Combining Fitness-based Search and User Modeling in Evolutionary Robotics

Josh C. Bongard
Dept. of Computer Science
University of Vermont
josh.bongard@uvm.edu

Gregory S. Hornby
Univ. of California Santa Cruz
NASA Ames Research Center
gregory.s.hornby@nasa.gov

ABSTRACT

Methodologies are emerging in many branches of computer science that demonstrate how human users and automated algorithms can collaborate on a problem such that their combined solutions outperform those produced by either humans or algorithms alone. The problem of behavior optimization in robotics seems particularly well-suited for this approach because humans have intuitions about how animals—and thus robots—should and should not behave, and can visually detect non-optimal behaviors that are trapped in local optima. Here we introduce a multiobjective approach in which a surrogate user (which stands in for a human user) deflects search away from local optima and a traditional fitness function eventually leads search toward the global optimum. We show that this approach produces superior solutions for a deceptive robotics problem compared to a similar search method that is guided by just a surrogate user or just a fitness function.

Categories and Subject Descriptors

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

General Terms

Experimentation, Algorithms

Keywords

Evolutionary Robotics, Interactive Evolutionary Algorithms, Evolutionary Algorithms

1. INTRODUCTION

One of the original goals of computer science in general, and Artificial Intelligence in particular, was complete automation: once a problem is formulated in enough detail an AI algorithm should be able to automatically generate a solution. Recently however there has been a turn toward algorithms that combine human users and automated algorithms such that the two complement one another. Humans supply intuition, provide 'outside of the box' ideas or recruit additional humans through social networks; computers provide brute force search within the space delineated by the human team participants. One of the first examples of so-called human-based computation include the ESP game in which images are tagged with appropriate text descriptions generated through indirect human corroboration [24]. In another example, video game players fold simulated proteins into appropriate configurations and often solve computationally-intractable protein folding problems [25]. Many such human-based computation algorithms recruit, filter and combine contributions from large numbers of users; thus crowdsourcing [9] can be seen as one form of human-based computation.

Arguably, the first class of human-based computer algorithms is the Interactive Evolutionary Algorithm (IEA). This technique was pioneered by Richard Dawkins: a set of computer-generated images resembling insects are shown to a user; the user selects a subset that they prefer; and the non-selected images are replaced by mutated and/or crossed versions of the preferred images [7]. Several examples of IEAs followed (e.g. [22], [8]). An interactive evolutionary robotics method was also proposed [10] in which the fitness of a given robot control policy was determined by hand.

However, all interactive evolutionary algorithms that employ a human instead of a fitness function suffer from one critical drawback: user fatigue. In practice, such methods require the user to supply hundreds or thousands of preferences to produce truly novel solutions, or solutions that compete with those produced by automated, fitness-function drive evolutionary algorithms.

To address this drawback, a new class of interactive evolutionary algorithms have appeared which build a user model. Such algorithms follow a four-step procedure: (1) the user is presented with several candidate solutions; (2) the user's preferences or rankings are collected; (3) a model is trained such that it takes as input features of a candidate solution and produces as output a prediction for how much the user will like that solution. Once trained, (4) the user model can then stand in for the human user. In [2], user feedback was input to a parameter estimation system. Although this leverages statistical machine learning methods, it remains limited to parameterized design spaces. An alternative method was not constrained to parameterized design spaces: user feedback was used to learn weights on grammatical rules for constructing designs [5]. However if designs are

GECCO'13, July 6–10, 2013, Amsterdam, The Netherlands.
ACM 978-1-4503-1963-8/13/07...$15.00..

described by a large number of rules, an intractable amount of user feedback is required to learn the requisite number of weights. In this work we search a parameterized design space, but little change to the proposed method would be required to expand it for searching open-ended design spaces.

Schmidt and Lipson [18] employed neural networks to model a user who indicated preferences for certain line drawings over others. Hornby and Bongard [12] extended this method and used it to show that such a user model could successfully 'stand in' for a human user: employing a user model improved an IEA such that it was 2.5 times faster and 15 times more reliable. Elsewhere, Hornby and Bongard [11] compared two ways of modeling users. In both cases the user is shown a grid of candidate solutions and forced to choose their favorites. In the first, the user model is supplied with pairs of candidate solutions and is trained to reproduce the user's preferences. In the second, scores are computed for all of the candidate solutions shown to the user: if the user preferred solution A over solution B and solution B over solution C, then score(A) > score(B) > score(C). Then, the user model is supplied with only one candidate solution and the model is trained to reproduce the score of that solution. This latter approach is employed in this paper. Outside of evolutionary computation, learning user models from preferences has also been explored in the domain of reinforcement learning [6].

Recently, user modeling has been applied to robotics [1]. The method, Preference-based Policy Learning, trains a user model to reproduce a user-generated ranking of robot control policies. The model takes as input a compression of the sensor/motor time series generated by the robot using a given control policy and must produce as output a successful prediction of that policy's user-generated rank.

However, in all of these methods that construct and employ a user model, the model replaces the fitness function, much like a human replaces the fitness function in Interactive Evolutionary Algorithms. This belies an assumption underlying both user-model-based methods and Interactive Evolutionary Algorithms: fitness functions are deceptive[1]. The fitness function is also dispensed with in novelty search [14] in which the quality of a candidate solution is determined to be its distance from previous solutions. However, Akrour et al [1] showed that a user-model-based method can outperform novelty search because the latter algorithm spends much effort in exploring very poor solutions.

Here we hypothesize that a user model and fitness function together can guide search better than either on its own. If only a user model is employed, the user must teach it to guide search away from local optima and toward the global optimum. If both are employed, the user model can guide search away from local optima but the fitness function can then automatically 'take over' and guide search toward the global optimum without further input from the user. The results we present here support our hypothesis: If a user model complements rather than replaces a fitness function for an evolutionary robotics task, better solutions can be found than either the fitness function guiding search alone or the user model guiding search alone.

The next section describes the robot task and these three algorithm variants. Section 3 presents the results from these variants and sections 4 and 5 provide some analysis, discussion and concluding remarks.

[1]If the problem is not deceptive, then user influence during search is not required.

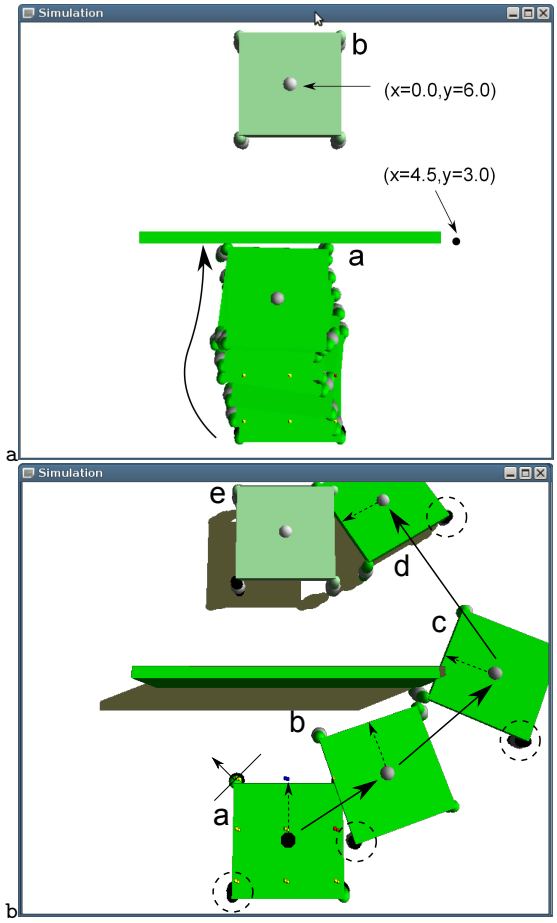


Figure 1: The robot, its task environment and two evolved behaviors. When only a fitness function is employed (a), control policies evolve that walk the robot toward the barrier and become stuck there (a.a) rather than moving around the barrier and reaching the goal position (a.b). If a user model and a fitness function are used (b), control policies evolve that successfully guide the robot around the barrier (b.a-e). Videos of the robots can be viewed at `bit.ly/127JNLM`.

2. METHODS

Here we describe three algorithm variants that generate control policies for a quadrupedal robot such that the robot travels around a rectangular barrier and reaches a target object located on the other side (Fig. 1). The first variant uses a fitness function that rewards control policies that minimize the distance from the robot to the target object (Sect. 2.1). The second variant learns a model of a human user who prefers control policies that enable the robot to reach the right edge of the barrier and then approach the target object (Sect. 2.2). The third variant uses both the fitness function and a user model to guide the robot to the target object (2.3).

2.1 Fitness-based Search (FS)

In the traditional evolutionary robotics paradigm, a fitness function is designed and used to perform differential survival and reproduction within a population of simulated or physical robots [16]. To investigate the relationship between user modeling and fitness-based search we evolve a population of control policies that are rewarded for guiding a simulated quadrupedal robot around a barrier to reach a target object on the far side.

2.1.1 The robot

The robot is composed of four, one degree-of-freedom rotational joints that attach each leg to the main body[2]. The axes of the four joints are set such that the leg sweeps through the plane diagonal to the main body (illustrated by the arrow-and-line in Fig. 1.b.a). This provides the robot with a holonomic drive system: the robot can move in any direction regardless of its orientation. More importantly, the robot can describe a curved trajectory around the barrier (Fig. 1b) without having to turn in that direction initially. Each joint is actuated with a motor that can rotate the joint through $[-45^o, +45^o]$.

Each foot contains a binary tactile sensor. The robot is also equipped with five photosensors, one in each component of its body. The target object on the far side of the barrier emits a light; occlusion is not modeled in these experiments, so it is assumed that the photosensors can detect the light source even if the barrier lies between the object and the sensor. Photosensors return a value between zero for complete darkness and one for maximal brightness. The robot also has a compass sensor that returns a value also in $[0, 1]$: zero when facing forward (indicated by the dotted arrow in Fig. 1a.a); 0.5 when facing backward; and 0.75 when facing to the right.

2.1.2 The control policy

A feedforward artificial neural network with no hidden layer is used to control the robot: each of the four tactile sensors, five photosensors and one compass sensor connect to each of the four motors, yielding 40 synaptic connections to optimize.

2.1.3 The evolutionary algorithm

The light green robot in Fig. 1 indicates the desired final state of the robot: this robot is standing over the target object (unseen in the figure). This generates strong signals in the five photosensors embedded in this final-state robot, denoted as $s_1^{(r)}$, ..., $s_5^{(r)}$. The fitness of a control policy (which is to be maximized) is calculated as

$$f = \frac{1}{1 + (\sum_{i=1}^{5} \sum_{t=1}^{T} |s_i^{(t)} - s_i^{(r)}|)/5T} \quad (1)$$

where $T = 1000$ is the total number of time steps that each control policy is evaluated for; and $s_i^{(t)}$ denotes the value of the ith photosensor at time step t.

We have shown in past work [3] that this characterization allows for the evolution of different behaviors without having to rewrite the fitness function for each new behavior. For example if brachiation is desired, the robot can be started at one end of a set of suspended rungs and the target object (along with the end-state robot) placed at the other end. If stair climbing is desired, the robot can be placed at the base of the stairs and the target object and end-state robot placed at the top. By creating an interactive 3D environment in which a human user can interactively construct the task environment of the robot, another form of interactive evolutionary robotics was achieved: users can select for different behaviors by altering the task environment rather than the fitness function.

In the present work, the fitness function was then incorporated as one objective in a bi-objective optimization method, Age-Fitness Pareto Optimization (AFPO) [20]:

$$\text{obj}_1 = f \quad (2)$$

$$\text{obj}_2 = age \quad (3)$$

where *age* is defined as the age since the control policy—or its oldest ancestor—was introduced into the evolving population [13]. In the first generation each policy has an age of 1. Any policies that survive into the next generation, or their offspring, have an age of 2, and so on. Policies that lie on the Pareto front produce offspring that replace dominated policies. At each generation a new, random policy with an age of 1 is injected into the population. AFPO protects young yet promising solutions long enough for some of them to evolve offspring that push older solutions out of the population.

Fig. 2 illustrates how the three algorithm variants differ from one another. In Fitness-based Search (FS), an initial population of randomly-generated control polices are created (Fig. 2a). Since no user preferences are ever requested from the user (Fig. 2b) and thus no preferences are ever received from the user (Fig. 2c), the objectives from Eqns. 2 and 3 are employed to evolve the control policies (Fig. 2d).

2.2 Preference-based Policy Learning (PPL)

Preference-based Policy Learning (PPL) [1] involves four steps: the robot demonstrates a number of behaviors; the user ranks the control policies that generated those behaviors; a model is learned until it takes as input the compressed sensor/motor time series of each policy and returns the rank of each policy; and finally the robot uses this model to search for policies that obtain ever higher ranking scores from the model.

PPL is here adapted to the robot and task at hand. First, a population of randomly-generated policies are created (Fig. 2a). Two policies are chosen at random (Fig. 2b) and shown to the user (Fig. 2e). Once the user indicates a preference (Fig. 2f), the preference is collected (Fig. 2h) and a ranking of the policies is computed. This is accomplished by creating an $n \times n$ matrix $\mathbf{P}$, where n is the number of policies that the user has supplied a preference for. Element $p_{ij} = +1$ if the user preferred policy i over policy j and $p_{ij} = -1$ if she preferred j over i. Element $p_{ii} = 0$ for each policy i.

The score of each policy is then stored in the vector $\mathbf{c}$. Each element in $\mathbf{c}$ is computed using

$$c_i = \frac{\text{rowsum}(\mathbf{P}, i) - \text{minrowsum}(\mathbf{P})}{\text{maxrowsum}(\mathbf{P}) - \text{minrowsum}(\mathbf{P})} \quad (4)$$

where $\text{rowsum}(\mathbf{P}, i)$ is the sum of the ith row in $\mathbf{P}$, $\text{minrowsum}(\mathbf{P})$ is the minimum row sum in $\mathbf{P}$ and $\text{maxrowsum}(\mathbf{P})$ is the maximum row sum in $\mathbf{P}$. This formulation ensures

[2]The robots were evolved within the Open Dynamics Engine physics simulator.

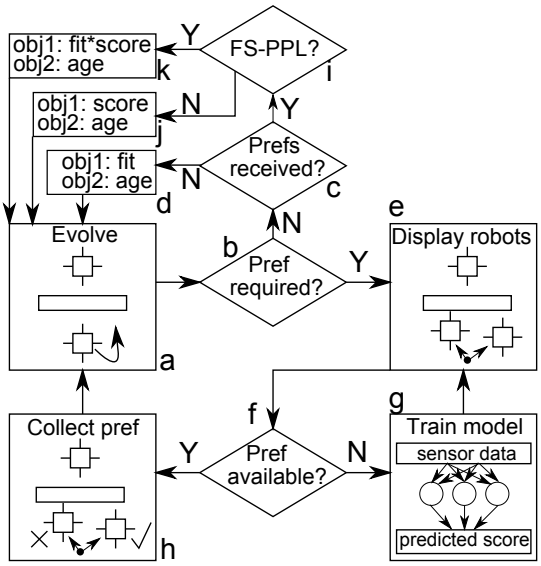


Figure 2: Program flow for the three algorithm variants. For Fitness-based Search, evolution uses only the fitness function and age (a-d). For Preference-based Policy Search, evolution (a) periodically requests (b) and receives (e,f,h) a preference from the user. While the user observes a robot pair (e), the user model is improved (f,g). When several preferences have been collected (c) user model-generated scores and age are used (j) during evolution (a). Fitness-based Search and Preference-based Policy Search (FS-PPS) follows the same flow as PPS but uses the fitness function, the user model and age (k) to evaluate control policies.

that the lowest-ranking policy has a score of 0 and the highest-ranking policy a score of 1.

A single policy is now drawn from the policy population (Fig. 2a) and shown alongside one of the original two policies shown to the user (Fig. 2e). While the algorithm waits for the user to indicate a preference, training of the user model can now commence.

2.2.1 User Model Training

In the PPL variant implemented here, the user model takes the form of an artificial neural network. Sensor data generated by a control policy is supplied at the input layer and the value arriving at the single output neuron is treated as the model's prediction of that policy's rank.

The input layer is composed of 12 neurons. The first six neurons report the values of the five photosensors (s_1-s_5) and one compass sensor (s_6) halfway through the evaluation period ($s_1^{(T/2)} \ldots s_6^{(T/2)}$). The second six neurons report the values of the five photosensors and one compass sensor at the end of the evaluation period ($s_1^{(T)} \ldots s_6^{(T)}$).

The original PPL formulation reported in [1] supplied a compression of the entire sensor-motor time series to the user model. Here, only a subset of the sensor data is supplied to the user model. Admittedly this subset was chosen because it is clear that for this task the robot should move to the right of the barrier about halfway through the evaluation period (Fig. 1b.c) and then to another position at the end of the evaluation period (Fig. 1b.e). Thus, this particular subset of sensor data may not allow the user model to learn a user's preferences if they prefer some other behavior. However, this issue of what aspect of the robot's behavior to present to the user model has been discussed elsewhere. Although important and in need of investigation, this issue is outside the scope of the present work.

While the user is considering which of two robots to prefer (Fig. 2e), the user model is trained using backpropagation [17]. For each policy i out of the n policies that the user has indicated a preference for so far (at the outset this is $n = 2$), the sensor data from policy i is presented at the input layer and the predicted score is collected from the output neuron. The error between this predicted score and the score stored at $\mathbf{c}_i$ is computed and back-propagated. Backpropagation continues to iterate across the n policies until the user supplies a new preference (Fig. 2f). (This allows the model to capture a user's strategy regardless of whether her preferences are transitive.)

Once the user does supply a preference (Fig. 2f,h), element p_{ik} and p_{ki} can be filled in the preference matrix $\mathbf{P}$, where i is one of the original two control policies and k is the policy newly-drawn from the population. So the user is now shown two robots controlled by policies j and k, where policy j was the second of the original two policies. Once the user supplies this preference, $\mathbf{P}$ is completely filled in and $\mathbf{c}$ can be updated. Thus whenever a new policy is drawn from the population (Fig. 2a,b), the user must supply m new preferences, where m is the number of policies that have already been shown to the user. Once $\mathbf{c}$ is recomputed, evolution can commence (Fig. 2a): the policies in the current population are evolved using their ages and their scores as predicted by the user model (Fig. 2b,c,i,j).

2.2.2 Drawing a Policy for Scoring

Ten generations of evolution elapse before the algorithm sends a new policy to the user for scoring. For each new policy drawn from the evolving population, an additional row and column are added to $\mathbf{P}$ and an additional element is added to $\mathbf{c}$ to accommodate the additional preferences and score it generates, respectively. When the score for the new policy has been determined, another 10 generations evolve before a new policy is sent to the user, and so on until the trial terminates.

There are various methods that could be employed to decide which policy, when scored by the user, will elicit the most information about what the user prefers. One method would be to employ query by committee [23] in which multiple models are trained, and the policy sent to the user is the only that maximizes the score predictions across the models. Another would be to employ the Estimation-Exploration Algorithm [4], which uses model disagreement as in query by committee, but evolves a separate population of policies such that the fitness of a policy is the amount of score prediction disagreement it induces in the user models. Query by committee was attempted in the present work (data not shown) but it did not yield a better result than a simpler method: Within the current population, the policy with the highest predicted score is sent to the user, as long as that policy has not previously been sent.

2.2.3 The Surrogate User

In order to compare different user modeling methods it

is necessary to collect a large number of preferences from a human user. Following [12, 11] we here create a surrogate user. The surrogate user is a separate algorithm that supplies preferences instead of a human user. The surrogate user can be programmed to mimic any detail of a human user, such as noise (mistakenly preferring the less satisfactory policy) or fatigue (the time to receive a preference from the user increases as evolution proceeds).

Because in the PPL variant the fitness function is not used, the user must guide the robot to the right edge of the barrier (Fig. 1b.c) and then to the end position (Fig. 1b.e) indirectly using preferences. Thus, for any pair of policies i and j shown to the surrogate user, it will prefer policy i over j if

$$\frac{\text{dEdge}(i) + \text{dEnd}(i)}{2} < \frac{\text{dEdge}(j) + \text{dEnd}(j)}{2} \quad (5)$$

$$\text{dEdge}(i) = \sqrt{(x_i^{(\frac{T}{2})} - 4.5)^2 + (y_i^{(\frac{T}{2})} - 3)^2} \quad (6)$$

$$\text{dEnd}(i) = \sqrt{(x_i^{(T)} - 0)^2 + (y_i^{(T)} - 6)^2}, \quad (7)$$

where $\text{dEdge}(i)$ reports how close control policy i gets the robot to the right edge of the barrier (located at $x = 4.5, y = 3.0$; see Fig. 1a) halfway through the evaluation period ($T/2$), and $\text{dEnd}(i)$ reports how close control policy i gets the robot to the end position (located at $x = 0.0, y = 6.0$) by the end of the evaluation period (T). Otherwise, it will prefer policy j over policy i.

2.3 Combining Fitness-based Search and Preference-based Policy Learning (FS-PPL)

The proposed method combines fitness-based search with preference-based policy learning (FS-PPL): both the fitness function and the user model guide search. The FS-PPL algorithm follows exactly the same program flow as the PPL variant described above, with four modifications.

First, the two objectives used during evolution combine age, fitness, and scores produced by the user model (Fig. 2k). It would have been possible to conduct evolution using three objectives rather than two, but there are known challenges with increasing the number of objectives [21] and this would have made a fair comparison to the first two algorithm variants (both of which employ two objectives) more difficult.

The second modification involves the surrogate user. The surrogate user in PPL prefers control policies that get the robot close to the barrier's edge halfway through evaluation and close to the target position at the end of evaluation (Eqn. 5). The surrogate user employed in FS-PPL prefers policy i over policy j if

$$\text{dEdge}(i) < \text{dEdge}(j). \quad (8)$$

That is, the surrogate user prefers policies that get the robot as close to the barrier's right edge as possible halfway through evaluation. It is assumed that once the user model learns this preference and guides the robot to this point, the fitness function will guide the robot toward the target object in the latter half of the evaluation period.

The third modification involved a reduction in the size of the user model. For PPL, the user model is a 12-input, 3-hidden, 1-output neuron feedforward neural network. For FS-PPL, the user model is a 6-input, 3-hidden, 1-output network. Since the user's preferences are only sensitive to the position of the robot halfway through the evaluation period, during training of the user model in FS-PPL, only the values of the five photosensors and one compass sensor halfway through the evaluation period are fed as input ($s_1^{(T/2)}$, ..., $s_5^{(T/2)}$, $s_6^{(T/2)}$).

This does reduce the number of free parameters that must be learned through backpropagation in the FS-PPL surrogate user compared to the PPL surrogate user, but was considered a fair comparison. If a fitness function is not employed, for this task, the user model must make predictions based on sensor data from halfway through and at the end of the evaluation period. The FS-PPL user model only needs to learn based on sensor data from halfway through the evaluation period.

The fourth difference between FS-PPL and PPL is that the FS-PPL surrogate user will stop supplying preferences once any control policy enables the robot to move above the barrier at any point during its trajectory (its position reaches $y > 3$; Fig. 1a). It is assumed that at this point, the user model is sufficiently well trained to guide robots to the barrier's right-hand edge. Henceforth, among policies that get the robot beyond the barrier, the fitness function will favor those that get closer to the target object in the latter half of the evaluation period.

3. RESULTS

For each of the three algorithm variants (FS, PPL and FS-PPL), 100 independent trials were conducted. Each trial was conducted using a population size of 30. Each trial continued until 9 hours of CPU time elapsed[3].

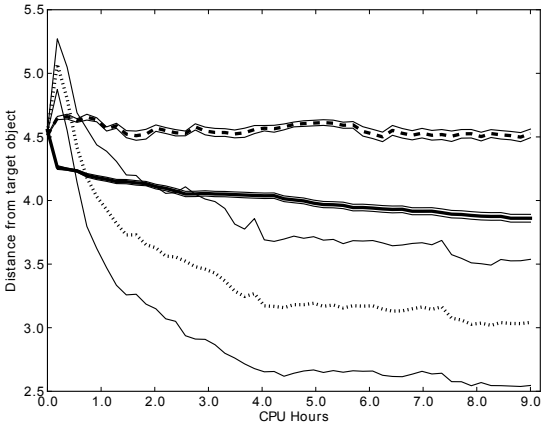
Fig. 3a reports how close the best control policy in the population got to the target object at the end of the trial. As can be seen, combining a fitness function and user model (dotted line) led to policies that got significantly closer to the target object than those policies evolved using only a fitness function (solid line) or only a user model (dashed line).

The preferences of the surrogate models are predicated on the absolute position of the robot (Eqns. 5-8), yet the user model only has access to the robot's photo- and compass sensor values. Thus, the user model must learn some function of these sensor values that approximates Eqns. 5-8. The success of FS-PPL demonstrates that this is possible.

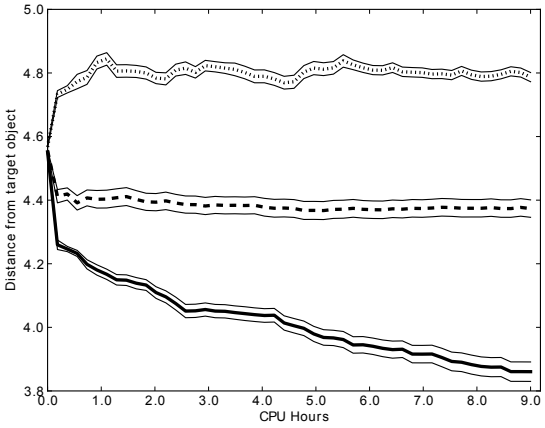
However, it is possible that in some circumstances the user model may not be capable of learning such a mapping. To test this we formulated two additional algorithm variants: we again ran PPL and FS-PPL, but employed neural network-based user models that had no hidden layer. We ran 100 trials of each, and report the result in Fig. 3b. Now, both PPL and FS-PPL perform worse than the algorithm that only employs the fitness function. This indicates that there is a mapping between sensor values and the absolute position of the robot, but that that mapping is nonlinear: the user models that can only learn linear functions failed to learn such a mapping.

Finally, it is possible for a user model to fail to learn a mapping from sensor values to absolute position because the robot is equipped with an insufficient number or type of

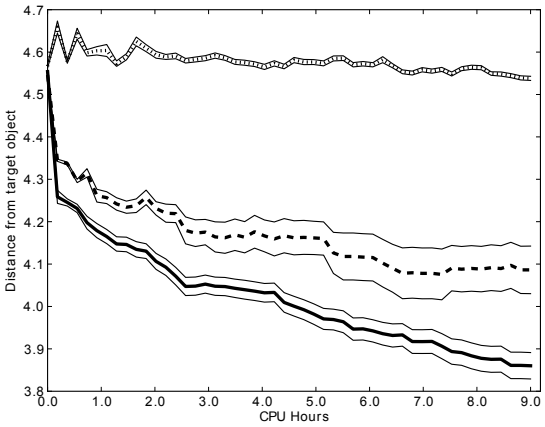
[3]Although CPU time does not report the absolute amount of computational effort required to reach a given distance from the target object, it does provide insight into the relative performance of the three algorithm variants given the same computational budget.



(a)



(b)



(c)

Figure 3: Performances of fitness-based search (dashed line), preference-based policy learning (solid line) and fitness-based search combined with preference-based policy learning (dotted line). Thick lines present the mean; pairs of thin lines bracket ± 1 standard error of the mean. Relative performances employing neural network-based user models with one hidden layer of three neurons (a); no hidden layer (b); and one hidden layer with three neurons but denied input from the compass sensor (c).

sensors. In a final pair of variants we ran PPL and FS-PPL in which the five photo sensor values were fed to the user model, but the compass sensor value was withheld (a constant value of 1 was supplied instead). As shown in Fig. 3c, as for the case of the linear user models, PPL (dashed line) and FS-PPL (dotted line) again failed to learn a predictive user model and both variants failed in comparison to the fitness function-only variant (solid line).

4. DISCUSSION

The failure of the fitness-based search variant is obvious: there is a local optimum in which the robot runs forward, collides with the barrier and stays trapped there.

The successful behavior of the FS-PPL variant using nonlinear models and both photo- and compass sensor data (Fig. 3a, dotted line) however is instructive. The initial spike early in evolution indicates the period during which the user model learns the surrogate user's strategy: the user model learns to reward control policies that guide the robot to the right-hand edge of the barrier by the halfway point of the evaluation period. Such policies are correctly given a high score by the user model, and control policies that drive the robot into the barrier (Fig. 1a) are correctly given low scores.

However, most of these strategies actually guide the robot further from the target object than those that drive the robot into the barrier: the robot may walk to the barrier's right-hand edge but then walk further to the right, or may collide with the barrier's edge and come to a stop. This explains the spike in distance during the early evolution of FS-PPL. At this point however the surrogate user stops providing preferences because at least one of these robots gets beyond the barrier ($y > 3$). Now, search gradually evolves policies that reach the barrier's right-hand edge but also manages to move beyond it and slightly toward the target object: such policies receive a higher fitness value than those that that get stuck at the barrier even though both policies may obtain about the same user model score. Thus, even though the user has stopped interacting with the system and the user model ceases to learn or discover policies with ever higher scores, the control policies continue to improve.

In contrast, the PPL variant fails because either it never learns the surrogate user's strategy or it requires many more user preferences: the user model must first learn to guide robots to the barrier's edge, and then collect additional preferences to learn to guide it to the target object.

4.1 User model analysis

A challenge for any user modeling system however is ensuring that the user model can learn the user's strategy even if she makes decisions based on data that is not directly available to the model. In the example given here, the surrogate user provides preference based on the absolute position of the robot, yet the robot cannot sense its own position. The success of the FS-PPL variant indicates that such a mapping can be learned, but only if a nonlinear model is employed, and only if both photo- and compass sensor data is made available to it.

The Eureqa symbolic regression tool [19] was used to discover what form this mapping from sensor data to absolute position is. This was accomplished by taking the trained user model from one of the successful FS-PPL trials and 200 control policies at random from later generations of

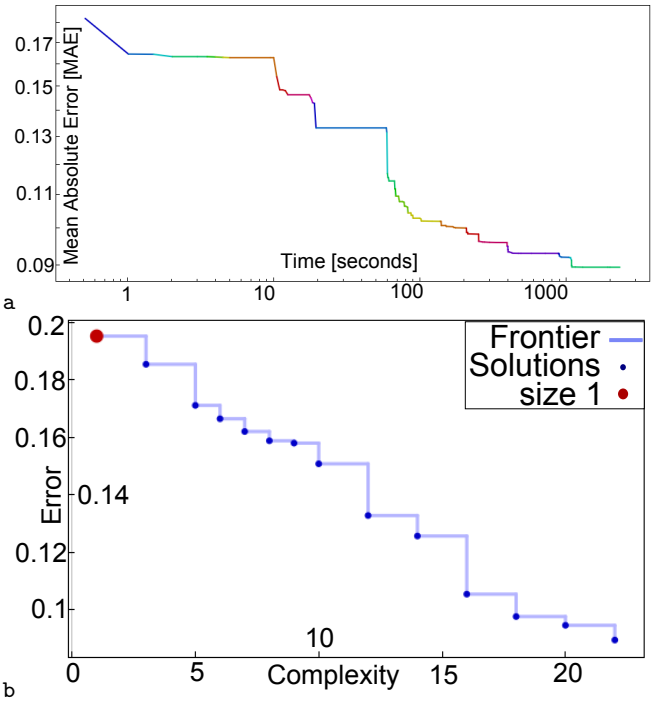


Figure 4: a: Evolutionary improvement in discovering a relationship between a successful user model's output and the sensor data supplied to it. b: The final Pareto front of the evolutionary search.

that run. Each of the 200 policies was re-run and the sensor values arising half way through the evaluation period were recorded. Each of these 200 sensor value sets was supplied to the user model and the score returned by the model was recorded. This provided us with 200 tuples of the form $(s_1^{(T/2)}, \ldots, s_6^{(T/2)}, o)$ where o is the score output by the user model and lies in the range of $[0, 1]$. This data was provided to Eureqa and allowed to evolve for 42 minutes on a standard PC[4]. Fig. 4a reports the evolutionary improvement in search and Fig. 4b reports the final Pareto front returned by Eureqa, which evolves low error and low complexity solutions of the form $o = f(s_1^{(T/2)}, \ldots, s_6^{(T/2)})$. Only the four algebraic operators $+$, $-$, $\times$, and $/$ were allowed.

Table 1 reports a subset of the equations from the front shown in Fig. 4b. The most accurate model of size 3 indicates that a high score is output if s_5 takes as low a value as possible. This fifth photosensor is located in the robot's back left leg (this leg is darkened and circled in Fig. 1b). As can be seen in that final successful trajectory, the back left leg points away from the target object at the half way point (Fig. 1b.c), lowering this sensor's value relative to the other four photosensors. However, the $-s_1$ terms in the denominators of the 20- and 22-node solutions indicate that higher scores are given for higher values of s_1, which is the front left leg. Again, in Fig. 1b.c it can be seen that the front left is closer to the target object than the back left leg.

Thus, part of the user model rewards for the front leg being closer to the target object than the back left leg. But these relative sensor values can also be achieved by the robot that drives into the barrier (Fig. 1a.a). So, the user model also rewards for particular values of the compass sensor (s_6 is prevalent in Table 1), which indicates that the user model rewards for particular orientations of the robot. By combining the photo- and compass sensor values, the user model is guarding against sensor aliasing. Finally, it is clear that the linear models (first three rows in Table 1) are poor approximations of the user model and only the nonlinear models (lower rows) approach good approximations of the user model. This corroborates our finding that only nonlinear user models (coupled with the fitness function) successfully discovered the optimal solution for this problem.

[4]Do not panic: this amount of computation time was chosen arbitrarily.

Table 1: Approximations of a successful user model. Size=number of operators and operands comprising the model; r^2 =the amount of variation explained by that model.

Size	r^2	User model
1	2.9×10^{-25}	$o = 0.154$
3	0.012	$o = 0.83 - s_5$
5	0.196	$o = 0.249 - 0.2s_6$
6	0.204	$o = \frac{0.0733}{0.194+s_6}$
7	0.254	$o = 0.991 - s_4 - 0.308s_6$
8	0.277	$o = \frac{0.0421}{s_3+s_6-s_5}$
...	...	...
20	0.731	$o = \frac{0.0102}{0.0894+s_2+0.457s_6^2-s_1-0.377s_3s_6}$
22	0.829	$o = \frac{0.0103}{0.0827+s_2+4.01s_6^3-s_1-2.11s_6^2}$

5. CONCLUSIONS

Here we have demonstrated an evolutionary robotics method in which a human user teaches a user model to guide search away from local optima, but can then cease interacting with the system while the fitness function automatically guides search toward the global optimum. This is demonstrated here as follows: the user guides the robot toward the right edge of a barrier indirectly by supplying preferences for rightward tending trajectories. Once the robot clears the barrier the user ceases interaction, and the fitness function selects for control policies that bring the robot from the barrier's edge to the target position.

This stands in contrast to Preference-based Policy Learning [1], which also employs a learned model of the user but, since there is no fitness function, the user must guide the robot to the barrier's edge and then to the target position. This approach was found to perform poorly on this task. Our finding resembles the finding in [15] where it was shown that novelty search—which, like PPL, attempts to guard search from becoming mired in local optima—can benefit from the focusing effect of a fitness function.

In the work presented here, the user's preference could easily be modeled because preferences were generated by a fixed strategy. In future work we will investigate user preferences that are more noisy, can only be described by complex functions derived from sensor data, and non-stationary.

Besides providing preferences, there are other ways that human users might positively influence an evolutionary algorithm without having to write code. Given the right fitness function, a user can create virtual task environments interactively that select for different kinds of behaviors [3]. Or, the user may manipulate a virtual robot to demonstrate an approximation of the desired behavior and multiobjective optimization may then balance retention of the demonstrated behavior with satisfaction of the fitness function.

In general, user modeling methods show promise because they allow a casual user to incorporate their intuitions about a robot task into search without having to write computer code: they do so simply by indicating which behaviors they like more than others. In future work we plan to investigate multiobjective systems that combine diversity-generating methods such as novelty search, fitness functions, user modeling, different options for user interaction and combining interactions from multiple users.

Acknowledgements

This work was supported by National Science Foundation Grant PECASE-0953837 and DARPA M3 grant W911NF-1-11-0076. The authors also acknowledge the Vermont Advanced Computing Core which is supported by NASA (NNX 06AC88G), at the University of Vermont for providing High Performance Computing resources that have contributed to the research results reported within this paper.

6. REFERENCES

[1] R. Akrour, M. Schoenauer, and M. Sebag. Preference-based policy learning. *Machine Learning and Knowledge Discovery in Databases*, pages 12–27, 2011.

[2] G. Barnum and C. Mattson. A computationally assisted methodology for preference-guided conceptual design. *Journal of mechanical design*, 132(12), 2010.

[3] J. Bongard, P. Beliveau, and G. Hornby. Avoiding local optima with interactive evolutionary robotics. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, pages 1405–1406. ACM, 2012.

[4] J. Bongard and H. Lipson. Automated reverse engineering of nonlinear dynamical systems. *Proceedings of the National Academy of Science*, 104(24):9943–9948, 2007.

[5] M. Campbell, R. Rai, and T. Kurtoglu. A stochastic graph grammar algorithm for interactive search. In *14th Design for Manufacturing and the Life Cycle Conference*, pages 829–840, 2009.

[6] W. Cheng, J. Fürnkranz, E. Hüllermeier, and S. Park. Preference-based policy iteration: Leveraging preference learning for reinforcement learning. *Machine Learning and Knowledge Discovery in Databases*, pages 312–327, 2011.

[7] R. Dawkins. *The blind watchmaker: Why the evidence of evolution reveals a universe without design*. WW Norton & Company, 1986.

[8] K. Deb, A. Sinha, P. Korhonen, and J. Wallenius. An interactive evolutionary multiobjective optimization method based on progressively approximated value functions. *Evolutionary Computation, IEEE Transactions on*, 14(5):723–739, 2010.

[9] A. Doan, R. Ramakrishnan, and A. Halevy. Crowdsourcing systems on the world-wide web. *Communications of the ACM*, 54(4):86–96, 2011.

[10] F. Gruau and K. Quatramaran. Cellular encoding for interactive evolutionary robotics. In *Fourth European Conference on Artificial Life*, pages 368–377, 1997.

[11] G. Hornby and J. Bongard. Accelerating human-computer collaborative search through learning comparative and predictive user models. In *Procs. of the fourteenth Intl. Conf. on Genetic and evolutionary computation conference*, pages 225–232, 2012.

[12] G. Hornby and J. Bongard. Learning comparative user models for accelerating human-computer collaborative search. *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, pages 117–128, 2012.

[13] G. S. Hornby. ALPS: The age-layered population structure for reducing the problem of premature convergence. In *Proc. of the Genetic and Evolutionary Computation Conference, GECCO-2006*, pages 815–822, 2006.

[14] J. Lehman and K. Stanley. Exploiting open-endedness to solve problems through the search for novelty. *Artificial Life*, 11:329, 2008.

[15] J. Mouret. Novelty-based multiobjectivization. *New Horizons in Evolutionary Robotics*, pages 139–154, 2011.

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

[17] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 1:213, 2002.

[18] M. Schmidt and H. Lipson. Actively probing and modeling users in interactive coevolution. In *Procs. of the Eighth Annual Conf. on Genetic and Evolutionary Computation*, pages 385–386, 2006.

[19] M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81, 2009.

[20] M. Schmidt and H. Lipson. Age-fitness pareto optimization. *Genetic Programming Theory and Practice VIII*, pages 129–146, 2011.

[21] O. Schutze, A. Lara, and C. Coello. On the influence of the number of objectives on the hardness of a multiobjective optimization problem. *Evolutionary Computation, IEEE Transactions on*, 15(4):444–455, 2011.

[22] J. Secretan, N. Beato, D. D'Ambrosio, A. Rodriguez, A. Campbell, J. Folsom-Kovarik, and K. Stanley. Picbreeder: A case study in collaborative evolutionary exploration of design space. *Evolutionary Computation*, 19(3):373–403, 2011.

[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] L. Von Ahn. Games with a purpose. *Computer*, 39(6):92–94, 2006.

[25] P. Wolynes. Latest folding game results: Protein A barely frustrates computationalists. *Proceedings of the National Academy of Sciences*, 101(18):6837–6838, 2004.