GP-Sumo: Using Genetic Programming to Evolve Sumobots

Shai Sharabi
Department of Computer Science
Ben-Gurion University, Israel
E-mail: shshai@bgu.ac.il

Moshe Sipper
Department of Computer Science
Ben-Gurion University, Israel
E-mail: sipper@cs.bgu.ac.il
Web: www.moshesipper.com

Abstract

We describe the evolution—via genetic programming—of control systems for real-world, sumo-fighting robots—sumobots, in adherence with the Robothon rules: Two robots face each other within a circular arena, the objective of each being to push the other outside the arena boundaries. Our robots are minimally equipped with sensors and actuators, the intent being to seek out good fighters with this restricted platform, in a limited amount of time. We describe four sets of experiments—of gradually increasing difficulty—which also test a number of evolutionary methods: single-population vs. coevolution, static fitness vs. dynamic fitness, and real vs. dummy opponents.

1 Introduction

Evolutionary robotics is a field that deals with the use of evolutionary techniques to evolve autonomous or semi-autonomous robots, both in simulation and in the real world [2, 3, 6, 15, 19, 20, 24]. Often, the control system of an evolving robot is an artificial neural network, which receives one or more inputs from the robot's sensors and then outputs actuator controls. The network's topology—either fixed or evolving [25]—along with the learned synaptic weights and neuronal thresholds, represents the robot's "intelligence." Understanding such an intelligence is usually hard, due to the neural network's black-box nature.

In the present work we shall represent the robotic control system by a program, and evolve this program via genetic programming [12]. Genetic programming has the advantage of *inherently* evolving structure. Moreover, the evolved programs are usually more well structured than evolved neural networks. In genetic programming one starts with an initial set of general- and domain-specific features, and then lets evolution evolve the structure of the calculation (in our case, a sumo-fighter strategy). In addition, genetic programming may produce programs that can be simplified and understood (for example, Sipper *et al.* have recently evolved chess endgame players [10, 22], whose cognition they are now endeavoring to analyze).

This paper details the evolution—via genetic programming—of control systems for real-world, sumo-fighting robots—sumobots, in adherence with the Robothon rules (`www.robothon.org`): Two robots face each other within a dohyo (circular arena), the objective of each being to push the other outside the dohyo boundaries. As stated, genetic programming has the distinct advantage of evolving the crucially important structure of the control program. Our robots are minimally equipped with sensors and actuators, our intent being to seek out good fighters with this restricted platform (which can neither pull nor lift), in reasonable evolutionary time (a number of days). As for the latter time constraint, Ebner, e.g., experimented with real mobile robots, evolving corridor-following control architectures using genetic programming [6]. Due to the length of the experiment (2 months) on a real robot Ebner was able to perform only a single run.

We seek to answer: Can genetic programming be used to find a sumobot strategy given the limited amount of time available, using two real robots fighting each other? A successful result would evolve control programs that can push a mobile robot out of the dohyo. Evolving fighting strategies for real robots provides the field of evolutionary robotics with the opportunity to showcase how evolutionary computation discovers real-world robotic behaviors for this hard task. We also wish to demonstrate the suitability of genetic programming as a tool in the field of evolutionary robotics. Toward this end we perform four sets of experiments of increasing difficulty, which also test a number of evolutionary methods: single-population vs. coevolution, static fitness vs. dynamic fitness, and real vs. dummy opponents.

In the next section we present previous work on machine-learning approaches relevant to our sumobot research. In Section 3 we delineate the evolutionary setup. Section 4 details experimental results, followed by a discussion in Section 5. Section 6 concludes and describes future work.

2 Sumobots

Koza demonstrated early on the effectiveness of genetic programming in simulated evolutionary robotics [12]. In the intervening years evolution has been shown to produce a plethora of results both in simulated and real-world robotics [20]. As stressed by Nolfi and Floreano [20], an important aspect of such research is the emergence of complex abilities from a process of autonomous interaction between an agent and its environment. In our work we employed genetic programming to evolve and coevolve sumobots using two real robots, which interact in a dohyo.

An event that showcases the capabilities and technological developments in robotics in general and sumobot fighting in particular is the *National Robothon Event* (`www.robothon.org`), which takes place in Seattle once a year. There are two sumo-fight categories: Mini Sumo and 3Kg Sumo. Our own work aims at the 3Kg category and we thus made every effort to adhere to the rules of this division.

Sims [21] described a system that generates three-dimensional virtual creatures that exhibit competitive strategies in a physical simulated world. Although this work involved entirely simulated creatures (as opposed to our work involving real robots), its importance lies in the successful coevolution of one-on-one game strategies of robot-like creatures. Later on, Floreano and Mondada evolved neural networks [7] able to navigate "home," using real robots (Khepera) in real time, rather than simulation. Floreano and Nolfi used simulations of Khepera robots [8] to show that competing coevolutionary systems can be of great interest in evolutionary robotics.

Liu and Zhang [14] presented a multi-phase genetic programming approach for sumobot and compared it with a standard GP approach. The task was to evolve a control program that pushes a dummy stationary opponent of various shapes, weights, and postures. In the standard GP approach all the functions are available to the evolutionary process throughout. In the multi-phase approach, early phases involved general functions (e.g., move forward), with more specific functions introduced in later phases (e.g., fast, moderate or slow forward movement).

Liu's and Zhang's sumobots were evaluated first in simulation and the best ones evolved at each generation were then executed and validated on a physical robot, as opposed to our experiments where *all* robots were tested in the real world. Their multi-phase approach was shown to yield good results, faster than the conventional genetic programming approach, namely, it took GP about 20% more generations than the multi-phase GP approach to evolve a program that achieved the desired result. Our own GP system made all functions available throughout the evolutionary process but used the phase concept of introducing more demanding conditions at later generations, as will be described below.

The robot we used is extremely simple: It has two wheels, whose motors can be controlled separately, and no onboard sensors (Figure 1a). This robot is used by our mechanical engineering students and has the advantage of being very sturdy, thus affording itself to the bumping inherent in sumobot fights.

At the beginning of a fight the robots face each other as shown in Figure 1b. As the robots have no onboard sensors, sensation is performed via an overhead camera—one per robot, each connected to its own computer. Each camera relays the battlefield to its computer, which runs a sumobot control program. The control program decides upon the actuation commands to the wheel motors, which are then transmitted to the robot (Figure 1c). It may be argued that the use of overhead cameras provides the robot with global information about the arena, which is not realistic for autonomous robots with onboard sensors. We counterargue by noting further ahead that we limit the robot's use of visual data.

3 Evolving Sumobots using Genetic Programming

We used Koza-style genetic programming [12] to evolve sumobot control systems in the real world—with no simulation phase whatsoever. This has the obvious advantage of exhibiting no simulation-to-real-world transference problem [3, 6, 15]. The downside is the lengthy times involved in real-world evolutionary experiments. We were able to afford a relatively low number of runs—about ten evolutionary runs per experimental setup, each run taking on average 35 hours. This section delineates our evolutionary setup.

3.1 Functions and terminals

Our evolved individuals are expressions constructed from functions and terminals, detailed in Table 1. The *terminals* include the location of the two robots (Sumo1 and Sumo2),

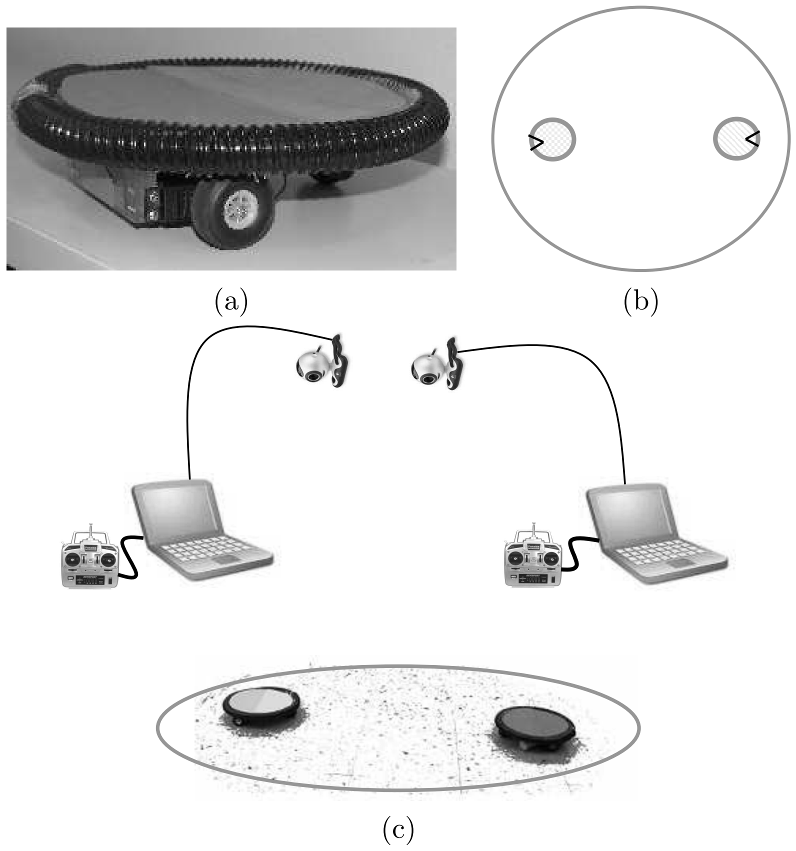


Figure 1: Sumobots: a) The robot. b) Dohyo (140cm in diameter) in initial setup, with two robots facing each other. The '<' symbols indicate the initial positions and orientations of the robots. c) System setup: Two overhead web cameras, each connected to its own computer. Each computer transmits via its remote controller the maneuver commands to the respective sumobot, which then acts accordingly.

Table 1: GP-Sumo: Functions and terminals.

Terminals	
$x1$	x position of Sumo1
$y1$	y position of Sumo1
$x2$	x position of Sumo2
$y2$	y position of Sumo2
$ERC1$	an ERC in the range [-15,15]
$ERC2$	an ERC in the range [-15,15]
α	angle between direction Sumo1 is facing and direction of Sumo2
Arithmetic functions	
$plus(x, y)$	$x + y$
$minus(x, y)$	$x - y$
$mul(x, y)$	$x \cdot y$
$sdiv(x, y)$	return x/y if y is nonzero; otherwise return 1
$abs(x)$	absolute value of x
$negative(x)$	$-x$
Motor functions	
$moveBW(x)$	Move both wheels in the same direction at speed x ($x > 0$ means forward movement, otherwise backward)
$moveRW(x)$	Move right wheel at speed x
$moveLW(x)$	Move left wheel at speed x
$spin(x)$	Move both wheels in opposite directions at speed x
$moveFree\_2(x, y)$	Move left wheel at speed x and right wheel at speed y
Logical functions	
$isl(x, y)\{block1\}$ else $\{block2\}$	if $x < y$ execute $block1$; otherwise, execute $block2$
$return\{func1\}$	*return* receives a motor function and returns control to main program

difference angle of Sumo1 vis-a-vis Sumo2, and two ephemeral random constants [12] (ERC1, ERC2). The terminal α is the angle between the direction Sumo1 is facing and the direction where Sumo2 is found. ERCs are randomly chosen integer constants in the range [-15,15] (maximal speed values of wheels). This value, once initialized, can change only through application of MutateERC.

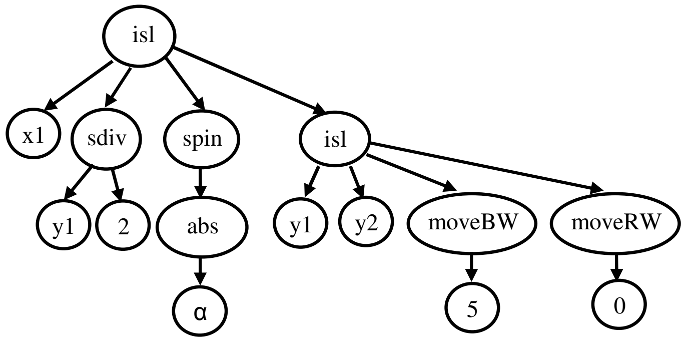
The functions belong to three categories (Table 1):

- Standard arithmetic functions.
- Motor functions, which set the velocities of the robot's two wheels.
- Standard logical functions.

An example of a random robotic control program from the initial population is given in

```
if(isl( x1 , sdiv( y1 , 2 ) ) ){
   return spin( abs( α ) );
}
else {
   if(isl( y1 , y2 ) ) {
      return moveBW( 5 );
   }
   else { return moveRW( 0 ); }
}
```

(a)



(b)

Figure 2: Sample random control program from generation zero: a) C language. b) Tree format.

Figure 2. The root level *isl* function has four sub-branches: one terminal, one arithmetic function, one motor function, and one logical function. The code can be read as follows: if Sumo1's x position is less than the result of dividing Sumo1's y position by 2, then spin the robot at a speed equal to the absolute value of the angle by which Sumo1 has drifted from Sumo2; otherwise, perform another check: if Sumo1's y position is less than Sumo2's y position then move forward at a speed of 5, else rotate right at speed 0, i.e., stand still. This program yields a behavior that depends on the robots' location.

We used strongly typed genetic programming [18] (STGP), which allows to add data types and data-type constraints to the programs, so as to avoid the formation of illegal programs under genetic operators. For example, *moveBW(x)* has an argument x which cannot be replaced by *spin(x)* (i.e., *moveBW(spin(x))* is not a valid expression). We thus need to ensure the safe combination of functions and terminals in the initial population of programs, and when crossover and mutation take place. STGP does just this: By pre-checking the type of the requested input and choosing it from the correct pool of functions and/or terminals we make sure that no type error will emerge from a genetic operation. Our control programs conform to the following STGP rules:

1. An arithmetic function accepts *number type*, meaning arguments that are either arithmetic functions or terminals, and its return type is *number*.

2. A motor function accepts *number type* arguments and its return type is *motor function.*

3. The *isl logical function* accepts predicates that are *number type* arguments, and accepts *logical functions* type in the block1 and block2 sections. It returns a boolean value.

4. The *return logical function* accepts as argument a *motor function.*

3.2 Fitness function

To drive evolution to find good sumobot fighters we followed the Robothon basic rules (`www.robothon.org`). A robot is given three tries (fights) to push the other robot out of the ring. Points are given for exhibiting aggressive gestures during the fight (e.g., approaching the opponent, fast maneuvering, etc'—see full details below). A bonus is given to a robot that manages to push its opponent outside the dohyo. Since we are dealing with evolution and no behavior should be taken for granted we also reward players for not leaving the dohyo during the fight.

We used coevolution with two independently evolving populations [1,4,9,11], each containing programs that controlled one of the two robots. This affords better diversity during evolution, since each population contains a separate genetic pool. We deemed that two such pools were better than one (as in single-population algorithms). A positive side-effect of our coevolutionary setup is that every evolutionary run yields, in effect, two (usually differently behaving) best robots.

We began our experiments with a simple fitness function, which rewards the sumobot for: 1) approaching the target, 2) pushing it, and 3) staying inside the dohyo. This function has the advantage of being straightforward and simple, but the unfortunate disadvantage of producing bad results; specifically, the sumobots evolved to move extremely slowly, i.e., they were much too overcautious for any serious fighting to occur. Further experimentation ultimately resulted in the more complex, yet viable, fitness function, given below:

$$w_1 \cdot \mathit{radius} + w_2 \cdot \mathit{sticky} + w_3 \cdot \mathit{closer} + w_4 \cdot \mathit{speed} +$$
$$w_5 \cdot \mathit{push} + w_6 \cdot \mathit{bonuspush} + w_7 \cdot \mathit{explore} + \mathit{bonustay}.$$

The fitness function is calculated after the game ends using the two robot-route arrays of robot position triplets. A position triplet, $\{x_i^t, y_i^t, \theta_i^t\}$, represents the location and orientation of Sumo$_i$ ($i = \{1,2\}$) at time t. The weights $w_1 \ldots w_7$, empirically derived, allowed us to finetune the requirements of the evolved individuals. We tested different weight values, most of which yielded interesting results.

As for the fitness components: Let $d_{i,j}^{u,v}$ be the distance between robot i at time u and robot j at time v, computed as:

$$d_{i,j}^{u,v} = \sqrt{(x_i^u - x_j^v)^2 + (y_i^u - y_j^v)^2}.$$

The arena radius is 170 image pixels (70 cm in reality), and all distance units are in pixels.

Let T be the number of iterations—*game ticks*—an evolved robot executes in a single fight (basically, a measure of fight time). Robot i's fitness components are computed as follows:

- *radius* rewards distance between starting position to robot's farthest location during a fight (larger distance is better):

$$radius = \max_{t=2..T} \{d_{i,i}^{t,1}\}.$$

- *sticky* rewards spending time close to the target, namely, the other robot, denoted j:

$$sticky = \frac{1}{T}\sum_{t=1}^{T} isS(t),$$

  where $isS(t) = 1$ if $d_{i,j}^{t,t} < \delta$, and $isS(t) = 0$ otherwise. δ is a pre-set threshold distance.

- *closer* rewards approaching the target in leaps larger than δ:

$$closer = \frac{1}{T}\sum_{t=2}^{T} isC(t),$$

  where $isC(t) = 1$ if $d_{i,j}^{t-1,t} - d_{i,j}^{t,t} > \delta$, otherwise $isC(t) = 0$.

- *speed* rewards speed,

$$speed = \frac{1}{T}\sum_{t=2}^{T} d_{i,i}^{t,t-1}.$$

- *push* rewards pushing the opponent:

$$push = \frac{1}{T}\sum_{t=2}^{T} \{isS(t) = isC(t) = isD(t) = 1\},$$

  where D is the dohyo's center, and $isD(t) = 1$ if $d_{j,D}^{t,1} > d_{j,D}^{t-1,1}$, which indicates that the opponent robot retreats from the dohyo's center; otherwise $isD(t) = 0$. $d_{j,D}^{t,1}$ denotes the distance of robot j at time t from the dohyo's center. In the *push* equation, 1 is added to the sum when the condition is true, 0 otherwise.

- *explore* rewards exploring the dohyo (more area explored is better). This component returns the number of robot positions that are distant more than δ pixels from each other.

$$explore = |E|,$$

  where we initialize $E = \{1\}$ and add to this group as follows:

$$E = \{t \ : \ d_{i,i}^{t,\tau} > \delta, \ t = 2..T, \ \forall \ \tau \in E\}.$$

Table 2: Summary of the four batches of experiments carried out.

Batch	**Fitness**	**Opponent**	**Evolution**	**Description**
A	static	dummy	single population	evolve a robot to successfully push a dummy
B	static	real	coevolution	coevolve sumobots starting from Batch A's results
C	dynamic	real	coevolution	coevolve sumobots with fitness function changing during evolution
D	static	fixed opponents	single population	evolve sumobots pitted against pre-designed opponents

- *bonustay* is a fixed bonus added for staying in the dohyo:

$$bonustay = 20.$$

- *bonuspush* rewards faster wining programs. It is calculated by dividing the fixed 40 seconds allocated for a game by the time the game actually lasted.

$$speed = \frac{40}{T}.$$

We began our experimentation with a simple scenario that allows evolution to "ease into" the problem, and then gradually increased the task's complexity. *In toto*, we ran four different sets of experiments. In the first set (Batch A) the robots evolved to push an *immobile*, "dummy" robot out of the dohyo. Only one population was evolved here. The second batch of experiments (Batch B) used the evolved population of the first batch as an initial population for coevolving robots that push a *mobile* robot (i.e., a real fight). Both populations in this coevolutionary scenario used robots from the Batch-A experiment. The third set of experiments (Batch C) used a dynamically changing fitness function that was adjusted during the coevolutionary run. The last batch of experiments (Batch D) tested the effectiveness of evolving sumobots by pitting them against our own pre-designed robots. A summary of the four experimental setups is given in Table 2.

3.3 Breeding strategy

After the evaluation stage we create the next generation of programs from the current generation. This process involves selection of programs from the current generation and application of genetic operators to them. The resulting programs enter the next generation.

Specifically, using *rank* selection to select our candidates we then apply the following standard breeding operators [12] on the selected programs:

1. Unary reproduction: Copy programs to the next generation with no modification, to preserve a small number of good individuals.

2. Binary crossover: Randomly select an internal node in each of the two programs and then swap the sub-trees rooted at these nodes.

Table 3: Control parameters for genetic programming runs.

Population size	20
Generation count	7 – 57
Selection method	rank
Reproduction probability	0.2
Crossover probability	0.8
Mutation probability	0.05 – 0.1
Elitism group	2

3. Unary mutation: Randomly select a node in a tree program and replace the sub-tree rooted at this node with a newly grown tree.

Using rank selection, 18 programs of the 20 in the population were selected. Each pair from the selected programs was either crossed over (with probability 0.8) and transferred to the next generation or copied to the next generation unchanged. Finally, this new population underwent mutation with a small probability. The remaining (2) programs were simply the two best programs passed along unchanged—the elitism group [17]. The control parameters were found empirically through several months of runs and are summarized in Table 3. We used rank selection to abate possible premature convergence that might occur in such a small population with other forms of selection (e.g., fitness-proportionate).

3.4 Diversity measures and the dominance tournament

To prevent premature convergence we employed rank selection (rather than fitness proportionate) and mutation. In order to gain insight into whether diversity is indeed maintained we used *pseudo-isomorphs* [5], which measure population diversity changes throughout evolution. Pseudo-isomorphs are found by defining a triplet of $< terminals, functions, depth >$ for each individual, and the number of unique triplets in a population is its diversity measure. Two identical triplets represent trees with the same number of terminals, functions, and depth which might be isomorphic. Each comparison of two triplets yields a binary value, 0 or 1.

Another diversity measure that indicates a difference between two trees, yet returns a real value, is the Tanimoto difference [16]. The idea is based on a ratio of counted subtrees of two individuals. Denote by S_{i_1} the set of all subtrees that appear in program i_1. The set has cardinality (size) $|S_{i_1}|$. The Tanimoto tree difference is defined as:

$$d_t(i_1, i_2) = \frac{|S_{i_1} \bigcup S_{i_2}| - |S_{i_1} \bigcap S_{i_2}|}{|S_{i_1} \bigcup S_{i_2}|} \quad ,$$

and the Tanimoto population diversity measure, for a population of size n, is defined as:

$$D_t(i_1, i_2, ..., i_n) = \frac{2}{(n-1)} \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} d_t(i_j, i_k) \quad , \quad n > 1.$$

This measure can discern small structural differences that pseudo-isomorphs cannot. On the other hand, isomorphic trees are not considered equivalent by the Tanimoto measure. Thus, we chose to employ both measures.

The *dominance tournament* [23] measure indicates whether any progress is made during *co*evolutionary experiments. Once a coevolutionary experiment is over and we have the best of generation (BOG) of both robots available we alternately add new dominance strategies per robot. A new robot dominance strategy must defeat all previous dominance strategies of the opposing robot. We start by pitting BOG 0 (best of generation 0) of both robots, the winner being set as the first dominance strategy. The next BOG strategy of the losing robot that defeats this strategy becomes the first dominance strategy of that robot. The second dominance strategy of the first winning robot is its next BOG that defeats the opposing dominance strategy—and so on. Each new dominance strategy defeats all previous ones (of the opposing robot). The dominance tournament measure is the number of strategies that were found during this tournament, for each robot.

4 Results

This section describes the results of the four batches of experiments appearing in Table 2. We ran 10 experiments per batch (as noted above, each one taking on average about 35 hours). Results in this section are described for a typical run (per batch), to underscore specific robotic capabilities that have evolved. In the next section we shall discuss our observations over all experiments.

4.1 Batch A: Static fitness, single population, dummy opponent

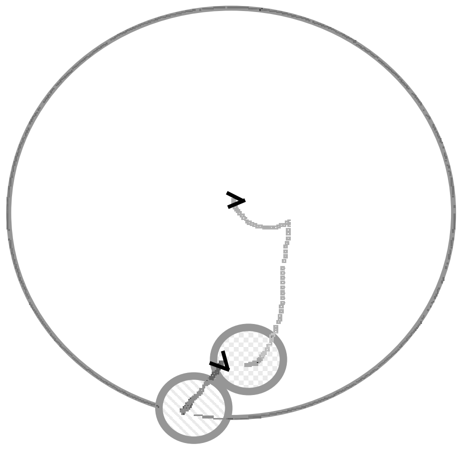
In this experiment we evolved sumobots to push a dummy robot. The initial fight setup included a dummy immobile robot placed at a random position along the border of the dohyo, and an evolving sumobot positioned at the center of the dohyo.

We witnessed the emergence of several winning strategies, as shown in Figure 3. Two prominent behaviors that evolved are: 1) The sumobot rotates until $\alpha < -11$ and then approaches the target, pushing it in more then 50% of the fights (if $-180 < \alpha < -20$ the sumobot runs straight and has no contact with the other robot); 2) another control system goes through a wide circling route that manages to push the target in more than 30% of the times, since it covers over 1/3 of the dohyo circumference.

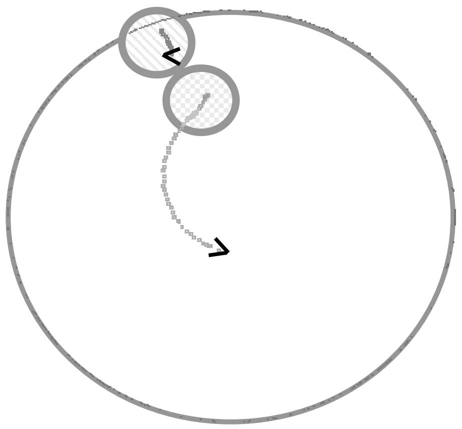
4.2 Batch B: Static fitness, coevolution, real opponent

Now that we have evolved a number of sumobot strategies that are able to cope successfully with an immobile target, we will use one of our evolved populations as an initial population for true sumo-fight evolution, i.e., with two fighting players. Each evolutionary run in this batch of experiments lasted 2-9 days.

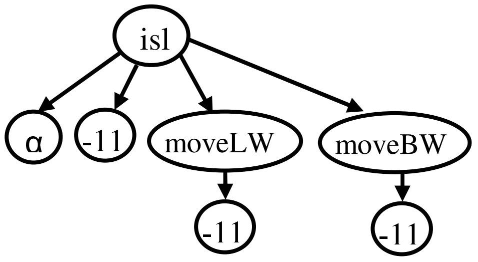
In batches B and C an evolved program might not be able to exhibit its potential if its evolved opponents keep leaving the dohyo. To resolve this problem we decided that any player spontaneously leaving the dohyo is scored according to its performance, while its



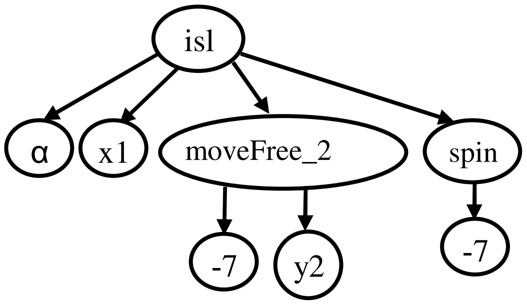
(a)



(b)



(c)



(d)

Figure 3: Batch A: Sumobot against dummy. The '<' symbols indicate the initial positions and orientations of the robots, and the line tracings show the robots' movements during the fight. a) Evolved fighter rotates toward the target and approaches it if starting from a position where $\alpha < -11$. b) Evolved fighter circles widely unconditionally. c) Simplified program of robot (a). d) Simplified program of robot (b).

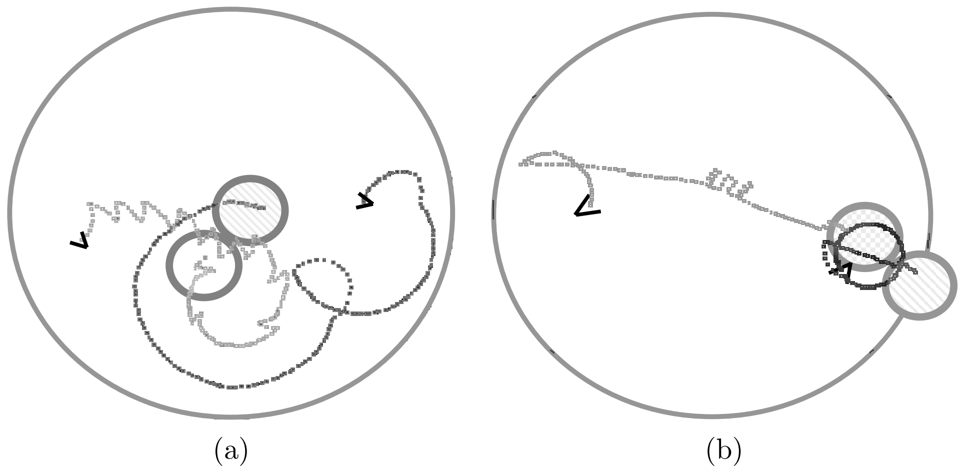


Figure 4: Batch B: Example of typical coevolved sumobots. a) Demonstration of a fight between two individuals, from generation 44 of Sumo1 and generation 39 of Sumo2. b) Sumo1 (on the left) evolved to face its opponent and approach it. The zigzags midway through Sumo1's route were caused by some offset (directional difference) in facing its opponent and trying to hastily rectify its direction.

opponent will not be scored and will be entitled to a rematch. This configuration means the opposing populations evolve asynchronously. A population of players that stay in the dohyo accumulates more rematches against opponents that leave the dohyo. Thus, a player of the first population might face opponents of more than one generation. For example, an evolved individual of generation 44 fought against an individual of generation 39, as in Figure 4a.

Examining the fitness plots in Figure 5 it is not evident that there was any progress in coevolving better sumobot fighter strategies. Using the dominance tournament to measure the progress of this typical coevolutionary run reveals that most of the progress is accomplished during the first few generations (although some progress occurs later). For better accuracy each pair played 10 games and the total score of each was compared. The robot dominance strategies, as shown in Figure 5a, indicate 4 levels of dominance, in generations 0,1,4, and 31. The opposing population (Figure 5b) produced its counter dominance strategies in generations 2,5,6, and 7, which were mainly spinners. In this run the first robot had difficulties overcoming the spinner strategy.

Another evolved sumobot (Figure 4b), which successfully handled spinner strategies, emerged at generation 8 of a different run. As depicted, this robot first rotates to face its opponent and then approaches it. It tends to be hasty when rectifying its orientation as can be seen midway through the fight. This usually happens when the robots are closer.

A batch-B run starts from an initial evolved population (of batch A), thus being susceptible to premature convergence more than the other batches. Utilizing the pseudo-isomorph and Tanimoto diversity measures (e.g., the run shown in Figure 5) reveals that diversity can be maintained (note that the "higher" the graphs, the more diverse is the population).

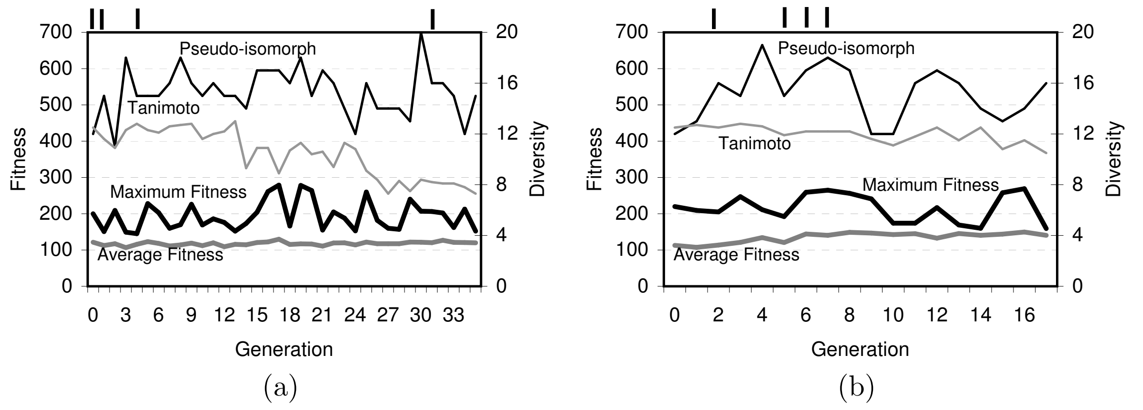


Figure 5: A typical run of Batch B. a) Dominance tournament results are represented as tick marks, and as can be seen there are 4 levels of dominance. Also shown are Tanimoto measure, pseudo-isomorph measure, BOG, and average fitness. b) Same as (a) for opponent. Population size is 20.

4.3 Batch C: Dynamic fitness, coevolution, real opponent

In this section we coevolved sumobots by changing the fitness function midway through evolution. Evolution starts with a general desired maneuver, like staying and exploring the dohyo. When this is (hopefully) achieved we changed the weights to evolve more specific, harder-to-achieve features, like pushing and fast-wining strategies.

The fitness function we used for this batch is similar to the one described in Subsection 3.2 above, with two additions: 1) we added a counter-*closer* component, named *retreat*, which subtracts points if the sumobot retreats (we shall use this component for Batch D as well); and 2) we added a fixed *leaving* penalty for sumobots that leave the dohyo. The fitness function thus becomes:

$$\begin{aligned}&w_1 \cdot \mathit{radius} + w_2 \cdot \mathit{sticky} + w_3 \cdot \mathit{closer} + w_4 \cdot \mathit{speed} + w_5 \cdot \mathit{push} + \\ &w_6 \cdot \mathit{bonuspush} - w_7 \cdot \mathit{retreat} + w_8 \cdot \mathit{explore} + w_9 \cdot \mathit{bonustay} - w_{10} \cdot \mathit{leaving}.\end{aligned}$$

The weights of the fitness function were adjusted to afford each component with a specific range of values. After the first ten generations (by which time sumo-fighting strategies had usually evolved) a change in the fitness function—as described in Figure 6d—was effected. For example, the *bonuspush* weight was increased by 30%, while *explore* was reduced by 66%. Thus, a "mature" sumobot gets less credit for exploring the dohyo and more for pushing its opponent out.

Observing the progress of the control systems in a typical run revealed that Sumo1 (Figure 6a, left) exhibited some effective approach maneuvers in early generations. However, with the course of evolution this behavior disappeared and a different strategy evolved. Sumo1 developed fast spinning as long as its opponent was far; when its opponent approached, which it always did after several generations, Sumo1 used various other evolved fast maneuvers (Figure 6b). These newest behaviors of Sumo1 managed to push its opponent once in a while (Figure 6c), thus collecting enough points to supplant all other strategies.

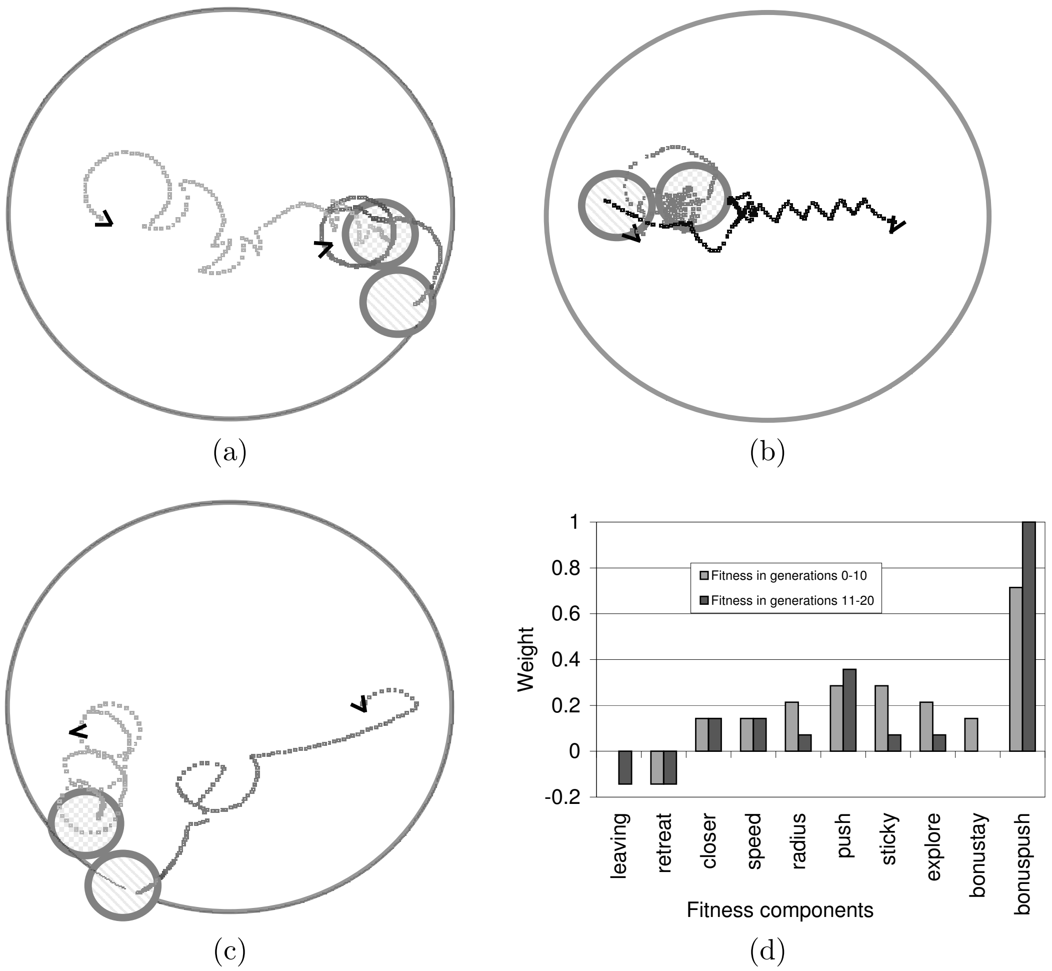
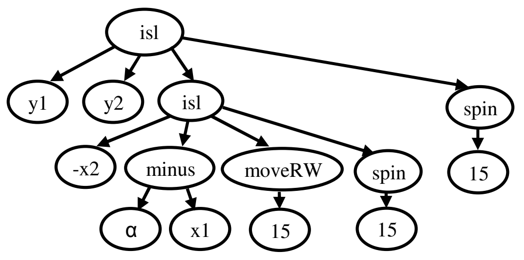
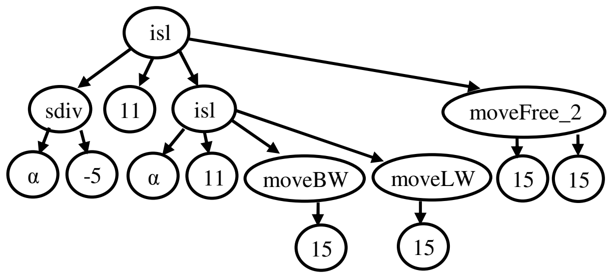


Figure 6: Batch C: Example of typical coevolved sumobots. a) A fight at generation 3. b) A fight at generation 7. c) Left robot scored a so-called Yuko at generation 13. This is the highest possible score, given when one contender manages to push its opponent out of the dohyo. d) The dynamic fitness computation we used: Halfway through the run (at generation 11) the fitness weights were adjusted as depicted in the figure.



(a)



(b)

Figure 7: Batch C: Simplified evolved programs of robots shown in Figure 6c. a) Sumo1 exhibits mostly spinning behavior. b) Sumo2 exhibits approaching behavior.

By generation 13 most of the programs of Sumo2 (right sumo) exhibited approaching behaviors. This reinforced Sumo1's fast spinning maneuver that "lured" its opponent to the edge, then to push it out (Figure 6c). The simplified code for the robots in this last fight is given in Figure 7.

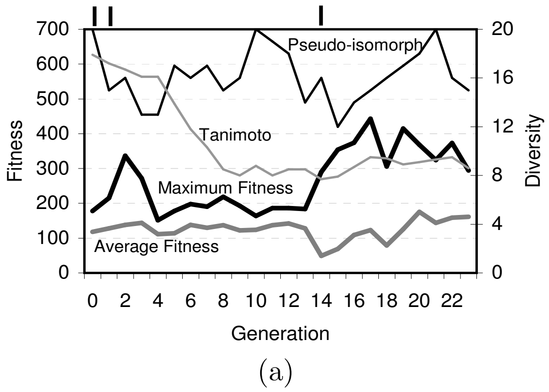
Generally, the behavioral progress of the evolved sumobots is more pronounced at early generations. Using dominance tournament and scoring based on the second-half fitness weights (Figure 6d), we can see that Sumo1 (Figure 8a) produced 2 dominance strategies at generations 0 and 1, and its last winning dominance strategy at generation 14. Sumo2 (Figure 8b) produced counter dominance strategies at generations 2 and 3. Sumo2 continued producing better strategies than BOG 3 (i.e., scored higher against Sumo1's dominance strategies), but none could win all the 3 dominance strategies of Sumo1, and thus did not become one of Sumo2's dominance strategies. Fight samples are shown in Figure 9.

Here, as for batch B, the pseudo-isomorph and Tanimoto diversity measures (Figure 8) indicate that our population maintained relative diversity. Both diversity measures suggest a modestly declining trend.

4.4 Batch D: Static fitness, single population, pre-designed opponents

Finally, we evolved sumobots by pitting them against three specially designed sumobot programs: Spinner, Bashful, and Pusher. Each evolving robot played against all three of these. The first program—Spinner—spins fast, thus motivating the evolution of approach behaviors, which are rewarded via the *explore, sticky,* and *closer* components of the fitness function. The second program—Bashful—was designed to avoid contact, thereby motivating the evolution of chasing techniques. The third program—Pusher—was designed to push the other robot, giving the evolved sumobot a real fight.

This experiment yielded various control systems. In one run, the evolved sumobot persistently chased its opponents in zigzag rotations (Figure 10). In this run, whose fitness progress is depicted in Figure 10d, we noticed an interesting trend. A routine examination of the robot before recharging it revealed that its left wheel torque capability had diminished. Checking its fitness progress we could estimate that this problem started at generation 12, evidenced by the sudden fitness drop at this point. Evolution, as evident in the graph, led



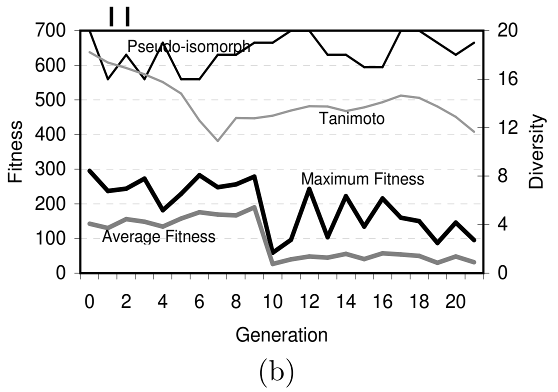
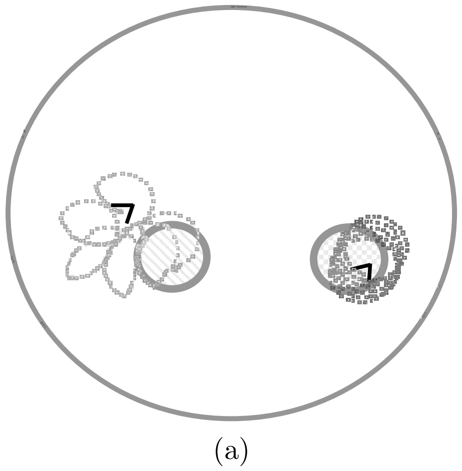


Figure 8: A typical run of Batch C. a) After 13 generations the fitness computation changed as described in Figure 6d. Dominance tournament results, represented as tick marks, show 3 levels of dominance. Also shown are Tanimoto measure, pseudo-isomorph measure, BOG, and average fitness. b) The corresponding opponent's fitness was changed after 9 generations. The opponent produced 2 levels of dominance strategies.



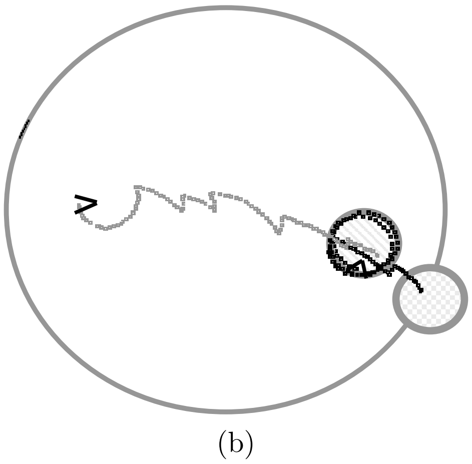


Figure 9: Batch C: Example of dominance strategies of coevolved sumobots. a) Sumo2's (on the right) dominance strategy of generation 3 wins in points its "flower shaper" opponent of generation 11. b) Sumo1 (on the left) evolved the last (and best) dominant strategy.

Table 4: Summary of sumobot tournament. Yuko: A win wherein the robot pushes its opponent out of the dohyo. Points: The fitness *push* component.

Contender	**Yuko**	**Points**
D	23	441
C	19	349
B	18	366
Pusher	7	445
Spinner	5	281
Bashful	3	202

to adaptation strategies that dealt with this mechanical failure.

In another typical run, diversity (Figure 11) was maintained as in the previous batches. The Tanimoto diversity measure indicates faster convergence than the pseudo-isomorph measure.

4.5 Post-evolutionary sumo championship

After performing the evolutionary runs we decided to test our evolved robots in an all-out tournament. The tournament takes place as follows: Each batch of experiments (B, C, D) "contributes" its top 20 sumobot fighters, i.e., the 2 top-fitness robot controllers of each of the 10 experiments per batch. We now have six player categories: B, C, D—each comprising 20 players—and Spinner, Bashful, and Pusher—each comprising the single (hand-crafted) player by that name. Each category plays (fights) 40 games against each other category (the total number of games per category is thus 200). For category B, C, and D each fight involves a player selected at random from the respective category. Each category's match involves 20 games wherein the individuals were loaded onto one robot and another 20 games wherein the same individuals were loaded onto the other robot. This equalizes the advantage one platform might have over the other. The results of the tournament are given in Table 4.

Category *D* won the top record with 23 Yukos. Categories *C* and *B* follow closely, and the hand-crafted programs are way behind. Individuals B20 (program 20 of batch B), shown in Figure 12b, and D9 (program 9 of batch D), shown in Figure 12a, won the highest score with 4 Yukos each. However, D9 takes the lead with 28 points vs. 13 points of B20. Category D, which was the one evolved against the 3 hand-crafted programs, seems to beat the other categories. The wins distribution of the 3 batches is shown in Figure 12. Batch B (Figure 12b) is shown to yield fewer winning sumo fighters while Batch C (Figure 12c) yields the most.

4.6 Is evolution better than randomness?

Observing the resulting robots' trajectories and programs it may be argued that these evolved individuals are somewhat simplistic, and that perhaps randomly created controllers would work just as well. We counterargue this by performing the following experiment: First, note that a typical evolutionary run involves a population of 20 individuals run for 20 generations, which amounts to 400 evaluated robot controllers. So we pitted our 60

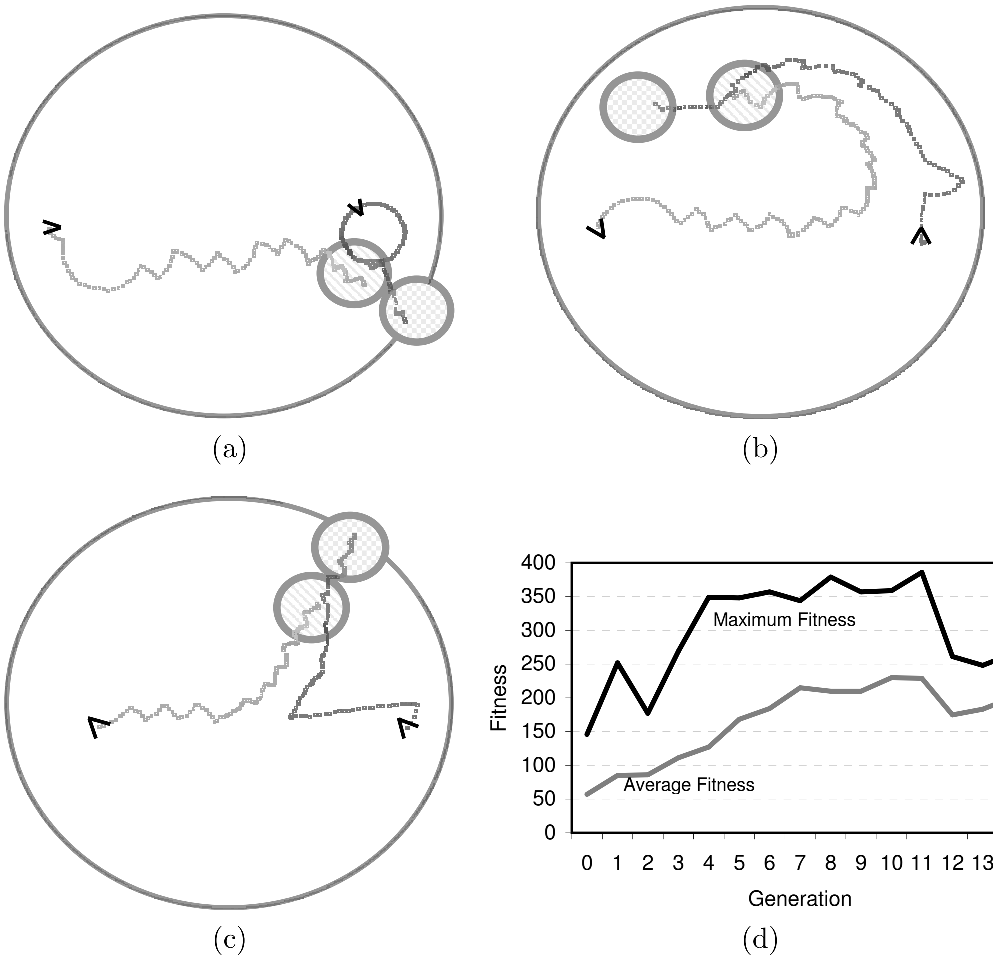


Figure 10: Batch D: Evolving sumobots pitted against pre-designed opponents. a) Evolved bot (initially at left) fights Spinner and scores a Yuko. b) Evolved bot (initially at left) fights Bashful. c) Evolved bot (initially at left) fights Pusher and scores a Yuko. d) Fitness graph of run that produced the evolved sumobot. The drop of fitness at generation 12 was due to a mechanical failure in the robot wheels. Nevertheless, evolution overcame this problem and slowly produced adapting programs.

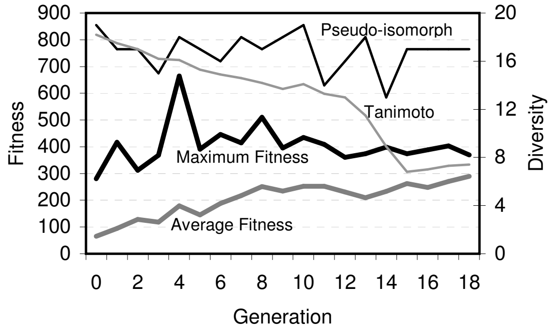


Figure 11: Batch D: Diversity and fitness measures in a typical run.

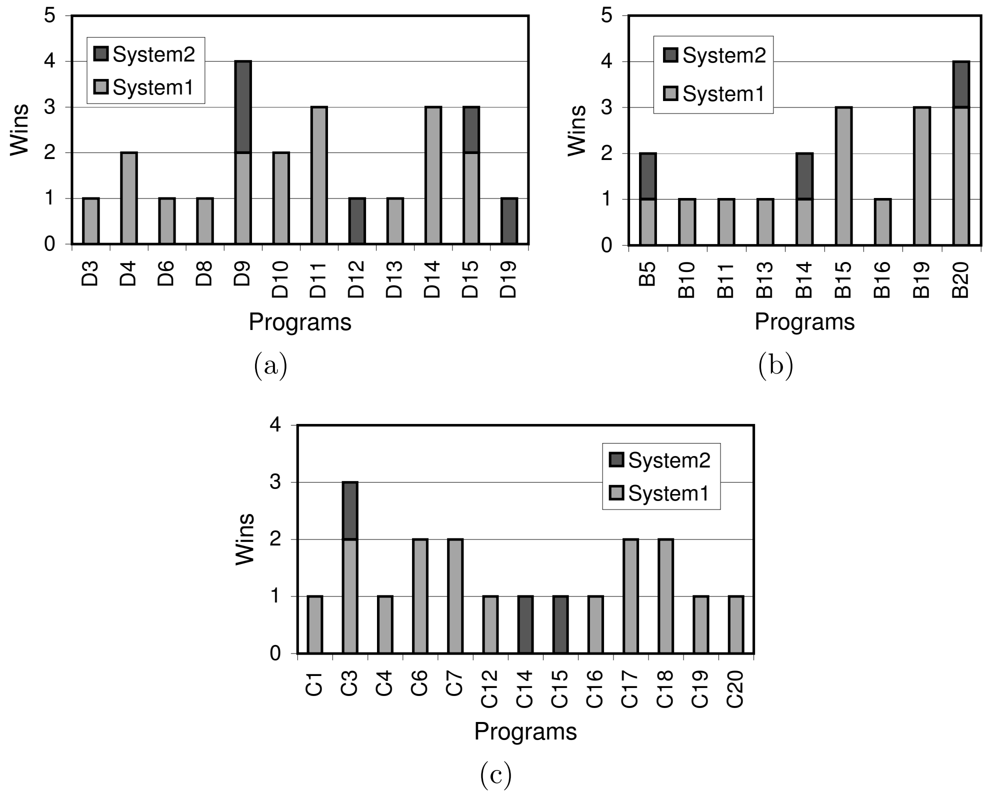


Figure 12: Tournament results by evolved controller. System1 and System2 indicate Yukos that were achieved when loaded onto the first or second robot, respectively. a) Batch D: Combines both best score and highest number of winning sumobot fighters. b) Batch B: Has few winning sumobot fighters, but evolved second best sumobot fighter. c) Batch C: Has the widest range of winning sumobot fighters.

top evolved controllers of batches B, C and D against 1020 random individuals, i.e., over twice the number evaluated during a standard evolutionary run. The random programs were generated using *ramped-half-and-half* [12] with a depth of 8. The results were quite convincingly in our evolved bots' favor: they won seven times more Yukos than the random programs—140:21. Clearly, evolution—even with a relatively small number of evaluated individuals (due to time constraints)—presents a huge advantage.

Finally, we pitted the 30 best random controllers (the 21 that scored a Yuko and another 9 with the highest scores) in a rematch against the 60 evolved ones. Each random robot of the 30 played 6 games against an evolved one of the 60 (thus each one of the latter played 3 games). The evolved sumobots won two times as many Yukos as the top random ones. Close inspection of the random programs that won Yukos revealed that most of them were spinners, while a few others shot straight across, uncaringly bumping into whatever came across their route. We conclude that in comparison to random bots evolution yielded better sumobots using less resources.

5 Discussion

We usually evolved chasing maneuvers relatively early on—after less than a dozen generations. In most cases the control system that evolved rotated the robot toward the target or toward the opposite side and then advanced forward or backward, respectively, approaching the target and sometimes pushing it. Another commonly evolved sumobot, emerging especially through coevolution, was a fast spinning robot able to push its opponent outside when the latter approached.

We observed a variety of sumobot programs that evolved using the same basic fitness function (perhaps with minor modifications)—all of which behave well. In some runs good sumobots evolved at an early evolutionary stage but were replaced later on by better control systems.

Fitness, as evidenced in the graphs, seems to oscillate. This is probably due to the real-world nature of these experiments, which involves noise from the camera and robots' accuracy. In addition, the robots do not start the fight from the same exact place and orientation, and the coevolutionary experiments introduced random opponents, thus increasing the variability of the fitness calculation for each program. Some control programs exhibited high fitness spikes at early evolutionary stages. Examining these programs revealed that pure chance was often at hand: Some robots moved straight and fast, and their unfortunate opponents simply happened to cross their path, resulting in a fast win.

In most of our evolutionary runs behavioral convergence of the population was ultimately observed. The individuals of the final generations behaved similarly. For example, one experiment produced clockwise and counter-clockwise spinners, while another experiment produced mainly approachers. The reason for this behavioral convergence is probably the small population size, which was necessary due to the lengthy run times (as noted earlier, we wanted to perform many runs that took days, rather than few runs that would take months). The genotypic diversity measures we applied indicated that structural diversity within the population was maintained. Thus, even though behaviors seem to converge, the structures do not. Phenotypic diversity measures that would reflect behavioral diversity rely mainly on fitness, which is inappropriate here, as two distinct strategies could score the same fitness, while the same exact individual often scores differently even against the same opponent, as often happens between people in real life.

Due to the extreme simplicity of our robot—only able to push—it would be unfair to compare it with more elaborate (though non-evolved) robots, e.g., ones that can lift their opponent, or have better friction control (the only other sumobot evolutionary work, that of Liu and Zhang [14], described in Section 2, also discusses their not comparing evolved robots with others). In the end one must remember that our intent herein has been to answer (positively, if possible) the question posed in the introduction: Can genetic programming be used to find a sumobot strategy given the limited amount of time available using real robots?

6 Concluding Remarks

We showed several evolutionary methods for evolving sumobot fighter strategies for real robots using genetic programming. In future work we plan to take into consideration other parameters, which might help improve our sumobots yet further: dimension and center of dohyo, location of robots in previous iteration—essential if we want to take the robots' *real speed vector* into account. Using a different robotic platform might yield other interesting results. Finally, automatically defined functions (ADFs) [13], might also be worth examining.

Acknowledgements

We are grateful to Dario Floreano, Ami Hauptman, Yonatan Shichel, and the anonymous reviewers for their careful reading of this manuscript and helpful comments. We thank Yaroslav Tenzer for his help in building the robots and writing their drivers.

References

[1] A. Ronge and M. G. Nordahl. Genetic programs and co-evolution: Developing robust general purpose controllers using local mating in two-dimensional populations. In H. M. Voigt, W. Ebeling, I. Rechenberger, and H. P. Schwefel, editors, *PPSN IV: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, pages 81–90, London, UK, 1996. Springer-Verlag.

[2] G. J. Barlow. Design of autonomous navigation controllers for unmanned aerial vehicles using multi-objective genetic programming. Master's thesis, North Carolina State University, Raleigh, NC, 2004.

[3] P. Chongstitvatana. Improving robustness of robot programs generated by genetic programming for dynamic environments. *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS '98)*, pages 523–526, 1998.

[4] E. H. Østergaard and H. H. Lund. Co-evolving complex robot behavior. In A. M. Tyrrell, P. C. Haddow, and J. Torresen, editors, *Evolvable systems: From biology to hardware, fifth international conference, ICES-2003*, volume 2606, pages 308–319, Odense M., Denmark, 2003. Springer-Verlag.

[5] S. Gustafson E. K. Burke and G. Kendall. Diversity in genetic programming: An analysis of measures and correlation with fitness. *Evolutionary Computation, IEEE Transactions on*, 8(1):47–62, 2004.

[6] M. Ebner. Evolution of a control architecture for a mobile robot. In D. Mange M. Sipper and A. Pérez-Uribe, editors, *Proceedings of the Second International Conference on Evolvable Systems: From Biology to Hardware (ICES 98)*, volume 1478, pages 303–310, London, UK, 1998. Springer Verlag.

[7] D. Floreano and F. Mondada. Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics*, 26(3):396–407, 1996.

[8] D. Floreano and S. Nolfi. God save the red queen! competition in co-evolutionary robotics. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 398–406, Stanford University, CA, USA, 1997. Morgan Kaufmann.

[9] D. Floreano, S. Nolfi, and F. Mondada. Competitive co-evolutionary robotics: From theory to practice. In J. A. Meyer R. Pfeifer, B. Blumberg and S. W. Wilson, editors, *R. Pfeifer, From Animals to Animats V: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, pages 515–524, Cambridge, MA, 1998. MIT Press.

[10] A. Hauptman and M. Sipper. GP-EndChess: Using genetic programming to evolve chess endgame players. In M. Keijzer, A. Tettamanzi, P. Collet, J. van Hemert, and M. Tomassini, editors, *Proceedings of 8th European Conference on Genetic Programming (EuroGP2005)*, volume 3447 of *Lecture Notes in Computer Science*, pages 120–131. Springer-Verlag, Heidelberg, 2005.

[11] J. R. Koza. Genetic evolution and co-evolution of game strategies. In *Proceedings of the International Conference on Game Theory and Its Applications*, Stony Brook, New York, 1992.

[12] J. R. Koza. *Genetic programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge MA, 1992.

[13] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge Massachusetts, 1994.

[14] J. Liu and S. Zhang. Multiphase genetic programming: A case study in sumo maneuver evolution. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(4):665–684, 2004.

[15] M. Mataric and D. Cliff. Challenges in evolving controllers for physical robots. Technical Report CS-95-184, Brandeis University, Computer Science Department, Waltham, MA, 1995.

[16] C. Mattiussi, M. Waibel, and D. Floreano. Measures of diversity for populations and distances between individuals with highly reorganizable genomes. *Evolutionary Computation*, 12(4):495–515, 2004.

[17] J. Miller and P. Thomson. Aspects of digital evolution: Geometry and learning. In A. Pérez-Uribe M. Sipper, D. Mange, editor, *ICES '98: Proceedings of the Second International Conference on Evolvable Systems*, volume 1478, pages 25–35. Springer-Verlag, London, UK, 1998.

[18] D. J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.

[19] A. L. Nelson, E. Grant, and T. C. Henderson. Evolution of neural controllers for competitive game playing with teams of mobile robots. *Robotics and Autonomous Systems*, 46(3):135–150, 2004.

[20] S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, intelligence, and Technology of Self-Organizing Machines*. MIT Press, Cambridge MA, 2000.

[21] K. Sims. Evolving 3D morphology and behavior by competition. In R. Brook and P. Maes, editors, *Artificial Life*, volume IV, pages 28–39. MIT Press, 1994.

[22] M. Sipper, Y. Azaria, A. Hauptman, and Y. Shichel. Designing an evolutionary strategizing machine for game playing and beyond. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, (to appear).

[23] K. Stanley and R. Miikkulainen. The dominance tournament method of monitoring progress in coevolution. In A. Barry, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002) Workshop Program*, pages 242–248. San Francisco, CA: Morgan Kaufmann, 2002.

[24] C. Teuscher, E. Sanchez, and M. Sipper. Romero's odyssey to Santa Fe: From simulation to real life. In M. Jamshidi, A. A. Maciejewski, R. Lumia, and S. Nahavandi, editors, *Robotic and Manufacturing Systems: Recent Results in Research, Development and Applications*, volume 10 of *TSI Press Series: Proceedings of the World Automation Congress*, pages 262–267. TSI Press, Albuquerque, NM, 2000.

[25] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.