Guarding Against Premature Convergence while Accelerating Evolutionary Search

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

Gregory S. Hornby
University Affiliated Research Center
UC Santa Cruz
NASA Ames Research Center, Mail Stop 269-3
Moffett Field, CA USA
gregory.s.hornby@nasa.gov

ABSTRACT

The fundamental dichotomy in evolutionary algorithms is that between exploration and exploitation. Recently, several algorithms [8, 9, 14, 16, 17, 20] have been introduced that guard against premature convergence by allowing both exploration and exploitation to occur simultaneously. However, continuous exploration greatly increases search time. To reduce the cost of continuous exploration we combine one of these methods (the age-layered population structure (ALPS) algorithm [8, 9]) with an early stopping (ES) method [2] that greatly accelerates the time needed to evaluate a candidate solution during search. We show that this combined method outperforms an equivalent algorithm with neither ALPS nor ES, as well as regimes in which only one of these methods is used, on an evolutionary robotics task.

Categories and Subject Descriptors

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

General Terms

Experimentation, Algorithms, Reliability

Keywords

Evolutionary Robotics, Premature Convergence, Evolutionary Algorithms

1. INTRODUCTION

Two of the most frequently-cited criticisms of evolutionary algorithms are that they become trapped in local optima, and that they are computationally inefficient in the sense that the majority of computation is spent on solutions that do not contribute to the final solution. This paper presents a method that incorporates and successfully combines two mechanisms for guarding against premature converge and minimizing the amount of time spent evaluating solutions destined never to produce offspring.

The first mechanism is an expanded version of the Age-Layered Population Structure (ALPS) algorithm, which partitions a population into several independent layers, and only allows an evolved solution to compete for breeding rights with other solutions in its layer. As ALPS proceeds, older and/or more fit solutions bubble into higher layers, and periodically the lowest layer is re-seeded with random genomes. ALPS is in this sense a diversity-maintenance method, many of which have been introduced into the literature, including preselection [4], crowding [5], deterministic crowding [15], sharing functions [7], and novelty emphasis [14, 16, 17, 20].

Where ALPS differs is that unlike other age-dependent algorithms [10, 11, 12, 13], the age of a genome is defined as the age since its original ancestor was created: an original ancestor is one that was created at the outset of the run, or during one of the periodic re-seedings of the lowest layer. This creates fairer competition between age groups, and allows for descendents of young ancestors to sometimes challenge, compete with and eventually unseat solutions derived from older ancestors which are mired on local optima.

The second mechanism is an early stopping method (not to be confused with the same term used in artificial neural networks) which terminates the evaluation of a solution early if it is guaranteed to not produce offspring even if it is evaluated fully. Early stopping has been used in the evolutionary robotics domain [18] in which solution evaluation is expensive, but these methods have been specific to the type of robot and task employed [22, 19, 3]; the early stopping method introduced here however is domain-independent.

The next sectcion introduces both the early stopping and ALPS methods used here, and how to combine them. Section 3 presents results demonstrating the complementarity of these two methods: both methods combined outperforms equivalent algorithms in which one or both methods are disabled. Section 4 provides discussion and some concluding remarks.

2. METHODS

This paper introduces a method that combines early stopping (Sect. 2.1), which minimizes the time required to evaluate sub-optimal solutions, and the Age-Layered Population Structure method (Sect. 2.2), which guards against premature convergence by simultaneously improving fit solutions and searching for novel, promising solutions. This combined method (Sect. 2.3) is used to evolve directed locomotion for a simulated legged robot (Sect. 2.4).

2.1 Early Stopping (ES)

Most evolutionary algorithms rapidly discover good solutions while the remainder of a run is spent incrementally improving them. However, random mutation and crossover events dictate that the vast majority of new solutions are less fit than their parents, and often do not reproduce: thus, a large fraction of time during an evolutionary run is wasted.

With a typical fitness function, fitness of a new solution i starts at zero and is added to during each time step of its evaluation

$$f(i) \quad = \quad \sum_{t=t_0}^{t_f} F(i,t) \tag{1}$$

(where $f(i)$ is the fitness of solution i, t is the current time step of the evaluation period, t_0 and t_f are the first and last time step, respectively, and $F(i,t)$ is a function that returns the fitness increment for solution i at time step t). Consequently, all steps of a fitness evaluation must be performed to determine whether or not this individual is worth keeping. In constrast, early stopping evaluates solutions by initially assigning each solution a maximum fitness value ($f\mathrm{max}$) and then gradually decrementing that value:

$$f(i) \quad = \quad f\mathrm{max} - \sum_{t=t_0}^{t_f} F(i,t). \tag{2}$$

In this way, once the fitness is reduced too low for the individual to be viable, evaluation can be halted.

In the applications reported here, multi-objective optimization is employed:

$$f(i,1) \quad = \quad f^{(1)}_{\mathrm{max}} - \sum_{t=t_0}^{t_f} F(i,1,t) \tag{3}$$

$$\ldots \tag{4}$$

$$f(i,j) \quad = \quad f^{(j)}_{\mathrm{max}} - \sum_{t=t_0}^{t_f} F(i,j,t) \tag{5}$$

where j fitness components $f(i,1), \ldots f(i,j)$ are calculated for each solution. *If during the evaluation of a new solution i it becomes dominated by an existing non-dominated solution k $((f(i,1) < f(k,1))\& \ldots \&(f(i,j) < f(k,j)))$, evaluation is terminated early.* If it is evaluated fully, it assumes a position on the non-dominated front and any solutions displaced from the front are deleted. By ensuring that $F(i,j,t) >= 0$ for all i, j and t, as solution i is evaluated it is ensured that $f(i,j)$ will monotonically decrease. Therefore, once a solution becomes dominated during evaluation it is ensured that it will never regain non-dominated status during the remaining time steps of its evaluation.

2.2 Age-Layered Population Structure (ALPS)

The age-layered population structure (ALPS) metaheuristic is an attempt to simultaneously improve existing good solution as well as continuous introduce less fit but promising solutions into the population. This is accomplished by dividing the total population into several layers (here, $L = 20$). In this work the steady-state version of ALPS is employed [9]. At each iteration a layer is chosen at random (Fig. 1, `ALPS_ES()`, line 2), and a non-dominated solution on that layer is chosen (line 10) for reproduction (line 11). The new solution j is evaluated using either ES (Sect. 2.1) or not.

The new solution j then attempts to displace a genome from its home layer i (`TryMoveUp`(i,j)) into the next highest age layer. A victim k is sought (`FindVictim`(i,j)) by first seeking a successful genome on layer i (lines 1-2); failing that, a genome that is too old for its layer is sought (lines 3-4); failing that, a solution is chosen that is dominated by the new solution (lines 5-6); failing that, no victim is returned.

In the application studied here, each robot is evaluated in several training environments (Fig. 2). A successful solution i is defined as one that achieves $F(i,j,k) = 0$ for each fitness component j at each time step k, so that $f_i^{(j)} = f_{\mathrm{max}}^{(j)}$ for all j. For the directed locomotion task (Sect. 2.4, Fig. 2), this equates to the robot approaching the target object in its environment with a sufficiently high mean velocity.

Each solution has an age, and each layer has a maximum age associated with it. In steady-state ALPS, the age of a solution i is set to $a_i = (s_c - s_i)/p$, where s_i indicates the number of solutions already evaluated at the time that its most senior ancestor[1] was created, s_c indicates the current number of solutions evaluated, and p indicates the population size (in this work, $p = 400$). The 20 layers used here were assigned maximum ages according to the Fibonacci series $1, 2, 3, 5, \ldots, f_k = f_{k-1} + f_{k-2}$ (following [8, 9]).

Once a victim is found, the algorithm attempts to move it in turn up into the next highest age layer (Fig. 1, `TryMoveUp`(i,j), lines 3,6). If during an attempted move the victim was successfully moved up a layer, the current solution takes its slot (`TryMoveUp`(i,j), lines 8,9). If a victim could not displace a solution in the layer above it, if it is dominated by the current solution it is deleted and replaced by the current solution (lines 13-15); if it instead dominates the current solution, the current solution is discarded (lines 11,12).

At the outset of each independent trial, for both tasks, one of 12 training environments is chosen at random. Robots are evaluated in this training environment until a successful genome is discovered (Fig. 1, `Move`(i,j,k), line 3). Once this occurs, the layer to which the successful genome belongs is expanded (line 4): a new training environment is selected at random; it is added to that layer's training set; all the genomes on that layer (including the successful genome) are evaluated against the new environment; new genomes arriving on this layer are evaluated against the original and new environment; and genomes arriving from younger layers that were only evaluated against the original environment (line 5) are evaluated against the new environment (line 6). This continues until a genome is discovered that succeeds in all 12 training environments, or the maximum allotted time expires.

As optimization proceeds, genomes migrate up from younger to older layers. In some cases their migration triggers expansion on the layer where they arrive, and sometimes they are themselves expanded. This eventually causes the layers to self-organize such that older layers host larger training sets than younger layers (see Fig. 4c). However, the successful solutions that originally triggered a layer expansion often migrate into higher layers. This means that younger and less experiened solutions are left to be evaluated against too many training environments, wasting computation time. To

[1] A solution's most senior ancestor is defined as its ancestor which was either created in the initial random population, or created during one of the re-seeding of the youngest layer (Fig. 1, `ALPS_ES()`, line 4 or 8).

combat this, compression is employed (`CompressLayers`()). After each solution is evaluated, the layers are scanned and if a layer is found to be compressible, it is compressed (`Compress-Layers`(), lines 2-4). A layer hosting k training environments is considered compressible if there is no solution in the layer that is successful on all of the first $k-1$ training environments. In such a case, the most-recently added training environment can be removed from the layer and new solutions on that layer are evaluated against the remaining $k-1$ training environments.

2.3 Combining ALPS and ES

In the two robot tasks investigated here, incremental shaping [23, 6, 21, 19] is employed to enable the evolving robots to perform successfully in an increasing number of training environments.

With multiple environments, the fitness function for each robot is

$$f(i,1) = \sum_{e=1}^{e_L}(f(i,e,1))/e_L \quad (6)$$

$$\ldots \quad (7)$$

$$f(i,j) = \sum_{e=1}^{e_L}(f(i,e,j))/e_L \quad (8)$$

$$f(i,e,1) = f_{\max}^{(1)} - \sum_{t=t_0}^{t_f} F(i,e,1,t) \quad (9)$$

$$\ldots \quad (10)$$

$$f(i,e,j) = f_{\max}^{(j)} - \sum_{t=t_0}^{t_f} F(i,e,j,t) \quad (11)$$

where $f(i,j)$ is the jth fitness component of robot i, e is the eth environment in which the robot was evaluated, e_L is the number of environments for which robots on layer L are evaluated against, and $f(i,e,j)$ is the jth fitness component of robot i evaluated in environment e.

When early stopping is employed, even if a new robot is to be evaluated against more than one environment, it may be terminated early if it performs poorly in some of the training environments. After a new robot has been evaluated in the first training environment, its fitness is estimated assuming that it will perform successfully in the remaining environments: this is accomplished by setting $F(i,e,j,t) = 0$ for all time steps t, fitness components j, and training environments $e > 1$, and then calculating all $f(i,j)$. If the robot becomes dominated by one of the robots on the Pareto front, the robot is not evaluated in the remaining training environments. If it remains non-dominated, the robot is evaluated in training environment $e = 2$; $F(i,e,j,t)$ is set to zero for all t, j and $e > 2$; $f(i,j)$ is recalculated and the process is continued until either the robot becomes dominated or it has been evaluated in all the available training environments.

2.4 Legged Locomotion

The ability of ALPS and ES to improve the probability of finding successful solutions in isolation and in combination was tested on the simulated robot shown in Fig. 2. This robot is composed of an upper and lower body and four legs. Each body part is attached to its connecting body part with an actuated, two degree-of-freedom (DOF) rotational joint. The first DOF of each joint sweeps the con-

```
procedure ALPS_ES()
 1: while !(SuccessfulOnAllObjs($j$) | TimeIsUp()) do
 2:   $i \leftarrow$ FindLayer()
 3:   if BottomLayer($i$) & $reinitializationMode$ then
 4:     $j \leftarrow$ CreateNewRandomGenome()
 5:   else
 6:     if BottomLayer($i$) & TooOld($i$) then
 7:       $reinitializationMode \leftarrow$ true
 8:       $j \leftarrow$ CreateNewRandomGenome()
 9:     else
10:       $parentIndex \leftarrow$ FindNonDominated($i$)
11:       $j \leftarrow$ CreateChild($parentIndex$)
12:       EvaluateChild($j$)
13:       TryMoveUp($i$,$j$)
14:     end if
15:   end if
16:   CompressLayers()
17: end while

procedure TryMoveUp($i$,$j$)
 1: if TopLayer($i$) then
 2:   $k \leftarrow$ FindVictim($i,j$)
 3:   TryMoveUp($i$,$k$)
 4: else
 5:   $k \leftarrow$ FindVictim($i+1,j$)
 6:   TryMoveUp($i+1$,$k$)
 7: end if
 8: if Empty(k) then
 9:   Move($i$,$j$,$k$)
10: else
11:   if Dominates($k$,$j$) then
12:     Discard($j$)
13:   else
14:     Discard($k$)
15:     Move($i$,$j$,$k$)
16:   end if
17: end if

procedure FindVictim($i$,$j$)
 1: if SuccessfulOnLayer($i$) then
 2:   return SuccessfulOnLayer($i$)
 3: else if TooOldOnLayer($i$) then
 4:   return TooOldOnLayer($i$)
 5: else
 6:   return DominatedOnLayerBySolution($i$,$j$)
 7: end if

procedure Move($i$,$j$,$k$)
 1: genomes[$k$] $\leftarrow$ genomes[j]
 2: $j \leftarrow k$;
 3: if Successful($j$) then
 4:   ExpandLayer($i$);
 5: else if TooLittleExperienceForLayer($i$,$j$) then
 6:   ExpandGenome($i$);
 7: end if

procedure CompressLayers()
 1: for $L = 1$ to maxLayers do
 2:   if Compressible($L$) then
 3:     CompressLayer($L$)
 4:   end if
 5: end for
```

Figure 1: Pseudocode for combining ALPS and ES.

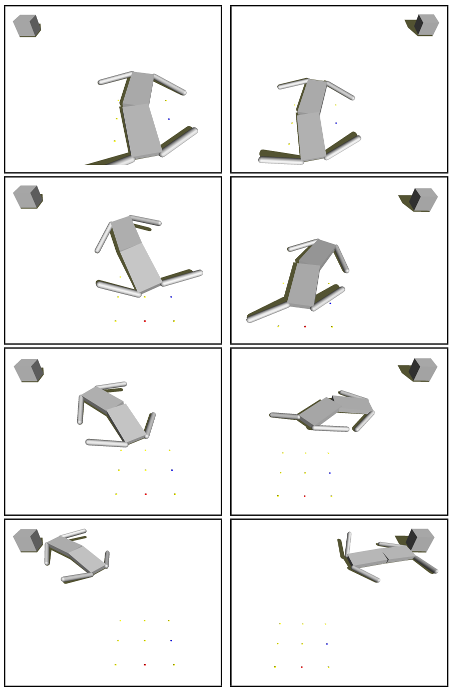


Figure 2: A sample evolved locomoting robot. A typical robot that has evolved to successfully approach objects placed at 12 different locations in an arc in front of it. The left and right columns show its behavior when the object is placed at the extreme left- and righthand endpoints of that arc, respectively.

necting parts through the robot's horizontal (sagittal) plane with a joint range of $[-50^o, 50^o]$; the second DOF of each joint sweeps the connecting parts through the robot's vertical (transverse) plane with a joint range of $[-20^o, 20^o]$. The robot contains four touch sensors, one in each leg. It also contains three distance sensors: one at the robot's midsection and two at its shoulders. The distance sensors return a value of 0 when the sensor overlaps the target object's center (an impossibility); a value of 1 when the sensor is at (or beyond) a defined maximum distance[2], and return an intermediate value between these distances.

The robot is controlled by a continuous time recurrent neural network (CTRNN; [1]). The robot contains 10 motor neurons, one for each DOF, and each is updated as follows:

$$\tau_i y_i' = -y_i + \sum_{j=1}^{10} w_{ji}\sigma(y_j - \theta_j) + \sum_{j=1}^{s_d+s_t} n_{ji} r_j \tag{12}$$

where τ_i is the time constant associated with neuron i, y_i is the value of neuron i, w_{ji} is the weight of the synapse connecting neuron j to neuron i, $\sigma(x) = 1/(1+e^{-x})$ is an activation function that brings the value of neuron i back into $[0, 1]$, θ_j is the bias of neuron j, $s_d = 3$ is the number of distance sensors, $s_t = 4$ is the number of touch sensors, n_{ji} is the weight of the synapse connecting sensor j to neuron i, and r_j is the value of sensor j. In this formulation, each sensor may have a direct effect on every motor neuron. However this effect may be minimized or eliminated by low values for r, or by behaviors that cause a motor neuron to saturate to extremal values. τ_i can range between $[0.0001, 1.0]$, w_{ji} in $[-16, 16]$, θ_i in $[-4, 4]$, and n_{ji} in $[-16, 16]$.

At the outset of each time step of an evaluation, the sensor values are retrieved from the physical simulator[3], one update of the CTRNN is made, and the resulting values of the motor neurons are calculated. The values are scaled to the minimum and maximum rotation angles of the corresponding joint, forming the desired angle for that joint. Torque is then applied to the joint commensurate with the difference between the joint's current angle and the desired angle. The positions and velocities of the objects in the simulation are then updated using a step size of 0.003.

This locomoting robot may be exposed to 12 different environments[4], in which the target object is placed at a different position on an arc in front of the robot: the left column in Fig. 2 illustrates the behavior of an evolved robot when the object is placed at the leftmost position on the arc (environment #1, defined in polar coordinates as $r = 1$ meter, $\theta = -45^o$), and the right column in Fig. 2 illustrates the behavior of the same robot when the object is placed at the rightmost position of the arc (environment #12, $r = 1$m, $\theta = +45^o$). The remaining 10 environments position the object at $\theta = (-45+1*(90/11))^o, \ldots, \theta = (-45+10*(90/11))^o$.

The fitness of a robot is defined as the rapidity with which it can approach the target object in each of the training environments to which it is exposed. This is achieved by selecting for CTRNNs that cause the robot to minimize the distance between its three distance sensors and the target object. Thus, for robot i, at each time step t of an evaluation in environment e, each of the j fitness components $f(i, j)$ is computed using

$$f^{(j)}_{\max} = 1.0, \tag{13}$$

$$F(i, e, j, t) = r_j^{(t)}/t_f, \tag{14}$$

where $r_j^{(t)}$ is the value of the jth distance sensor at time step t, and $t_f = 1000$ is the number of time steps per evaluation. This ensures that if the robot remains still, at the end of an evaluation $F(i, e, j, t) = 0$ for all fitness components j. A robot is considered successful in an environment if at the end of its evaluation all three fitness components $f(i, j) >= 0.5$.

3. RESULTS

One hundred independent trials of the combined ALPS-ES algorithm (Fig. 1) were conducted for the legged locomotion task (Sect. 2.4). Each trial was stopped if a robot was found that could successfully walk to each of the 12 object placements, or 24 CPU hours elapsed. In each trial 20 age layers were used[5], with 20 genomes per layer. Of the 100 trials, 32 finished successfully before the allotted time.

Fig. 3 shows the evolutionary progression of a typical run from among those that finished successfully. Fig. 3a indicates that over time the older layers (darker lines) tend to accumulate genomes with higher fitness than those on

[2]Here defined as the distance from the robot's midsection at the outset of an evaluation and the target object: 1 meter.

[3]`www.ode.org`

[4]The robot may be evaluated in less than 12 environments if early stopping is employed.

[5]The number of layers was chosen arbitrarily; the impact of this parameter setting on the running of the algorithm was not investigated here.

younger layers (lighter lines). Fig. 3b reports the number of non-dominated solutions on each layer. As the membership size of the Pareto front is often a good indicator of a population's variation, it can be seen that there seems to be little difference in the genetic variation on the various layers. Fig. 3c reports the mean number of objects in the training set for each layer. As indicated, older layers tend to host larger training set sizes than younger layers.

Layer 13 eventually discovers a successful robot; however, for several CPU hours previously this layer (and several others) host robots only capable of successfully approaching nine objects (denoted by the plateau at $y = 10$ in Fig. 3c). During this period layer 13 undergoes a large increase in genetic diversity, as indicated by the spike in non-dominated solutions around the seventh CPU hour (thick line in Fig. 3b). Despite this variation, during this period the fitness of the best genome on layer 13 is lower than those of older layers which also host training set sizes of 10 (the thick line is lower than the dark lines during CPU hours 6 to 8 in Fig. 3a).

This pattern exemplifies the power of the ALPS algorithm: layer 13 exports highly fit solutions to upper layers and is thus able to breed genetic variation, as well as inherit novelty from lower layers. This allows this layer to eventually discover a global optima. However, this advantage comes at a price: a significant portion of the population (layers 15-20) produce genomes that take a long time to evaluate, because the robots must be evaluated against up to 10 different environments; this search does not contribute to the final discovery of a local optima because genetic material is never passed down to younger layers. Early stopping helps to minimize this lost time by terminating the evaluation of most robots before they have been evaluated in all the training environments.

Fig. 4 reports the average characteristics of the 20 age layers over all 100 trials. Fig. 4a indicates that over time, older layers tend to inherit and breed genomes with higher fitness than younger layers. Fig. 4b shows that although more variable, older layers tend to host more non-dominated solutions than younger layers: this indicates that older layers tend to accumulate novelty from lower layers while preserving previously-discovered fit solutions. Fig. 4c indicates that older layers tend to host larger training set sizes than younger layers. This shows that the population tends to self-organize into a learning gradient: as successful genomes and their offspring migrate into older layers, they trigger expansions in the training set sizes of those layers; younger genomes then evolve against this learning gradient as they in turn migrate to older layers.

However, genomes evaluated against many environments can significantly slow down search. In order to determine whether early stopping can accelerate search, four additional experimental regimes were conducted: 100 independent trials were performed with only early stopping disabled; 100 trials with only ALPS disabled; and 100 trials with ALPS and ES disabled. In addition, a fourth regime was performed in which both ALPS and ES was used, but compression was disabled.

In the trials without ALPS, a single 'layer' with 400 individuals was evolved, and no new random genomes were ever introduced. In the trials without ES, each genome was evaluated in all k environments hosted by its resident layer. All other algorithmic details across the five regimes were kept the same. Fig. 5 reports the mean performance of all of the regimes. Clearly, early stopping provides a significant performance improvement (dark gray lines higher than light

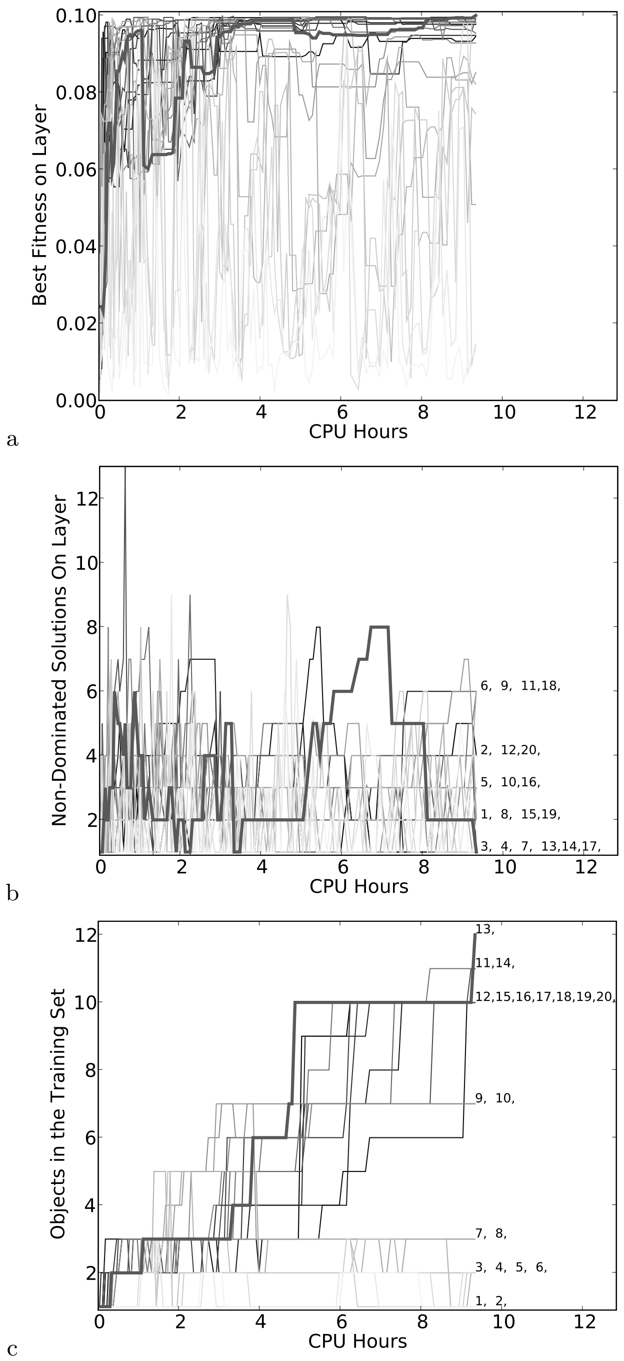
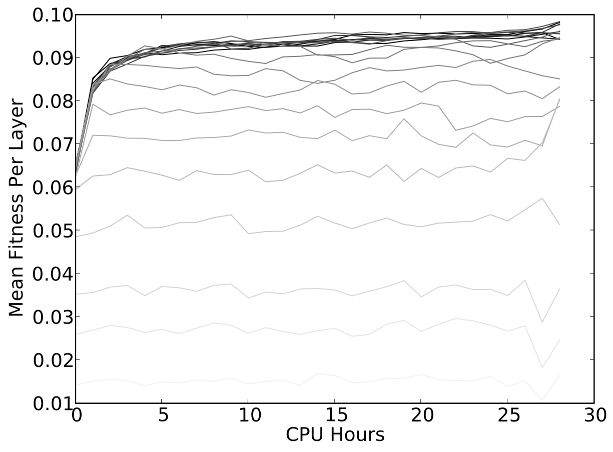
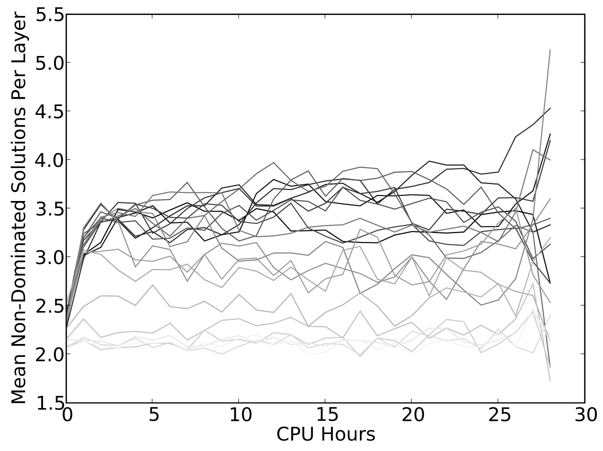


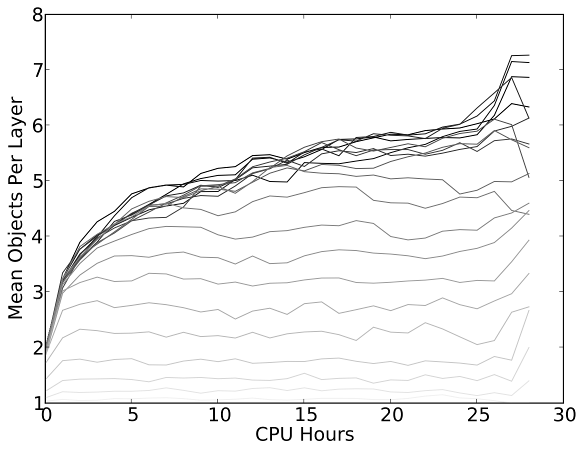
Figure 3: Evolutionary change in a typical run. (a) The fitness of the best genome on each of the 20 age layers. Lighter lines indicate younger layers. The thick line indicates the layer (13) which eventually produces a robot that can successfully walk to all 12 object placements after ∼9.5 CPU hours. (b) The number of non-dominated solutions on each layer. (c) The number of objects in the training set for each layer.



a



b



c

Figure 4: Mean evolutionary change over 100 runs. Lighter lines indicate younger layers. a: Mean fitness of all of the genomes on each layer. b: Mean number of non-dominated solutions on each layer. c: Mean number of objects in the training set for each layer.

gray lines). Moreover, ALPS provides a performance benefit regardless of whether ES is disabled (thick light gray line is higher than thin light gray line) or enabled (thick dark gray line is higher than thin dark gray line). When ES is enabled (dark gray lines), the regime with ALPS enabled initially lags behind the regime with ALPS disabled: this is

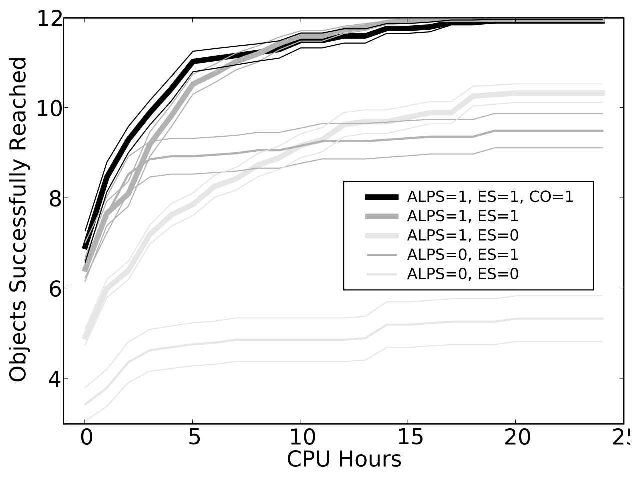


Figure 5: Relative rates of improvement using ALPS and Early Stopping. The vertical axis indicates the mean number of objects that have been successfully reached by a robot after that many CPU hours have elapsed. Thin lines indicate one unit of standard error of the mean.

because with ALPS disabled, all search focusses on evaluating genomes against a growing number of environments. Just before the third hour however, the combined ALPS-ES regime surpasses this regime as genomes in older layers trapped in local optima are replaced by genomes migrating up from younger layers.

Fig. 6 further illustrates the performance advantage of combinining ALPS and ES: Fig. 6a reports the fraction of successful runs that each experimental regime achieves. Fig. 6b reports the mean number of generations performed before a successful genome was found, averaged over all the successful runs for each regime. Finally, Fig. 6c reports the mean CPU time required for each regime to find a successful genome, averaged over all the successful runs. As can be seen, only when both ALPS and ES are employed do most of the runs finish successfully before the allotted time expires (Fig. 6a).

However, when either ALPS or ES is not employed, the few runs that do finish successfully finish on average earlier than when both ALPS and ES are employed (lower three lefthand bars in Fig. 6b). This can be explained as follows: if any run is fortunate to be seeded with a solution that is near a global optimum in terms of the mutations required to reach it, relatively little time is required to discover a descendent of this solution that occupies this optimum. However, if ALPS is not employed, those runs that do not initially contain such a solution become mired on local optima and do not finish on time. If Early Stopping is not employed but ALPS is, the longer time required to discover a random solution near a global optimum, percolate its descendants up through the layers to unseat non-optimal converged solutions and breed an eventual solution is longer than the allotted time, and the run terminates unsuccessfully.

Fig. 6c indicates that employing compression significantly improves the performance of runs that employ both ALPS and ES: although ALPS+ES runs with or without compression evaluate approximately the same number of solutions before finding a globally successful one (two rightmost bars

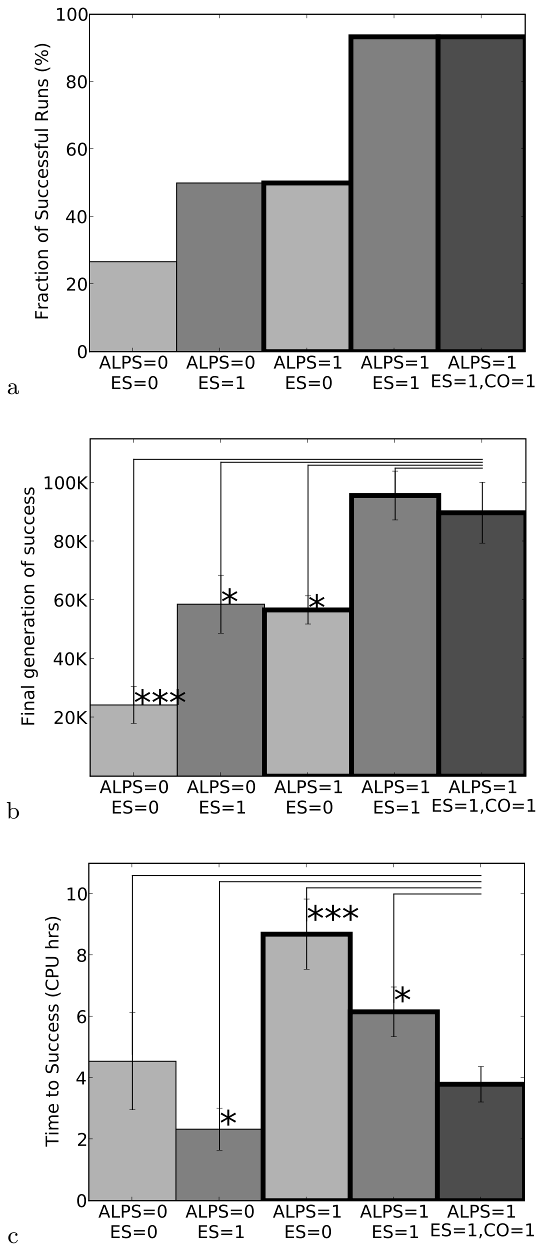


Figure 6: Relative performances using ALPS and Early Stopping. (a) Fraction of runs that finished within the alloted time. For those runs that finished successfully, the number of generations (b) and the number of CPU hours (c) that elapsed before the successful genome was found. The significance levels (*=$p < 0.05$; ***=$p < 0.001$) were determined using the Mann-Whitney U test.

in Fig. 6b), compression ensures that solutions on each layer are not evaluated against too many training environments, and therefore speeds up search.

The acceleration of search conferred by early stopping is shown in Fig. 3. Although steady-state ALPS [9] was employed here, the mean time to evaluate 400 genomes (the length of a ‘generation’) is reported as a function of the mean number of training set sizes across the layers. As expected, regardless of whether ES is employed or not, the time to evaluate one generation increases as the layers host increasing numbers of environments. However, this rate of increase is considerably curtailed when ES is employed: indeed the rate of increase is reduced from polynomial time to sub-linear time: regression produced a model with a positive nonlinear term when ES was disabled, but a negative nonlinear term when ES is employed. This indicates that as the number of training environments to which the robots are exposed increases, the time savings incurred using ES increases, compared to not using ES. This reduction in running time has previously been observed [2], but not when using ALPS.

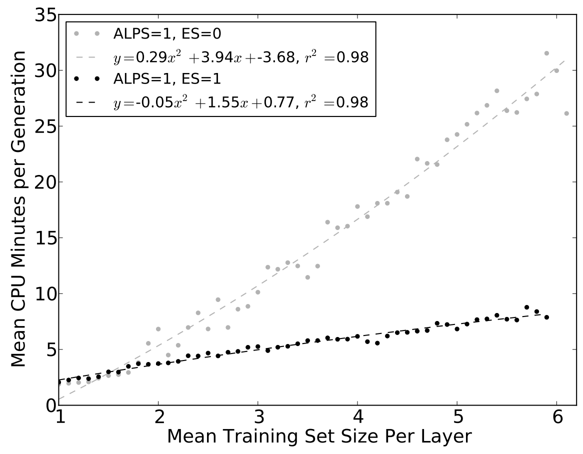


Figure 7: Time to evaluate a single generation when ES is disabled (gray dots) compared to when ES is enabled (black dots). Models of each growth rate were produced using nonlinear regression.

4. CONCLUSIONS

The crossing of the dark gray lines around the third CPU hour in Fig. 5 provides an example of the fundamental trade-off between exploration and exploitation in evolutionary algorithms. Without a mechanism to ensure continued exploration (ALPS = 0, thin dark gray line), there is rapid early improvement in the population as search focuses on improving the best solutions found so far. However, this often leads to a population becoming mired in local optima as individuals become more genetically homogeneous to one another. If a mechanism is employed to continuously introduce novelty into a population (ALPS = 1, thick dark gray line), the rate of improvement slows: search focuses not just on highly fit individuals but also on newer individuals who show promise. This investment in exploration may pay off after time however, as older genomes mired on local optima are displaced by the more-fit descendants of younger genomes who are still climbing gradients elsewhere in the search space.

This investment comes at a price however: evolutionary runs that balance exploration and exploitation must be continued for longer time periods to match and exceed the early performance of exploitation-heavy runs. For the locomotion task investigated here, the payoff in the exploration investment only began to yield dividends after the third CPU hour. As evolutionary algorithms mature and are employed to solve more complex problems that require long running times (such as evolutionary robotics [18]), they will require additional mechanisms to speed evolutionary search. This was accomplished here by integrating the exploration-exploitation mechanism of ALPS with a method that can quickly terminate evaluation of those new genomes that will not produce offspring, even if fully evaluated. It was shown that the combinations of these methods outperforms either method alone, or in equivalent regimes in which neither method was used.

This combined method has proven to work well on the computationally-intensive task of evolving behaviors for simulated robots in a three-dimensional physical simulator. However, it is anticipated that additional mechanisms will be required to handle deceptive tasks in which initial fitness improvement does not correlate with the eventual discovery of near-optimal solutions. Recently, a promising technique has been formulated [14, 16, 17, 20] that has proven to work well on deceptive tasks. In future work we plan to incorporate this mechanism into the method presented here, and demonstrate that the expanded method performs well on both complex and deceptive tasks, thereby further improving the scalability of evolutionary search.

Acknowledgements

This material is based in part upon work supported by the National Science Foundation under Grant No. 0757532.

5. REFERENCES

[1] R. Beer. Parameter space structure of continuous-time recurrent neural networks. *Neural Computation*, 18:3009–3051, 2006.

[2] J. C. Bongard. Innocent until proven guilty: Reducing robot shaping from polynomial to linear time. *IEEE Transactions on Evolutionary Computation*, 2010. In review.

[3] J. C. Bongard and C. Paul. Making evolution an offer it can't refuse: Morphology and the extradimensional bypass. In J. Kelemen and P. Sosik, editors, *Sixth European Conference on Artificial Life*, pages 401–412, 2001.

[4] D. J. Cavicchio. *Adaptive search using simulated evolution.* PhD thesis, University of Michigan, Ann Arbor, 1970.

[5] K. A. DeJong. *Analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, Ann Arbor, 1975.

[6] M. Dorigo and M. Colombetti. Robot Shaping: Developing Autonomous Agents Through Learning. *Artificial Intelligence*, 71(2):321–370, 1994.

[7] D. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49. L. Erlbaum Associates Inc. Hillsdale, NJ, USA, 1987.

[8] G. Hornby. ALPS: The age-layered population structure for reducing the problem of premature convergence. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, page 822. ACM, 2006.

[9] G. Hornby. Steady-state ALPS for real-valued problems. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 795–802. ACM, 2009.

[10] G. Hornby, M. Fujita, S. Takamura, T. Yamamoto, and O. Hanagata. Autonomous evolution of gaits with the Sony quadruped robot. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1297–1304, 1999.

[11] A. Huber and D. Mlynski. An age-controlled evolutionary algorithm for optimization problemsin physical layout. In *Proceedings of the 1998 IEEE International Symposium on Circuits and Systems*, pages 262–265. IEEE Press, 1998.

[12] J. Kim, J. Jeon, H. Chae, and K. Koh. A novel evolutionary algorithm with fast convergence. In *IEEE International Conference on Evolutionary Computation*, pages 228–29. IEEE Press, 1995.

[13] N. Kubota, T. Fukuda, F. Arai, and K. Shimojima. Genetic algorithm with age structure and its application toself-organizing manufacturing system. In *IEEE Symposium on Emerging Technologies and Factory Automation*, pages 472–477. IEEE Press, 1994.

[14] J. Lehman and K. Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *Proceedings of the Eleventh International Conference on Artificial Life (ALIFE XI) Cambridge MA: MIT Press*, volume 54, 2008.

[15] S. Mahfoud. Crowding and preselection revisited. In R. Männer and B. Manderick, editors, *Parallel Problem Solving From Nature*, pages 27–36. North-Holland, 1992.

[16] J. Mouret and S. Doncieux. Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity. In *Proceedings of the Congress on Evolutionary Computation*, 2009.

[17] J. Mouret and S. Doncieux. Using behavioral exploration objectives to solve deceptive problems in neuro-evolution. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 627–634. ACM, 2009.

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

[19] T. Reil and P. Husbands. Evolution of central pattern generators for bipedal walking in a real-time physics environment. *IEEE Transactions on Evolutionary Computation*, 6(2):159–168, 2002.

[20] S. Risi, S. Vanderbleek, C. Hughes, and K. Stanley. How novelty search escapes the deceptive trap of learning to learn. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 153–160. ACM, 2009.

[21] L. Saksida, S. Raymond, and D. Touretzky. Shaping robot behavior using principles from instrumental conditioning. *Robotics and Autonomous Systems*, 22:231–250, 1997.

[22] K. Sims. Evolving 3D morphology and behaviour by competition. *Artificial Life IV*, pages 28–39, 1994.

[23] S. P. Singh. Transfer of learning across sequential tasks. *Machine Learning*, 8(3):323–339, 1992.