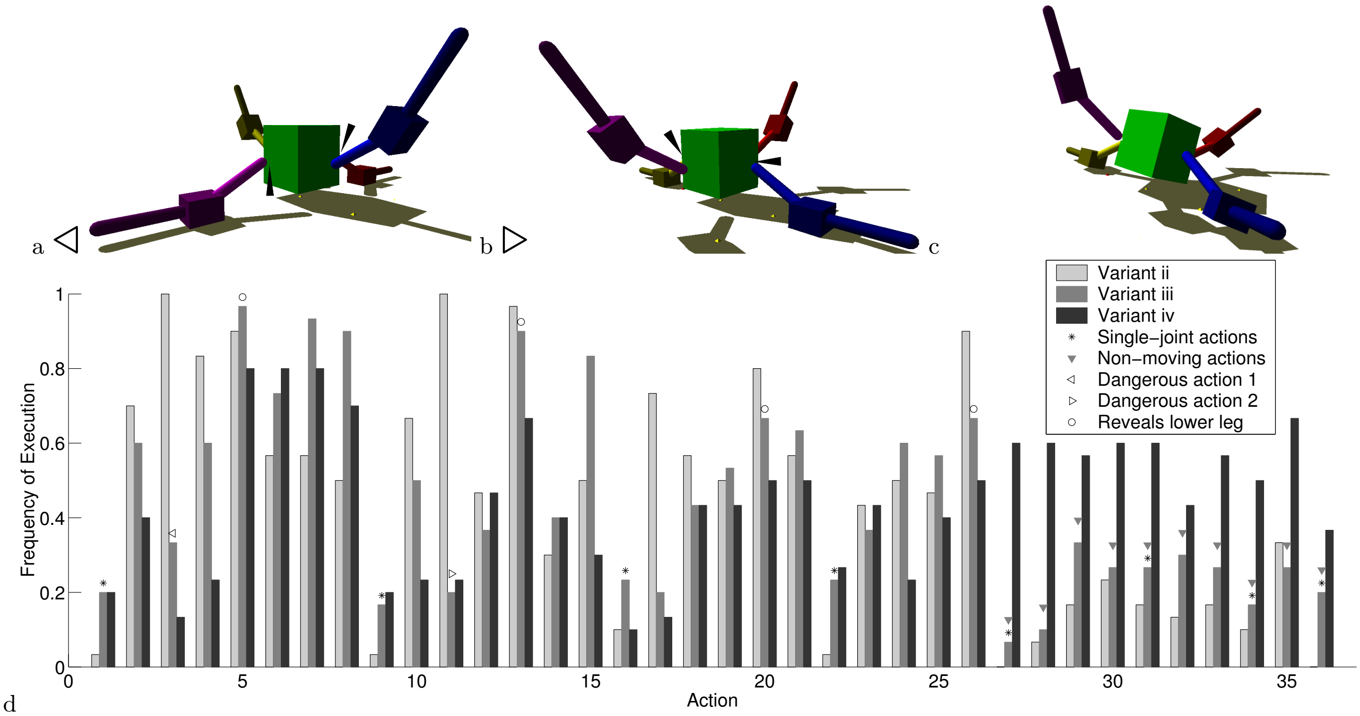

Figure 3: (a,b): Two 'dangerous' actions, performed by the target robot, which stays upright. Model robots that are morphologically similar to the target robot may fall to one side or the other (c), therefore incurring a drop in fitness. (d) The frequencies with which different algorithm variants chose actions to be sent to the target robot. Actions marked with an asterisk only rotate one part downward; all other actions rotate two parts downward. Actions marked with a downward-pointing triangle do not cause the main body to tilt at all. Actions marked with a left- and right-pointing triangle correspond to the dangerous actions shown in (a) and (b). Only the actions marked with a circle reveal information about the four lower legs.

sent to the target robot to generate the first training data. On the second and subsequent cycles, the optimized models from the modeling phase are used to determine which new action to send to the target robot for evaluation. Each action is supplied to the fifteen models, and the fitness of an action is computed using

$$a_f = \frac{\sigma^2(\mathbf{m}_{lr}) + \sigma^2(\mathbf{m}_{fb})}{2}, \tag{2}$$

where $\sigma^2(\mathbf{m}_{lr})$ is the variance across the left and right tilting of the 15 optimized models when supplied with action a, and $\sigma^2(\mathbf{m}_{fb})$ is the variance across the forward and backward tilting of the 15 optimized models. In short, the more that an action can cause the optimized models to tilt in different directions, the better it is.

Of the 36 actions, the one with maximal a_f, and which has not yet been executed is sent to the target robot for execution. It is then bundled with the resulting tilt information, and passed to the modeling phase, where the number of training data is incremented from k to $k+1$. Once the modeling phase has been cycled through 16 times, the algorithm halts, and returns the model with the lowest error as its best guess as to the topology of the target robot.

2.4 The algorithm variants

Initially it was found that even after 16 cycles, the algorithm was only able to infer the correct positions of a few of the body parts. In this paper a series of variants are introduced to the algorithm to perform its modeling ability. First, it was observed that the errors of the models increased from one cycle to the next (data not shown), indicating that the algorithm is being overwhelmed with information from the target robot: it cannot reduce the errors of the models against the current training data before modeling halts, and new training data is injected into the modeling process. In previous work [5] we found such algorithms must often be fed easier training data in order to make progress.

Variant i—In the first variant, we narrowed the range of angles that an action can rotate motors from 60 to 30 degrees (Fig. 1i).

Variant ii—In the second variant, the possible actions were further constrained. Actions were still limited to rotations in $[-30^o, 30^o]$, but instead of actuating all motors differently, an action could rotate only one or two motors downward, and all the others upward, thereby removing their influence on the main body's tilt (Fig. 1ii). This variant was created in light of the observation that human scientists, when investigating phenomena, only alter one or a few independent variables, and then observe the result; it is difficult to attribute phenomena to a particular variable if all variables are changed at once. This variant allows the algorithm to in effect investigate single or pairs of body parts at a time. This constraint reduces the possible number of actions to 36: 8 single joint rotations, and 18 double joint rotations.

Variant iii—During experimentation with variant ii, it was observed that two actions effectively halted further modeling. These actions are shown in Fig. 3a and b. These 'dan-

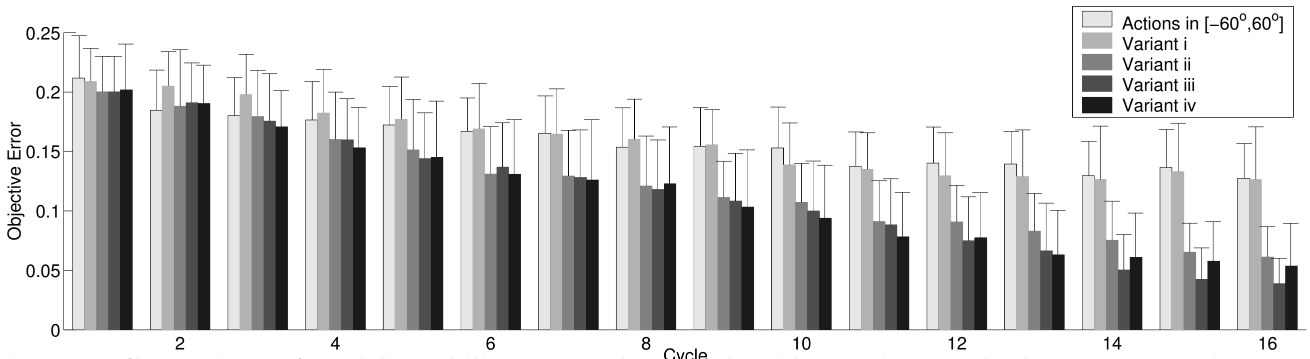

Figure 4: Comparison of modeling ability across the five algorithm variants. The bars correspond to mean objective error, averaged over the best models from 30 independent trials. The models used for averaging were those that had the lowest subjective error (Eqn. 1) at the end of each cycle through the modeling phase. Error bars indicate 1 unit of standard deviation.

gerous' actions rotate the two opposing lower legs downward, raising the main body off the ground. Any models that are very accurate, but not identical to the target robot will have slightly different mass distributions and will fall to one side or the other (Fig. 3c). This will cause a previously accurate model to suddenly experience an increase in error, as its behavior diverges from that of the target robot. Worse still, the testing phase tends to focus in on these actions, because near the end of the run, when the 15 models are all similar, and close to the target, they will continue to disagree—they will fall in different directions—about these two actions, so they tend to be sent to the target robot for evaluation.

These actions are known in dynamical systems as bifurcations: two similar models exhibit very different behavior when exposed to one of these actions. In the third variant, the fitness function in the testing phase was altered to steer selection away from such 'dangerous' actions. This was accomplished as follows.

A mutant of each of the 15 models used to evaluate actions is created by mutating their underlying genotypes as explained above. Two such mutations are shown in Fig. 1iii: model $*\dagger$ is a mutant of model $*$, and model $**\dagger$ is a mutant of model $**$. As before, all 15 models are actuated using the current action under consideration; now, these 15 mutants are also actuated with the current action. The fitness of an action a_f now is computed as

$$\frac{\sigma^2(\mathbf{m}_{lr}) + \sigma^2(\mathbf{m}_{fb})}{2} - \frac{\sum_{i=1}^{15} |m_{lr}^{(i)} - m_{lr}^{'(i)}| + |m_{fb}^{(i)} - m_{fb}^{'(i)}|}{30}, \quad (3)$$

where the first term, as before, awards actions for maximizing disagreement across the 15 models. The second term penalizes actions for causing disagreement between model i ($m_{lr/fb}^{(i)}$) and its mutant, $m_{lr/fb}^{'(i)}$. For example, in Fig. 1aiii, a given action causes models $*$ and $**$ to disagree: the former falls to the right, and the latter falls forward. However, the same action causes their mutants to disagree: model $*\dagger$ falls forward, rather than right, and model $**\dagger$ falls forward and left, rather than forward. By penalizing actions in this way, the algorithm selects against potentially bifurcating actions: if two similar models m and m' exhibit different behavior, and this is supported by many of the models ($m^{(}1)$ to $m^{(}15)$), it is likely that the current action will stall the modeling process. In short, this fitness function selects both for informativeness (the first term) and reliability (the second term).

Variant iv—In the fourth variant, variant iii is retained, but crossover is added to the parallel hillclimber in the modeling phase (Fig. 1div). Crossover is conducted as follows. Whenever a new child model is to be created, the genotype matrix of its parent is copied and mutated as explained above. With a 50% probability, another parent model is selected as a donor. Each row of the donor matrix is copied over the same row in the new genotype matrix, with a 50% probability. This effectively combines data from two models into the new model.

3. RESULTS

For each of the five algorithms described above (the basic algorithm, plus variants i-iv), 30 independent trials were conducted, yielding a total of 150 trials. Each trial begins with a random action, and then cycles through the algorithm 16 times. At the end of each cycle through the modeling phase, the model with the lowest error is output, providing a series of 16 increasingly accurate models.

Within the algorithm, model error is calculated using data obtained from the target robot, but this is **subjective error**, from the point of view of the model: it only indicates how well the model fits a subset of information. A second error metric, known as **objective error**, calculates how similar the topologies of the model and target robot are. This is calculated using

$$m_o = \frac{\sum_{i=1}^{8} \sqrt{(t_x^{(i)} - m_x^{(i)})^2 + (t_z^{(i)} - m_z^{(i)})^2}}{8},$$

where $t_x^{(i)}$ is the horizontal position of body part i on the target robot when it lies flat, $m_x^{(i)}$ is the horizontal position of body part i on the model robot, $t_z^{(i)}$ is the z-position of body part i on the target robot, and $m_z^{(i)}$ is the z-position of body part i on the model robot. Therefore, the objective error of a model is the mean Euclidean difference between the positions of the target robot's body parts, and the model robot's body parts. This metric then gives an unbiased indication of how close the model is to the topology of the target robot.

Fig. 4 reports the mean objective errors of the model robots from the five algorithms, arranged from the best model produced at the end of the first cycle, to the best model produced by the last cycle. As can be seen, in general the objective error of the models decreases as the trials

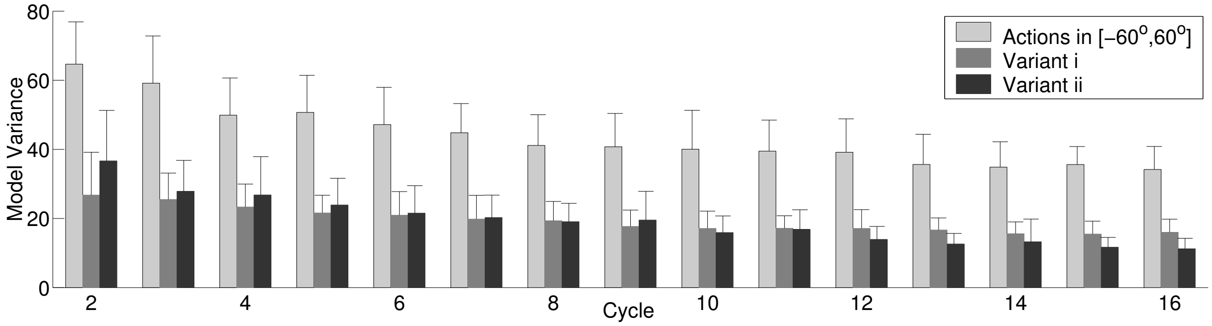

Figure 5: The mean model variance induced by the actions sent to the target robot (Eqn. 2), for three algorithm variants. During the first cycle through the testing phase, there are no models available from the modeling phase for evaluating actions, so an action is chosen at random and therefore its model variance cannot be calculated. Model variances are therefore plotted for testing cycles 2 to 16.

proceed, indicating that some improvement in the models is achieved. However, the base algorithm and variant i make relatively little progress, compared to variant ii, when only one or a pair of body parts are allowed to rotate downward. This indicates that, like a human scientist, the computer is better able to model the system in question when it perturbs one or at most two variables at once, rather than changing them all in parallel.

However, the actions produced by variant ii produce less model disagreement than the actions produced by the base algorithm and variant i. Fig. 5 reports the mean model prediction disagreements (Eqn. 2) caused by the 15 actions sent to the target robot by the base algorithm, variant i and variant ii. Not surprisingly, the base algorithm can induce more disagreement, because it can tilt the main body further when motors can rotate between $[-60^o, 60^o]$ than when they can only rotate between $[-30^o, 30^o]$. Further, restricting the actions to only rotate one or two motors downward (variant ii) yields actions that produce about the same amount of model variance as when they can all be rotated downward (variant i). In late cycles it can be seen that variant ii achieves less model variance than variant i, but this is because the models during these later cycles are more accurate than those from variant i, and therefore more similar to each other. This finding contradicts the theoretical conclusion of [23]: it is shown here that greater model disagreement does not necessarily imply accelerated modeling.

Fig. 4 indicates that from cycle 14 on, variant iii outperforms the previous three algorithms, indicating that choosing actions not simply based on their informativeness (i.e., model variance) but also on their reliability (Eqn. 3) improves modeling. This finding is supported by Fig. 3d, which reports the frequency with which different algorithm variants selected actions to send to the target robot. As can be seen, actions 3 and 11 correspond to the 'dangerous' actions shown in Figs. 3a and b, and were sent by algorithm variant ii to the target robot at some point during all 30 runs (frequency of appearance = 1). This is not surprising, as these actions can best induce disagreement among models, even during later stages of the run when the models are similar to the target robot, and to one another. However, variant iii sent these actions to the target robot very rarely (frequencies of appearance = 0.33 and 0.2 for actions 3 and 11), because these actions were heavily penalized for being unreliable (the second term in Eqn. 3).

Other biases in selecting actions are revealed by Fig. 3d. First, those actions that rotate both the upper and lower parts of the same leg (indicated by a circle) were selected very often. These actions are the only ones that reveal information about the four lower legs. When only one or two motors may be rotated downward (variant iii), if only the lower leg is rotated downward but the upper leg is rotated upward, the lower leg does not touch the ground and cannot be inferred. The fact that the algorithms usually propose these actions to the target robot indicates that the models are becoming more accurate: the models predict that exactly these pairs of downward motor rotations will reveal where body parts 5 to 8—the lower legs (Fig. 2a)—are located. Similarly, the models predict that certain actions (actions 27 to 36) do not cause the main body to tilt at all, and are therefore not very informative, and should not be executed by the target robot.

Finally, Fig. 4 reports the effect on modeling if crossover is introduced into the modeling phase (variant iv). Surprisingly, this has an adverse effect on modeling: the mean objective errors of models in cycles 14, 15 and 16 are generally higher than variant iii, which is the same as variant iv except crossover is not employed. Fig. 6 reports the average quality of the worst models from algorithm variants iii and iv. The worst model is defined as the model at the end of each cycle through the modeling phase that has the highest subjective error (Eqn. 1). Clearly, crossover reduces the average subjective error across the 15 models, as the worst model has a significantly lower subjective error when crossover is used (variant iv), than when it is not (variant iii). Therefore, crossover is successful in the sense that it is transferring useful genetic material from better models to worse models. Therefore, the adverse effect of crossover as seen in Fig. 4 cannot be due to crossover disrupting model improvement during the modeling phase.

An alternative explanation is illustrated in Fig. 7. When the algorithm lacks crossover, and the models are almost correct during the later cycles of a run, the models will still disagree about the location of the few body parts (Figs. 7a and b) for which they have not obtained any evidence (such as part 8 in Fig. 7). Due to lack of evidence, that body part may reside anywhere on the robot's body. The testing phase will converge on an action that moves body part 8, because its different locations on the models will cause them to tilt in different directions when that body part is actuated (Fig. 7a,b). However, when crossover is employed, the more accurate model (such as model 1 in Fig. 7c) will contribute genetic material to the lesser accurate models (such as model 2 in Fig. 7d), including position information

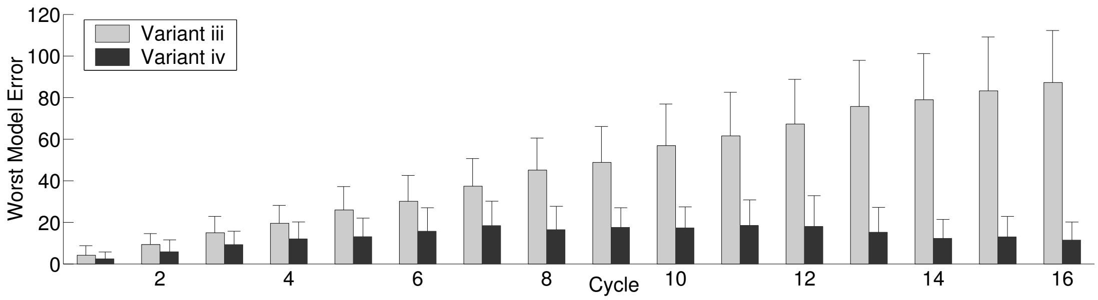

Figure 6: A comparison of mean error of the worst models produced by the algorithm variant without crossover (variant iii) and with crossover (variant iv). The gradual rise in error observed in variant iii reflects the increasing failure of the worst models to explain the new training data being accumulated by the algorithm. Crossover negates this effect by transferring genetic information from more accurate models into these less accurate models.

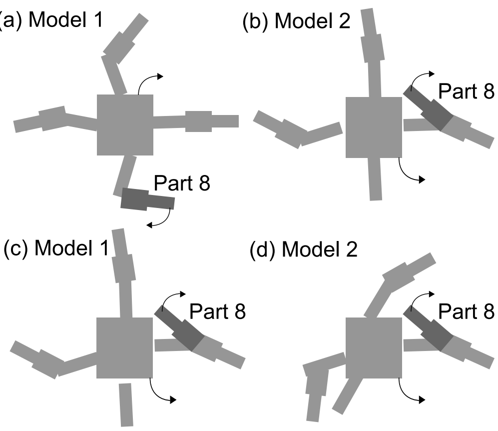

Figure 7: An example illustrating why crossover compromises the ability of the algorithm to model the target robot. In models from the algorithm lacking crossover (a,b), body parts for which no information has been gathered (i.e. part 8) appear at different locations on the models. Actions are then selected and executed by the target robot that actuate these parts, because they cause the models to tilt in different directions (indicated by arrows). When crossover is employed, the models agree about the locations of poorly modeled body parts (c,d), and actions do not target the actuation of them.

for body parts yet to be inferred. Therefore models will agree not just about body parts that have been inferred, but also those that have not yet been inferred. Because of this, actions that rotate non-inferred body parts such as body part 8 in Fig. 7 will not induce any more model disagreement than actions that rotate well understood parts (Fig. 7c,d). Therefore crossover frustrates the ability of the testing phase to locate new, informative tests, and modeling suffers as a result. In other words, information about the remaining, poorly-modeled parts of the target robot cannot be extracted. This line of reasoning is supported by Fig. 3, which indicates that the only four actions which reveal information about the lower legs (marked with circles) are executed less often when crossover is employed (variant iv) than when it is not (variants ii and iii).

4. DISCUSSION AND CONCLUSION

The computational framework discussed here incorporates active learning research [2] which has shown that it is better to alternate modeling with the collection of new training data, rather than simply generating training data at random before modeling commences. Seung *et. al* [23] derived the optimal method for selecting new training data that should be labeled by the system: the most informative training data is that which induces the most disagreement among the predictions of the current model set.

The framework previously developed, the Estimation - Exploration Algorithm (EEA) [9, 8, 3, 6, 4, 5, 7], employs evolutionary computation to optimize a set of models using the training data collected from the system under study so far, and also to search for new training data, once the model set has been optimized. In this paper several variants of this framework were explored, when applied to a particular system: a legged robot, able to execute actions (the training data) and observe its own resulting behavior (the 'labeling' of that training data). More specifically, it was shown that simply seeking actions that induce the most model disagreement does not, as theory predicts [23], accelerate modeling. Rather, constraining the space of possible training data such that only one or a pair of variables of the system are perturbed at once improves the ability of the algorithm to model the system under study. This agrees with the way in which human scientists investigate phenomena. By only perturbing one or a few variables at a time, and observing any resulting change in the system, the experimenter can draw causal relationships between different parts of the system. Developing automated systems that can not only model physical systems, but also propose new experiments to improve those models is just now becoming feasible [16], and it has been shown that evolutionary computation has an important role to play in these endeavors [7].

In addition to the kinds of actions that should be used in such a process, it has been shown here that a fitness function that only awards training data for how much model disagreement it induces is insufficient. We demonstrate that a more sophisticated fitness function that awards for both model disagreement and reliability is superior. Reliability is measured by the ability of the proposed action to avoid bifurcation, as predicted by the current model set. The EEA is a type of coevolutionary algorithm [13, 15], in the sense that the relative fitness between two models changes when a new piece of data from the target system is introduced; in effect, the new training data alters the fitness landscape for the models. Actions that induce bifurcations (accurate models behave very differently from the target system, when supplied with the same action) cause the space of possible models to become rugged: highly fit models perform differently than the target system, and therefore obtain a low fitness. In effect, the fitness function described here ensures that the fitness landscape for the models remains smooth over the course of the run, thereby accelerating modeling.

Finally, it was shown that even when crossover is introduced and is functional in the sense that the overall fitness of the model population decreases, it compromises the EEA's ability to model the system. This is due to the fact that the dormant variability in the model population—those parts of the target system that have not been perturbed are modeled in different ways by the different models—dissipates, and the algorithm cannot find training data that perturbs those unseen parts of the system.

In the same way that evolutionary algorithms rely on the latent variation in populations of candidate solutions, the EEA is able to better model target systems by relying on model populations, rather than a single model. This has wider implications not just for system identification [20], but for neuroscience. Evidence is emerging that suggests competition between competitive processes in the brain may be the underpinning of intelligent behavior [19]. The work presented here, which enables an adaptive robot to model itself, may provide the first few steps in a systematic method for exploring this new way of looking at cognition.

5. REFERENCES

[1] H. Andrew. System identification using genetic programming. In *Proceedings of the Second Intl. Conf. on Adaptive Computing in Engineering Design and Control*, pages 57–62, 1996.

[2] Y. Baram, R. El-Yaniv, and K. Luz. Online choice of active learning algorithms. *Journal of Machine Learning Research*, 5:255–291, 2004.

[3] J. Bongard and H. Lipson. Once more unto the breach: Co-evolving a robot and its simulator. In *Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems (ALIFE9)*, pages 57–62, 2004.

[4] J. Bongard and H. Lipson. Active coevolutionary learning of deterministic finite automata. *Journal of Machine Learning Research*, 6(Oct):1651–1678, 2005.

[5] J. Bongard and H. Lipson. 'Managed challenge' alleviates disengagement in co-evolutionary system identification. In *Proceedings of the 2005 Genetic and Evolutionary Computation Conference*, pages 531–538, Washington DC, 2005.

[6] J. Bongard and H. Lipson. Nonlinear system identification using coevolution of models and tests. *IEEE Transactions on Evolutionary Computation*, 9(4):361–384, 2005.

[7] J. Bongard, V. Zykov, and H. Lipson. Resilient machines through continuous self-modeling. *Science*, 314:1118–1121, 2006.

[8] J. C. Bongard and H. Lipson. Automated robot function recovery after unanticipated failure or environmental change using a minimum of hardware trials. In *Proceedings of The 2004 NASA/DoD Conference on Evolvable Hardware*, pages 169–176, Seattle, WA, 2004.

[9] J. C. Bongard and H. Lipson. Automating genetic network inference with minimal physical experimentation using coevolution. In *Proceedings of The 2004 Genetic and Evolutionary Computation Conference*, Seattle, WA, 2004.

[10] D. Cliff, P. Husbands, and I. Harvey. Evolving visually guided robots. In J.-A. Meyer, H. Roitblat, and S. Wilson, editors, *Proceedings of the Second International Conference on the Simulation of Adaptive Behaviour*, Boston, MA, 1993. MIT Press.

[11] D. Floreano and F. Mondada. Hardware solutions for evolutionary robotics. In P. Husbands and J.-A. Meyer, editors, *EvoRobots*, pages 137–151, 1998.

[12] G. Gray, D. Murray-Smith, Y. Li, K. Sharman, and T. Weinbrenner. Nonlinear model structure identification using genetic programming. *Control Engineering Practice*, 6:1341–1352, 1998.

[13] W. D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D*, 42:228–234, 1990.

[14] N. Jakobi. Evolutionary robotics and the radical envelope of noise hypothesis. *Adaptive Behavior*, 6(1):131–174, 1997.

[15] E. D. Jong and J. Pollack. Ideal evaluation from coevolution. *Evolutionary Computation*, 12(2):159–192, 2004.

[16] R. D. King, K. E. Whelan, F. M. Jones, P. G. K. Reiser, C. H. Bryant, S. H. Muggleton, D. B. Kell, and S. G. Oliver. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427:247–252, 2004.

[17] B. Kouchmeshky, W. Aquino, H. Lipson, and J. C. Bongard. Coevolutionary strategy for structural damage identification using minimal physical testing. *International Journal for Numerical Methods in Engineering*, 69(5):1085–1107, 2006.

[18] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Boston, MA, 1992.

[19] A. Livnat and N. Pippenger. An optimal brain can be composed of conflicting agents. *PNAS*, 103(9):3198–3202, 2006.

[20] L. Ljung. *System Identification: Theory for the User*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1999.

[21] S. Nolfi and D. Floreano. *Evolutionary Robotics*. MIT Press, Boston, MA, 2000.

[22] J. B. Pollack, H. Lipson, S. Ficici, P. Funes, G. Hornby, and R. Watson. Evolutionary techniques in physical robotics. In J. Miller, editor, *Evolvable Systems: from biology to hardware*, pages 175–186. Springer-Verlag, 2000.

[23] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Workshop on Computational Learning Theory*, pages 287–294, New York: ACM Press, 1992.

[24] R. Tedrake, T. Zhang, and H. Seung. Learning to walk in 20 minutes. In *Proceedings of the Fourteenth Yale Workshop on Adaptive and Learning Systems*, Yale University, New Haven, CT, 2005.