DOI: http://dx.doi.org/10.7551/978-0-262-32621-6-ch158

Improving Robot Behavior Optimization by Combining User Preferences

Anton Bernatskiy[1] , Gregory S. Hornby[2,3], Josh C. Bongard[1]

[1] University of Vermont, Burlington, VT 05401
[2] University of California Santa Cruz, Santa Cruz, CA 95064
[3] NASA Ames Research Center, Mountain View, CA 94035
abernats@uvm.edu

Abstract

Recently it has been demonstrated that collaboration between automated algorithms and human users can be especially effective in robot behavior optimization tasks. In particular, we recently introduced a Fitness-based Search with Preference-based Policy Learning (FS-PPL) approach, in which the algorithm models the user based on her preferences and then uses the model, along with the fitness function, to guide search. However, so far only interaction between a single human user and an evolutionary algorithm was considered. If multiple users contribute preferences, the algorithm must determine whether to model them separately or jointly. In this paper we describe an algorithm in which one evolutionary algorithm interacts with two users and determines the best way to model them automatically. We test the algorithm with automated substitutes for human users and show that it performs better for two users working together than for the same users working separately, thus demonstrating the potential for crowd-sourcing robot behavior optimization.

Introduction

Historically, interactive evolutionary algorithms are typically used to solve search problems in which automatic evaluation of a solution candidate is impractical for some reason – for example, artistic tasks. In this case the duty of solution evaluation is fully entrusted to the user. Many successful algorithms were designed using this approach, including those which allow multiple users to collaborate on the same problem (Secretan et al. (2008); Szumlanski et al. (2006); Kuzma et al. (2009)). Much less is known, however, about the algorithms which distribute the burden of solution candidates evaluation between the users and the computer.

In this work we employ this latter approach to address an important issue arising in traditional fitness-based evolutionary algorithms – namely, the phenomenon of premature convergence, i.e. convergence to a local optimum with a large basin of attraction rather than to the global optimum with a much narrower basin.[1] One approach used to combat this problem is to use multiple objectives instead of just a single fitness value to evaluate solutions. Some objectives shown to be effective are age (Schmidt and Lipson (2011)) and novelty (Mouret (2011)). However, depending on a task, even multiobjective algorithms can become trapped on local optima.

For some tasks this problem can be greatly reduced by adding human preference as an optimization objective. This is particularly true for robot behavior optimization, because humans have good intuition about legged locomotion and are able to visually determine that search has become trapped on a local optimum (Bongard et al. (2012)). The major problem with these methods, however, is the quantity of preferences required from the user, which is often so demanding that it makes the algorithm too labor-intensive to be practical.

This problem can be approached in several ways. One way is to use a machine learning algorithm to build a model of the user and then use the model to supply preferences on the human user's behalf as behavior optimization continues (Takagi (2001); Schmidt and Lipson (2006); Akrour et al. (2011); Bongard and Hornby (2013)). In (Bongard and Hornby (2013)) we investigated the efficiency of this approach in a robot behavior optimization task with a deceptive fitness landscape. Using an algorithm based on Age-Fitness Pareto Optimization (AFPO) (Schmidt and Lipson (2011)) with an additional user preference objective and a neural network-based user model, we showed that a user model and fitness function together can guide the search to convergence more rapidly (in terms of wall-clock time) than either of them on its own.

Another way to cope with the labor intensity of interactive evolution is to utilize evaluations coming from multiple users. This approach has been investigated theoretically to some extent (Szumlanski et al. (2006)) and successfully applied to artistic tasks (Secretan et al. (2008); Kuzma et al. (2009)).

Our hypothesis is that it is possible to make the optimization of robot behavior faster by collecting evaluations simultaneously generated by multiple users into one common evolutionary algorithm. Consider an algorithm which attempts to learn preferences supplied by multiple users based on

[1] Fitness landscapes with such optima are said to be *deceptive*.

their evaluations. If n users simultaneously indicate preferences and if their preferences agree, then the machine learning algorithm can train on these preferences as if they were indicated by a single user. Therefore, it will have up to n times more training data, which will allow it to build an accurate user model faster.

If user preferences disagree, the algorithm will have to model users separately using their respective preference sets. In this case the speed of learning of each user model is reduced back to the level of the single user case, and additional computational costs associated with training multiple user models can impact the performance of the behavior optimization method (see the Experiments section). However, disagreement in users' preferences is likely to indicate that more than one global optimum – or several similar (in terms of fitness) local optima – have been intuited by the users and are present in the fitness landscape. In the latter case it is possible to exploit the disagreement to explore both of the user-favored optima, evaluate them and determine if one of the user-favored optima is better than the other in terms of fitness.

To test these suppositions we have developed an interactive, user-modeling algorithm which can simultaneously accept preferences from one or two users. We measure its performance with two users working together and compare it to the combined performance of two users working separately, each with her own evolutionary algorithm and user model.

Test Problem

We use the test problem from (Bongard and Hornby (2013)). The goal is to navigate a simple quadrupedal robot around the wall to a target object on the far side (Fig. 1a). The robot is composed of a square plate and four rigid vertical legs, each attached to the plate by an actuated joint with one degree of freedom (Fig. 1b).

Each body part has one light sensor and one touch sensor. Signals from the photosensors are real values from $[0, 1]$ varying linearly depending on their euclidean distance from the light source. [2]. Touch sensors produce 1 if the body part touches the ground or collides with the wall and -1 otherwise. Additionally, the robot is equipped with a compass sensor which gives the current robot's orientation relative to the Y axis, normalized to be in $[0, 1]$.

The robot is controlled by a feedforward neural network without hidden nodes. A total of 11 sensors connect to four actuators, which yields a total of 44 synaptic weights. Hereafter we will refer to a particular set of synaptic weights as *a controller*.

Methods

The algorithm uses a client-server computational architecture. The client here is an interactive program which takes a pair of controllers as input, simulates[3] two copies of the robot with controllers from the pair and shows the resulting behaviors to the user (Fig. 2). The user is forced to prefer one – she cannot skip a pair. After the preference is provided, the client sends it to the server.

The server performs the following functions:

- it supplies controllers to and receives preferences from multiple clients via asynchronous communication;
- it optimizes the robot's behavior with an evolutionary algorithm;
- it generates the controller pairs to be evaluated by users and maintains the users' preference tables;
- it trains the user models based on users' preferences;
- it employs predictions from the user models along with the fitness function to guide the evolutionary algorithm.

A *user model* is defined as a mapping from a pair of robot behaviors to a prediction of the user's preference for this pair. The mapping is learned by an artificial neural network[4] with a hidden layer using backpropagation. For details, see the User Models section below.

If only one user has supplied preferences so far, only one user model is maintained. If two users supply preferences, the program must find an optimal way to utilize these. For this purpose our program maintains three separate user models – one *individual model* for each user and one *collective model*, which is trained on the combined preferences of both users. For details see the Coordinated Score Generation section below.

Evolutionary Algorithm

For robot behavior optimization the server uses Age-Fitness Pareto Optimization (Schmidt and Lipson (2011)), an evolutionary algorithm with two explicit objectives – fitness and age. In all experiments described below the algorithm starts with a population of 30 controllers, initialized with random synaptic weights in $[-1, 1]$. The server simulates controllers sequentially and records the full time series of the resulting sensor values. When all controllers in the population have been simulated, the algorithm calculates their fitness values and constructs the Pareto front, taking the time controllers have spent in the population – their *age* – into account. The next generation is composed of

- one new, completely random controller,

[2] The sensors saturate to 0 for distances about 5 times greater than the maximum distance any robot traveled in our experiments.

[3] All physics simulations use Open Dynamics Engine, `http://www.ode.org`.

[4] This network is not to be confused with the robot's controller (see the Test Problem section), which is another artificial neural network employed in the program. Unlike the one described here that one has no hidden neurons.

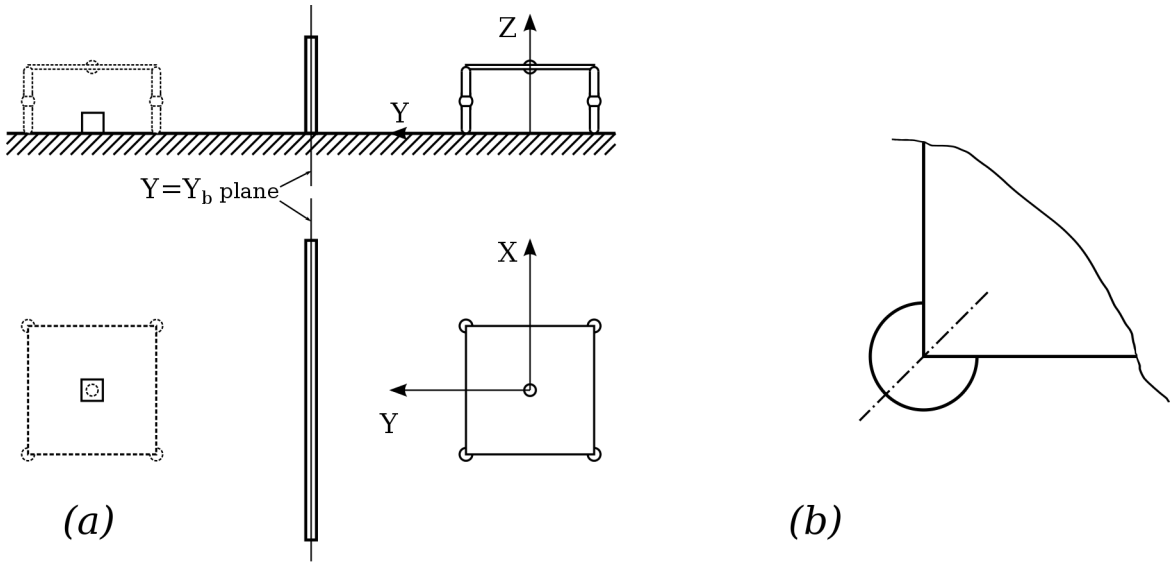

Figure 1: Test problem. *(a)* Side and top views of the robot and its environment at the beginning of the simulation. The small square to the left denotes the light source; spheres on the robot's body are light sensors. The target position that the robot should reach is depicted with dotted lines. Y_b denotes the Y coordinate of the barrier. *(b)* Joint between the robot's main body (square plate) and a limb, top view. The dotted line denotes the axis of rotation. The angle of the limb's rotation relative to its default position (as in *(a)*) can take values in $[-45°, 45°]$. A video of the robot with a successfully evolved controller can be viewed at `http://youtu.be/ByDfAcDBsHI`.

- nondominated controllers from the previous population and
- their mutated copies, in a quantity sufficient to restore the initial size of the population.

The fitness function is

$$f = f_u \sigma, \tag{1}$$

where f_u is the unscaled fitness (Bongard and Hornby (2013)):

$$f_u = \frac{1}{1 + \left(\sum_{i=1}^{5} \sum_{t=1}^{T} \| s_i^{(t)} - s_i^{(r)} \| \right) / 5T}. \tag{2}$$

$T = 1000$ here is the number of time steps during which behavior is simulated, $s_i^{(t)}$ is a value of ith light sensor at time step t, and $s_i^{(r)}$ is the value of the ith light sensor at the goal position (see Fig. 1a).

σ is the *coordinated score*: a number in $[0, 1]$ which represents a combined prediction from all of the user models about how much the user (or users) would like this controller. In particular, σ near 1 indicates that at least one of the two user models tended to prefer this controller when it was presented multiple times, while a score near 0 indicates that the user models predict that both users will greatly dislike this controller. In the beginning of the program's operation, when no users' preferences have been provided yet, it is equal to 0.5 for all controllers. For details on σ see Coordinated Score Generation section below.

In the current implementation, the second generation commences only after the first pair of controllers has been evaluated by a user. This ensures that the coordinated score σ affects evolution from the outset. However, in practice, this should have little impact on evolution, because the user models learn more slowly than the evolutionary algorithm improves the robot's behavior: it takes many before the user models' predictions deviate significantly from 0.5.

User Preference Gathering

After evaluating the first generation, the server ranks the controllers from the Pareto front by fitness and requests the evaluation of the four best controllers from the users. The first user must compare the first and the second controller, and the second user compares the third controller to the fourth. The program waits for either user to evaluate her pair and then enters the evolutionary loop of reproduction and selection (see the section above). The server never pauses to wait for any user action after the indication of this first preference.

Every time the program evaluates all unscaled fitness values f_u of the controllers from the current generation, it checks whether any of its previous requests for user preferences were granted. If that is not the case, the program continues with the next iteration of the evolutionary loop. Otherwise, it stores the obtained preference into a table of

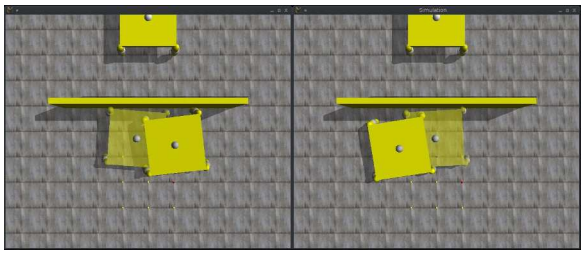

Figure 2: Screenshot of two clients running on the same computer. The user can select a behavior she likes by cycling through the robots. The selected robot is highlighted and the other one is made translucent. The preference is sent to the server as soon as the user confirms the selection. For example, in the left window the user is about to confirm her preference towards the highlighted robot to the right of the other contestant.

preferences, selects a new pair of controllers for user evaluation, sends it to the client and, if appropriate, retrains some user models on the expanded set of preferences.

All controllers sent to the client for user evaluation are stored, along with their respective sensor time series, in an archive. The obtained user preferences are stored in the preference table P, such that $P[i,j] = 1$ if the ith controller of the archive was preferred to the jth, -1 if the jth controller was preferred to the ith, and 0 if the preference is neutral (in the current implementation that is possible only for $P[i,i]$) or not yet known (Bongard and Hornby (2013), Akrour et al. (2011)).

To accelerate the filling of the preference table we assume that user preferences are transitive. Consider a situation when the user has seen n controllers $c_1, c_2, \cdots, c_n$ so far, and for every $i < j$ she preferred c_j to c_i. The program assumes then that if a new controller c' is preferred over c_j for some $j \leq n$, then all controllers c_i (for which $i \leq j$) are assumed to not be preferred over c'. Similarly, if c' is not preferred over c_j, then all controllers from the upper part of the ranking, c_i (with $i \geq j$) are assumed to be preferred over c'.

To determine how a new controller fares against controllers previously shown to a user, the program uses a version of binary search adapted for our purposes. First, for each controller already shown it produces a score: the number of times this particular controller was preferred to its peers minus the number of times it was not preferred. If a new controller c' is preferred over some previously shown controller c_i, then it is assumed that c' is preferred over all previously shown controllers with a score less than or equal to the score of c_i, and the corresponding entries of $P[i,j]$ are stored. Similarly, if some previously shown controller c_i is preferred to the new one, the algorithm assumes that all controllers with the score higher than that of c_i are preferred to the new controller.

The old controllers are shown to the user (paired with the new controller) in the following order: the controller with the highest score is shown first, then the one with the lowest score and then – repeatedly – the closest one to the middle of the current interval of possible values of the score for the new controller. The algorithm terminates when all of the relationships between the new controller and the previously known ones are established. In the worst case this happens after the user has indicated $2 + \log_2 n$ preferences; in the best case one preference is sufficient.

When the binary search described above terminates, two events occur. First, a new controller is selected among the current evolutionary population to be evaluated by the user. In the experiments described here, the algorithm selected the most fit controller among those which have not been seen by any user yet. The server sends the pair, as dictated by the first step of the bisection algorithm described above, to the user.

Second, two user models are retrained: the individual model corresponding to the newly gathered preference's author and the collective user model. The models are trained on the fully evaluated subsets of users' archives, i.e. on those subsets for which the preference is known for each pair of controllers in the subset. The individual model is retrained on the preference table of the user who indicated the last preference; the collective model uses the tables of both users.

This process of robot behavior optimization, preference gathering, and user modeling is repeated indefinitely, or until the server process is terminated.

Note that with the pair selection strategy described above a user never gets to evaluate a controller which has already been seen by her peer. The motivation for this is twofold. First, this approach maximizes the diversity of controllers available to the collective user model, which in turn maximizes its potential for accurate prediction. Second, it facilitates the detection of situations when the learned user models tend to overfit the user data (see the Discussion section).

User Models

A *user model* is a mapping between the robot's behavior and an assessment of its quality by the user. In this particular algorithm we employ a mapping which takes as input two robot behaviors compressed into *feature vectors* and maps them onto a value from $[-1, 1]$, approximating the record of the preference table P (see the User Preference Gathering section).

In all experiments described here we use the values from six sensors (five light sensors and one compass sensor) of the robot recorded at the middle ($t = T/2$) of the simulation as the feature vector (Bongard and Hornby (2013)). This kind of compressed representation simplifies the problem of learning the user model. Designing a general way in which such a vector can be generated to facilitate learning is

a nontrivial problem and it is not considered in this work.

The mapping is learned by an artificial neural network with 12 inputs – six for each feature vector of the two controllers which the model is supposed to compare. These neurons are connected to the only output of the network through a single hidden layer containing 12 neurons.

For convenience, the output neuron is trained to reproduce not the $P[i,j]$ itself, but its linear transformation to $[0,1]$:

$$\text{target}(i,j) \equiv \frac{P[i,j]+1}{2}. \quad (3)$$

The network is trained using error backpropagation (Rumelhart et al. (1986), Bongard and Hornby (2013)). The algorithm iterates through all entries of the preference table $P[i,j]$ and backpropagates the network's errors associated with each entry once. If the network being trained is the collective user model, the same procedure is applied to the other user's preference table as well. Then it iterates through all of the entries again and compares the sign of the model's prediction to the sign of the original entry. If the signs coincide for all entries, the network is considered to be successfully trained. Otherwise, the procedure is repeated, but no more than $10^4(m/2-n)$ times, where m is the total number of table entries and n is the total number of controllers. If this number is reached, the learning process is considered to have failed.

Depending on the outcome of the learning procedure, the algorithm assigns *model errors* to each generated user model as follows:

- 10 if the learning failed;
- 2 if the learning was attempted on one preference table and succeeded;
- 1 if the learning was attempted on two preference tables and succeeded.

This value is used to determine the optimal way to utilize the three user models. As we will see in the next section, the behavior of the algorithm we use to accomplish that does not depend on the particular values we chose to represent the models errors, but rather on the relative position of these values on the real axis with respect to each other.

Coordinated Score Generation

To generate coordinated scores σ (see the Evolutionary Algorithm section above) for the newly-evolved controllers, the server starts by producing scores based on each one of the user models present (Bongard and Hornby (2013)).

To determine these scores for an evolutionary population of size 30, each user model fills a 30×30 table $\mathcal{P}[i,j]$ with its preference approximations (from $[0,1]$). The score is then calculated as

$$\sigma_k(j) = \frac{1}{30}\sum_{i=1}^{30} \mathcal{P}_k[i,j], \quad (4)$$

where $k \in \{0,1,C\}$ is the index of the user model: 0 and 1 correspond to the first and second individual user models respectively and C corresponds to the collective model[5].

Denoting the errors of the models (defined in the previous section) as ϵ_0, ϵ_1 and ϵ_C, the coordinated score σ can be computed as follows:

1. If there is only one user model, use its score;
2. If $\epsilon_C < \epsilon_0$ and $\epsilon_C < \epsilon_1$, use σ_C;
3. Otherwise, use $\max(\sigma_0, \sigma_1)$.

The first rule describes the trivial behavior the algorithm exhibits when only one user supplies preferences. The second corresponds to the condition under which the score from the collective user model should be used. With model error defined as we did in the previous section, this decision is always made when the backpropagation algorithm was able to train the collective user model successfully. The basis for this decision is the assumption that if it is possible to successfully train the collective user model on the data provided by two independent users, then these users are likely to be "allied", i.e., they are guiding the evolutionary search towards the same optimum.

The third rule describes the case when users are likely to have different opinions regarding which optimum is a global one. In that case the $\max$ function helps to retain controllers which are favored by one of the two users. This allows us to take both users' opinions into account and subject behaviors favored by each one of them to direct competition in the evolutionary algorithm.

We do not consider users who make errors or change their opinion over time in this work.

Experiments

To reduce the amount of effort required to test the algorithm and increase the experiments' reproducibility, we employed surrogate users in place of humans (Bongard and Hornby (2013)). A surrogate user is a version of the client program which emulates the behavior of a human user with particular preferences. In our experiments surrogate users preferred robots that attempt to circumnavigate around the right edge of the barrier, which is detected by measuring which one of the two controllers yields the largest X coordinate of the robot (Fig. 1) at the mid-point of the simulation ($t = T/2$)

[5] A similar metric was defined in the User Preference Gathering section to rank controllers by the degree to which a user likes or dislikes them. The value we generate here serves a similar purpose, but is computed using a different set of controllers.

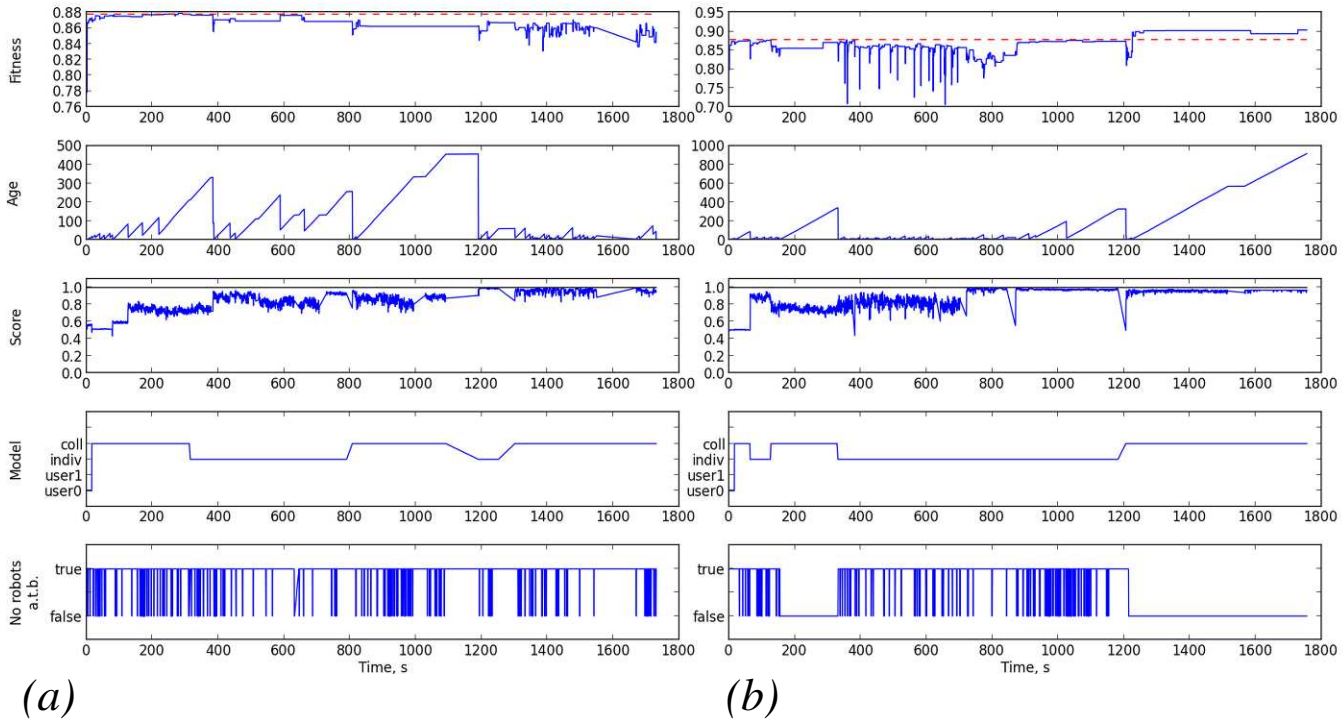

Figure 3: Time series for some parameters of the server over the course of typical allied simulations: **(a)** an unsuccessful simulation and **(b)** a successful simulation. The three topmost graphs represent the unscaled fitness f_u, age and the coordinated score σ of the current best controller in the evolutionary population (computed using the product $f_u\sigma$). The red dotted line in the fitness graph (top) shows a rough estimate of the maximal value of f_u which the robot *not going* around the barrier can have (0.88). The fourth graph from the top shows how the way in which the server modeled users changed over the course of the simulation: "coll", "u0" and "u1" indicate the usage of the collective user model and the individual models of the first and the second user, correspondingly; "indiv" corresponds to the two users being modeled separately. See the Coordinated Score Generation section for details. The graph at the bottom gives the logical value "No robots above the barrier": false if there are any controllers in the current population which make robot travel beyond the barrier (i.e., have $Y \geq Y_b$ at some point of the behavior simulation) and true otherwise.

(henceforth we will say that such a surrogate user *prefers "rightmost" behavior*). For emulating users with different strategies, we also made a version of the surrogate user who prefers behaviors with the lowest X coordinate at the same point of the evaluation period (i.e. a user who prefers "leftmost" behaviors).

Also, the surrogate user stopped supplying preferences and terminated the client if it encountered a controller which is able to guide the robot around the barrier, i.e., to have some points in its trajectory with Y greater than the coordinate Y_b where the barrier is located (see Fig. 1).

Results

In each run discussed below the server ran for 30 minutes of wall clock time. One or two clients controlled by the surrogate users were run on the same computer as parallel processes (Fig. 2). Once every 60 seconds the clients supplied preferences to the server.

Three types of simulation were performed:

- *single user simulations* with one surrogate user preferring "rightmost" behaviors;
- *allied simulations* with two surrogate users preferring "rightmost" behaviors;
- *opposing simulations* with one surrogate user preferring "rightmost" behaviors and one surrogate user preferring "leftmost" behaviors.

We considered a simulation to have succeeded if during the last 20 generations it had at least one controller in the server's evolutionary population which was able to guide the robot around the barrier (defined as above).

Figure 3 demonstrates how some parameters of the server change over the course of two typical allied simulations. Periodically, the age of the most fit controller stays constant for short time periods ("plateaus" on the age graphs). This occurs when the server is busy with model training for a significant portion of time and indicates the presence of a

Simulation type	# of wins/ # of runs	Rate of success	# of generations per run Average±std. deviation	# of generations spent using the collective model/ Total # of generations
Single user	53/300	0.177	$(3.12 \pm 0.13) \times 10^2$	0/937443 (0%)
Allied users	86/300	0.287	$(2.59 \pm 0.28) \times 10^2$	538610/777250 (69%)
Opposing users	25/300	0.083	$(2.48 \pm 0.20) \times 10^2$	397220/744647 (53%)

Table 1: Experimental results

significant computational overhead related to training of the user models.

The graphs for the opposing simulations are very similar. Graphs for single user simulations differ from Figure 3 in two respects. First, there is no switching between usage of individual and collective user models to guide evolution: the algorithm has only one user model. Second, the amount of time the server spends training the user model is substantially lower.

We performed 300 runs of each of simulation type with the servers configured as described above. The results are presented in Table 1.

We used the one-tailed Z-test to compare the success rates (LeBlanc (2004)). The rate of success for allied simulations was found to be significantly higher than the success rate of the single user simulations ($p \leq 7 \times 10^{-4}$), despite the significantly lower ($p < 10^{-4}$ by the standard t-test) average number of evolutionary generations per run.

The success rate for the opposing simulations was found to be significantly lower ($p \leq 7 \times 10^{-4}$) than the success rate of the single user simulations. The average number of generations per run is about the same as for the allied simulations, and is significantly less than the number of generations for the single user simulations ($p < 10^{-4}$).

The ratio between the number of generations which the server spent using the collective model and the total number of generations was found to be significantly higher ($p < 10^{-5}$) in the allied simulations than in the opposing simulations.

Out of all 25 opposing simulations which succeeded, at least 10 did so by taking the robot around the right side of the barrier and at least 9 used the left side of the barrier[6].

Discussion

If a simulation involves two users, on average it iterates through fewer generations of the evolutionary algorithm than a simulation with only one user. This is explained as follows: in single user simulations the server maintains only one user model, which reduces the computational expense required for model training compared to the two user case in which three user models must be continually trained and re-trained. This reduced computational burden is exploited by the evolutionary algorithm, which is now able to perform more generations.

The results also indicate that in allied simulations the program performs better than in the single user simulations. We explain this as follows: increased rate at which the common user model receives user preferences, coupled with the consistency of these preferences, causes the increase in the model's learning speed, yielding an accurate user model quicker (compared to the single user case). This result confirms our hypothesis that it is possible to accelerate robot behavior optimization by utilizing preferences from multiple users, despite the additional cost incurred by having to train individual and collective user models.

We hypothesize that the inferior performance of the program when it hosts opposing users is due to the three following factors:

1. When the coordinated score is generated as a maximum of scores by the individual user models, the evolutionary population is effectively divided into two subpopulations, each of which consists of controllers favored by the corresponding individual user model. This leads to a growth of the Pareto front and ultimately slows down search. We hypothesize that this problem may be remedied by utilizing an evolutionary algorithm which treats the Pareto front in a different way and/or employs a larger population.

2. In the experiments presented here, during a substantial fraction of generations (53%) opposing simulations employed the collective user model to guide search. The collective user model "successfully" learned a data set which has implicit internal inconsistencies. That is, the model must learn to take two similar inputs yet output two very different predictions: for example, the first user very much liked the first behavior, but the second user greatly disliked the second, similar behavior. This suggests that the model has overfit the data and its usage can negatively impact the algorithm's search ability.

   Notice that if there were at least two controllers which both users has seen, it would become impossible to "successfully" train the collective user model. This can conceal the problem of overfitting. That's why, although such overlap would help the algorithm to recognize the situation when it is better to model users separately, it

[6]For the remaining six runs the wall was circumvented late into the simulation, which prevented the winning controller from being recorded.

is not allowed in the experiments reported here. This was one of the reasons why we decided to query the users on completely disjoint sets of controllers (see the User Preference Gathering.

This problem might be solved by using a different metric for the user model's learning efficiency.

3. The additional computational overhead mentioned above in the context of allied simulations already places this simulation at a computational disadvantage compared to the single user simulation.

Despite the general failure to accommodate opposing users, the algorithm still managed to solve the task in a substantial fraction of runs. These runs succeeded by discovering both user-favored optima, which indirectly confirms our second hypothesis about the possibility of finding and comparing multiple user-favored optima in the fitness landscape.

Conclusions

Our findings confirm that in robot behavior optimization tasks it is possible to increase the performance of fitness-based, user-assisted evolutionary algorithms by utilizing preferences from multiple users. This constitutes a step towards fitness-based, crowd-assisted algorithms which may potentially solve problems too deceptive to be solved by purely automated algorithms.

We demonstrated that employing more than one user can help solve robot behavior optimization tasks in at least two ways. First, if users approach the task with the same strategy, this approach allows the optimizer to recognize and employ the strategy more rapidly. Second, if the users employ different strategies, it is possible to find all optima recognized by the users and choose the best one among them.

However, the task of designing such algorithms is far from trivial. Here we would like to highlight some difficulties particular to search algorithms guided by multiple users, which employ user modeling. A good algorithm of this type must

1. be able to distinguish between different user strategies and model each appropriately;

2. employ user modeling algorithms flexible enough to adapt to any or almost any benign user strategy, yet not overfit user input and thus retain good extrapolation properties; and

3. employ a search algorithm which is able to retain good performance while utilizing user models that change in number and quality.

Every one of these tasks constitutes a nontrivial design problem in its own right. However, we believe that all of these challenges can be addressed by a suitable combination of machine learning techniques. Possible future work may include utilizing clustering to solve the first subproblem listed above (pioneered in (Kuzma et al. (2009))), evolving user models of varying accuracy and complexity to address the second one and designing evolutionary algorithms with better scaling properties to handle the last one.

Acknowledgments

This work was supported by DARPA M3 grant W911NF-1-11-076.

References

Akrour, R., Schoenauer, M., and Sebag, M. (2011). Preference-based policy learning. In *Machine Learning and Knowledge Discovery in Databases*, pages 12–27. Springer.

Bongard, J. C., Beliveau, P., and Hornby, G. S. (2012). Avoiding local optima with interactive evolutionary robotics. In *Proceeding of the fourteenth annual conference on Genetic and evolutionary computation conference*, pages 1405–1406. ACM.

Bongard, J. C. and Hornby, G. S. (2013). Combining fitness-based search and user modeling in evolutionary robotics. In *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*, pages 159–166. ACM.

Kuzma, M., Jaksa, R., and Sincak, P. (2009). Clustering of users inputs in multi-user interactive evolutionary font design. In *Applied Computational Intelligence and Informatics, 2009. SACI '09. 5th International Symposium on*, pages 41–46.

LeBlanc, D. C. (2004). *Statistics: concepts and applications for science*, volume 2. Jones & Bartlett Learning.

Mouret, J.-B. (2011). Novelty-based multiobjectivization. In *New Horizons in Evolutionary Robotics*, pages 139–154. Springer.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Schmidt, M. and Lipson, H. (2011). Age-fitness pareto optimization. In *Genetic Programming Theory and Practice VIII*, pages 129–146. Springer.

Schmidt, M. D. and Lipson, H. (2006). Actively probing and modeling users in interactive coevolution. In *Proceeding of the eighth annual conference on Genetic and evolutionary computation conference*, pages 385–386. ACM.

Secretan, J., Beato, N., D'Ambrosio, D. B., Rodriguez, A., Campbell, A., and Stanley, K. O. (2008). Picbreeder: evolving pictures collaboratively online. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1759–1768. ACM.

Szumlanski, S. R., Wu, A. S., and Hughes, C. E. (2006). Conflict resolution and a framework for collaborative interactive evolution. In *AAAI*, pages 512–517. AAAI Press.

Takagi, H. (2001). Interactive evolutionary computation: fusion of the capabilities of ec optimization and human evaluation. *Proceedings of the IEEE*, 89(9):1275–1296.