Virtual Bagging for an Evolved Agent Controller

Bobby D. Bryant, *Member, IEEE*

***Abstract*— An evolved agent controller is used to produce multiple votes for its choice of actions by presenting it with rotations and reflections of its actual sensory inputs. Since the agent's behavior with respect to the orientation of its various sensors must be learned independently, the votes can be treated as independently learned opinions on the optimal choice of actions, i.e. a form of Breiman's *bagging*. The mechanism is tested on neural controllers for agents trained by neuroevolution in a game-like simulator, and is found to improve their task performance as well as ensuring perfect symmetry of behavior with respect to orientations of the environment.**

Keywords: Agents, Multi-Agent Systems, Adaptive Team of Agents, Games, Simulators, *Legion II*, Behavior, Sensors, Symmetries, Neuroevolution

I. Introduction

Bagging – a portmanteau word for *bootstrap aggregation* – refers to the use of voting among independently trained pattern classifiers to improve classification rates [1]. Such voting helps because independent training of the classifiers results in different weak spots for each; where one produces a wrong classification, others may produce the correct one. When a majority of the voters produce the correct classification on a given pattern the errors of the minority are negated, and if this is a sufficiently common occurrence the classification rate for the aggregate improves upon the rates of its individual participants.

Intuition suggests that what works for pattern classifiers should also work for agent controllers. Indeed, for agents that choose between discrete actions a controller can be viewed as a type of pattern classifier: features of the system state observed by the agent are "classified" according to the most appropriate action to be taken when that state is encountered. Learning control becomes a special case of a learning classifier system, and many of the techniques developed for the latter – such as bagging – can be expected to be immediately useful for the former.

Intelligent agents in games and simulators often operate in geometric environments subject to reflections and rotations. For example, a two dimensional map can be reflected across an explorer agent or rotated about it, providing a new and different but still plausible map. Similarly, the visible universe can be reflected or rotated on any of the three axes of a robotic construction worker in deep space. A well trained general purpose agent for deployment in such environments should be able to operate equally well in a given environment and its symmetric transformations. In general it is desirable for intelligent agents to exhibit symmetrical behavior as well. That is, if the optimal action in a given environment is to move to the left, then the optimal action in a mirror image of that environment would be to move to the right.

Thus symmetry of behavior is desirable for two independent reasons. First, if a correct or optimal move can be defined for a given context, failing to choose the symmetrical move in the symmetrical context will be sub-optimal behavior, and will degrade an agent's overall performance if it ever encounters such a context. Second, if the agent operates in an environment observable by humans, such as a game or a simulator, the humans will expect to see "visibly intelligent" behavior, i.e., they will expect the agent to always do the right thing because it is smart, rather than intermittently doing the right thing because it has been programmed or trained to manage only certain cases [2].

If an agent's controller operates by mapping sensory inputs onto behavioral responses, the desired symmetries can be identified by analyzing the structure of the agent and its sensors. For example, if the agent and its sensors are both bilaterally symmetrical then it will be desirable for the agent's responses to be bilaterally symmetrical as well. However, if they are not symmetrical – e.g. for a construction robot with a grip on one side and a tool on the other – then its optimal behavior is asymmetrical. Thus the desirable symmetry of behavior depends critically on the symmetry of the agent and its sensors, as well as on potential symmetries in the environment.

When agents are trained by machine learning methods there is not, in the general case, any guarantee that they will learn behaviors that are invariant across rotations and reflections: the behaviors with respect to the orientation of its sensors must be learned independently. In earlier work the symmetry of behavior for agents trained by supervised learning was improved by using rotations and reflections to extend a set of training examples [3]. Overall performance improved as well, but it was shown that most of the improvement could be credited to the mere increase in the number of training examples, rather than to the fact that those examples were symmetrical transformations of the original set.

In the work reported here, symmetries are exploited to produce bagging for evolutionary reinforcement learning. A controller is trained for agents that operate in an environment that supports multiple symmetries of reflection and rotation. Whenever an agent must make a decision, rotations and reflections of its actual sensory input are processed by the agent's controller in addition to that actual input, and the controller's choice of actions for each case is un-rotated or un-reflected and then counted as a vote. This results in a type of bagging: since the agents' behaviors with respect to its various possible orientations are learned independently,

The author is on the Computer Science and Engineering faculty at the University of Nevada, Reno (mail: University of Nevada, Reno/171, Reno, Nevada, USA 89557; phone: +1-775-784-6974; e-mail: bdbryant@cse.unr.edu).

 In *Proceedings of the IEEE 2010 Conference on Computational Intelligence and Games (CIG'10)*, pp. 99-106. Piscataway, NJ: IEEE Press.

their controllers do not reliably produce invariant decisions for symmetrical transformations of their inputs. Voting can exploit these independent evaluations for the controller just as in bagging for a learning classifier system. However, since only a single controller is actually in use, which is virtualized into additional "independently trained" controllers via the presentation of rotated and reflected inputs, the method is called *virtual* bagging (VB).

This mechanism addresses both of the motivations for symmetrical behavioral invariants, i.e. it improves both the agents' task performance and the symmetry of their responses to their inputs. Unlike the supervised learning method described previously, virtual bagging ensures absolute consistency of behavior with respect to sensory inputs: any input is a member of an equivalence class of rotations and reflections, each of which will produce exactly the same set of votes, and thus an invariant choice of behaviors with respect to the environment. Performance-wise, virtual bagging provides the best results yet obtained by the various methods that have been applied to the game used for testing it.

The learning environment used to test virtual bagging, and the structure of the agents' sensors and controller, are described in the following section. Then the learning mechanism, experimental methodology, and experimental results are presented in section III. Those results are discussed in section IV, along with a look at future directions for the research.

II. The Learning Environment

Virtual bagging was tested in a game/simulator called *Legion II*, a discrete-state strategy game designed as a test bed for multi-agent learning problems. The game requires a group of legions to learn to contest the control of a province against a horde of barbarian warbands. *Legion II* has been used previously for experiments in real-time adaptation of behavioral roles in a team [4], stochastic control to reduce the predictability of game agents [5], inductive behavioral modelling [6], and the exploitation of sensor symmetries in supervised learning, as described above [3]. *Legion II* is described briefly below; see [2] for a full treatment.

A. The Legion II game/simulator

The *Legion II* game/simulator is played on a map that represents a province of the Roman empire, complete with several cities and a handful of legions for its garrison (figure 1). Gameplay requires the legions to minimize the pillage inflicted on the province by a steady stream of randomly appearing barbarian warbands. The barbarians collect a small amount of pillage each turn they spend in the open countryside, but a great deal each turn they spend in one of the cities.

The game is parameterized to provide enough legions to garrison all the cities and have a few left over, which can be used to disperse any warbands they find prowling the countryside. The original purpose of this parameterization was to require the legions to learn a dynamic division of labor between garrisoning the cities and patrolling the countryside, in a multi-agent cooperative architecture called an *Adaptive Team of Agents* (ATA) [4]. The game is used here to test virtual bagging, because it is a challenging learning task that offers multiple symmetries in its environment. Though the specific choice of multi-agent architectures is orthogonal to the evaluation of virtual bagging, the ATA continues in use here, and is enforced by the simple expedient of using the same controller for each of the legions in play.

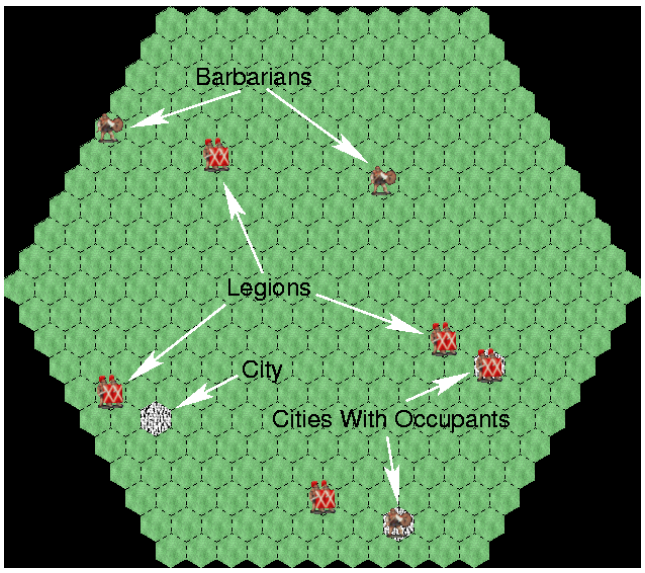

Fig. 1. **The *Legion II* game.** A large hexagonal playing area is tiled with smaller hexagons in order to discretize the positions of the game objects. Legions are shown iconically as close pairs of men ranked behind large rectangular shields, and barbarians as individuals bearing an axe and a smaller round shield. Each icon represents a large body of men, i.e. a legion or a warband. Cities are shown in white, with any occupant superimposed. All non-city hexes are farmland, shown with a mottled pattern. The game is a test bed for multi-agent learning methods, whereby the legions must learn to contest possession of the playing area with the barbarians. (An animation of the *Legion II* game can be viewed at `http://nn.cs.utexas.edu/keyword?ATA`.)

The *Legion II* map is in the shape of a large hexagon, divided into small hexagonal cells to discretize the placement of the game objects: legions, barbarians, and cities (figure 1). Moves are taken in sequential turns. During a game turn each legion makes a move, and then each barbarian makes a move. All moves are atomic, i.e. during a game agent's move it can either elect to remain stationary for that turn or else move into one of the six hexagons of the map tiling adjacent to its current position.

Only one agent, whether legion or barbarian, can occupy any map cell at a time. A legion can bump off a barbarian by moving into its cell as if it were a chess piece; the barbarian is then removed from play. Barbarians cannot bump off legions; they can only hurt the legions by running up the pillage score. Neither legions nor barbarians can move into a cell occupied by one of their own kind, nor can they move off the edge of the map. Either type of mobile agent can move into the stationary cities.

A game is started with the legions and cities placed at random positions on the map; the combinatorics allow a vast number of distinct game setups. The barbarians enter play

at random unoccupied locations, one per turn. If the roving legions do not eliminate them they will accumulate over time until the map is almost entirely filled with barbarians, costing the province a fortune in goods lost to pillage.

Play continues for 200 turns, with the losses to pillage accumulated from turn to turn. At the end of the game the legions' score is the amount of pillage lost to the barbarians, rescaled to the range $[0, 100]$ so that the worst possible score is 100. Lower scores are better for the legions, because they represent less pillaging. The methods described in this paper allow the legions to learn behaviors that reduce the score to around 3 or 4 when tested on a random game setup never seen during training, i.e. to reduce pillage to about 3%-4% of what the province would have suffered if they had sat idle for the entire game.

The barbarians are programmed to follow a simple strategy of approaching cities and fleeing legions, with a slight preference for the approaching. The are not very bright, but such behavior suits the needs of the game and perhaps approximates the behavior of barbarians keen on pillage.

B. Agent sensors and controllers

The legions must be trained to acquire appropriate behaviors. They are provided with sensors that divide the map up into six pie slices centered on their own location (figure 2). All the relevant objects i in a pie slice are sensed as a single scalar value, calculated as $\sum_i 1/d_i$. This design provides only a fuzzy, alias-prone sense of what is in each sector of the legion's field of view, but it works well as a threat/opportunity indicator: a few barbarians nearby will be seen as a sensory signal similar to what would be seen of a larger group of barbarians further away.

There is a separate sensor array for each type of object in play: cities, barbarians, and other legions. There are additional sensors in each array to provide more detail about what is in the map cells adjacent to the sensing legion, or colocated in the legion's own cell (figure 3, next page). In practice only a city can be in the legion's own cell, but for simplicity the same sensor architecture is used for all three object types.

The scalar sensor values, 39 in all, are fed into a feed-forward neural network with a single hidden layer of ten neurons and an output layer of seven neurons (figure 4, next page). The output neurons are associated with the seven possible actions a legion can take in its turn: remain stationary, or move into one of the six adjacent map cells. When virtual bagging is not in use, this localist *action unit coding* is decoded by selecting the action associated with the output neuron that has the highest activation level after the sensor signals have been propagated through the network. The modified decoding methods for virtual bagging are described in the next section.

The *Legion II* sensor architecture allows reflections and rotations of the world about a legion's egocentric viewpoint. Such transformations can be represented by permutations of the values in the sensors. For example, a north-south reflection can be implemented by swapping the northwest (NW) sensor values with the southwest (SW), and the NE with the SE.

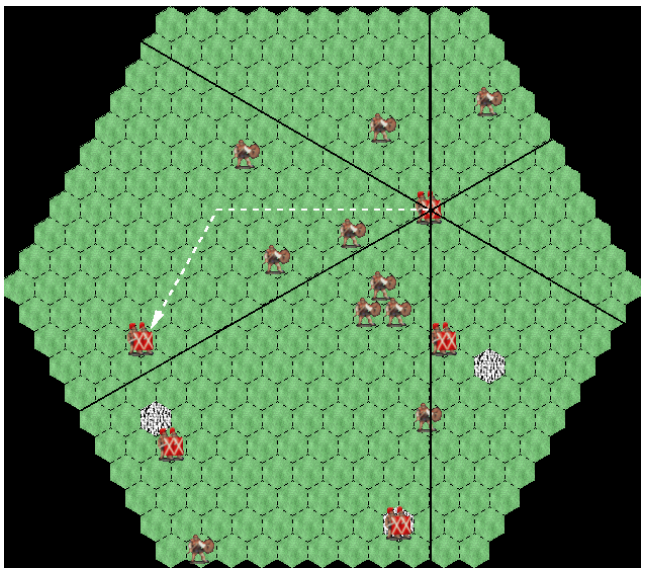

Fig. 2. **A legion's sensor fields.** A legion's sensor array divides the world into six symmetrical "pie slices", centered on the legion itself (black lines). The objects i falling within a slice are detected as the scalar aggregate $\sum_i 1/d_i$, where d is the hexagonal Manhattan distance to the object (white arrow). Note that for any given sensory input the symmetries in the sensor architecture allow a set of six 60° rotations about the legion, plus a reflection of each rotation, for a total of twelve isomorphic sensory views of the world, each a plausible game state in some game. If a legion makes the optimal move in all circumstances, then a reflection and/or rotation of its sensory inputs produces a corresponding reflection and/or rotation in its choice of moves. This desirable behavioral invariant can be exploited to generate artificial training examples from actual examples, as reported in previous work, or can serve as the basis for *virtual bagging* as reported here.

Similarly, a 60° clockwise rotation can be implemented by moving the sensor values for the eastern (E) sector to the southeastern (SE) sensor, for the SE to the SW, etc., all the way around the legion. The legions' choices of action for a reflected or rotated sensory input can be reflected or rotated by the same sort of swapping. For example, a 60° clockwise rotation would convert the choice of a NE move to an E move. The option to remain stationary is not affected by reflections or rotations: if a legion correctly chooses to remain stationary with a given sensory input, it should also remain stationary for any reflection or rotation of that input.

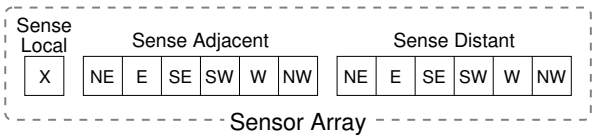

Fig. 3. **A legion's sensor architecture.** Each sensor array for a legion consists of three sub-arrays as shown here. A single-element sub-array (left) detects objects colocated in the map cell that the legion occupies. Two six-element sub-arrays detect objects in the six radial fields of view; one only detects adjacent objects, and the other only detects objects farther away. The legions are equipped with three complete sensor arrays with this structure, one each for detecting cities, barbarians, and other legions. The three 13-element arrays are concatenated to serve as a 39-element input layer for an artificial neural network that controls the legion's behavior (figure 4). Artificial reflections and rotations of a legion's view of the world can be generated on demand by appropriate permutations of the activation values of the sensors in the sub-arrays.

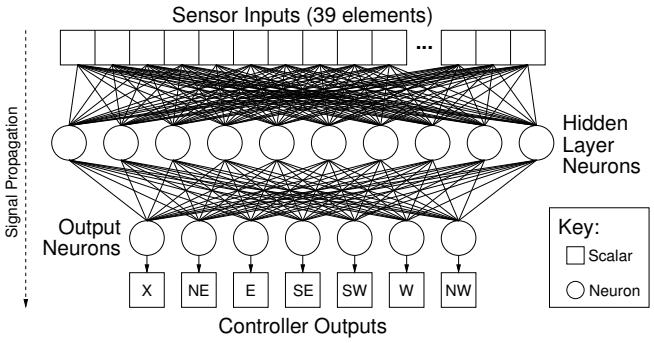

Fig. 4. **A legion's controller network.** During play the values obtained by a legion's sensors are propagated through an artificial neural network to create an activation pattern at the network's output. This pattern is then interpreted as a choice of one of the discrete actions available to the legion. When properly trained, the network serves as the controller for the legion as an autonomous intelligent agent.

III. Experimental evaluation

Virtual bagging was evaluated experimentally in the *Legion II* environment. Two basic mechanisms were considered: *discrete* virtual bagging (DVB) and *continuous* virtual bagging (CVB). In DVB a vote of "1" is cast for the most highly activated output for each of the various rotations and reflections of the actual sensory input, and "0" for all other outputs. After un-rotating/un-reflecting the votes back to the original orientation, the action corresponding to the output with the most accumulated votes is chosen for the agent's current move. In the event of a tie, one of the tied actions is selected at random.

In CVB every output of the network receives a fractional vote equal to the activation level of its neuron, for each of the various rotations and reflections of the actual sensory input. These fractional votes are un-rotated/un-reflected back to the original orientation, and summed across all the rotations and reflections of the current input. The action corresponding to the output with the highest sum is then chosen for the agent, and ties are again broken by a random selection.

Autonomous intelligent agents in other applications are very commonly bilaterally symmetrical but not rotationally symmetrical. Therefore CVB, which proved to be more effective than DVB, was further tested for its effectiveness when using reflections but no rotations, as would be necessary for such agents.

Finally, intuition suggests that an agent may benefit from using virtual bagging when it is deployed post-training, even if VB was not used during training. Likewise, the use of bagging during evolutionary training may produce controllers that are superior, even if it is not used when they are deployed post-training. Therefore CVB was additionally tested for both of those cases.

The mechanics of the underlying neuroevolutionary algorithm are given in section III-A. Then a summary of the higher-level experimental methodology is given in section III-B, and the effectiveness of the various forms of virtual bagging is examined in section III-C.

A. Neuroevolution with enforced sub-populations

All the methods described here used a minor variant of Neuroevolution with Enforced Sub-Populations (NE-ESP) [7], [8]. ESP works by maintaining an independent breeding population for each neuron's position in a network. Fitness evaluations are obtained by drawing one neuron at random from each sub-population and assembling them into a complete controller network (figure 5). The controller is tested, and the fitness score for the entire network is ascribed back to each neuron that participated in it.

The fitness so obtained is somewhat noisy, because the fitness measured for a "good" neuron can be brought down by one or more "bad" neurons used in the same network, or vice versa. This noise is reduced by performing repeated evaluations with different groupings of neurons, and reporting each neuron's individual average of its evaluation scores as its fitness for the current evolutionary generation.

A full treatment of ESP can be found in [7] or [8]. The usage presented here differs from the original in that (a) since a feed-forward network is being evolved, the breeding neurons have representations only for their input weights, rather than for both inputs and outputs, and (b) for the repeated evaluations the algorithm enforced exactly three evaluations for each neuron, rather than repeated random associations until some minimum number of evaluations had been obtained for each. The use of input-weights-only is also the primary difference between *Legion II* and the original *Legion* game reported in [4].

All the runs reported here used a population size of 1000 (for each sub-population). A soft-selection method replaced each member of the sub-populations in order from the worst to the best, selecting two equally or better ranked neurons at random to breed a replacement for the current neuron. This allows poor-performing neurons to be selected for breeding, albeit with an strong overall bias toward using the better-performing neurons during breeding. Note that the mechanism also ensures that the best performer of a generation is represented in the new generation (via breeding with itself), although with a chance of mutations.

Within a sub-population the neurons were represented by simple vectors of floating-point input weights and a bias value. Breeding was done by either 1-point or 2-point crossover, chosen at an independent 50% chance for each breeding pair. After breeding, each value in the resulting representation was subjected to an independent 10% chance of mutation. Mutations were deltas chosen from an exponential distribution (probability density function $\lambda e^{-\lambda x}$, $x > 0$, and $\lambda = 5$), and added to the current value for the weight; the delta was inverted to a negative with a 50% chance. Note that this mechanism produces small deltas with high probability and large deltas with low probability.

B. Experimental methodology

As noted in previous work, evolutionary learning in the *Legion II* game is noisily asymptotic – longer runs produce better results, albeit with diminishing returns [6]. No fixpoint

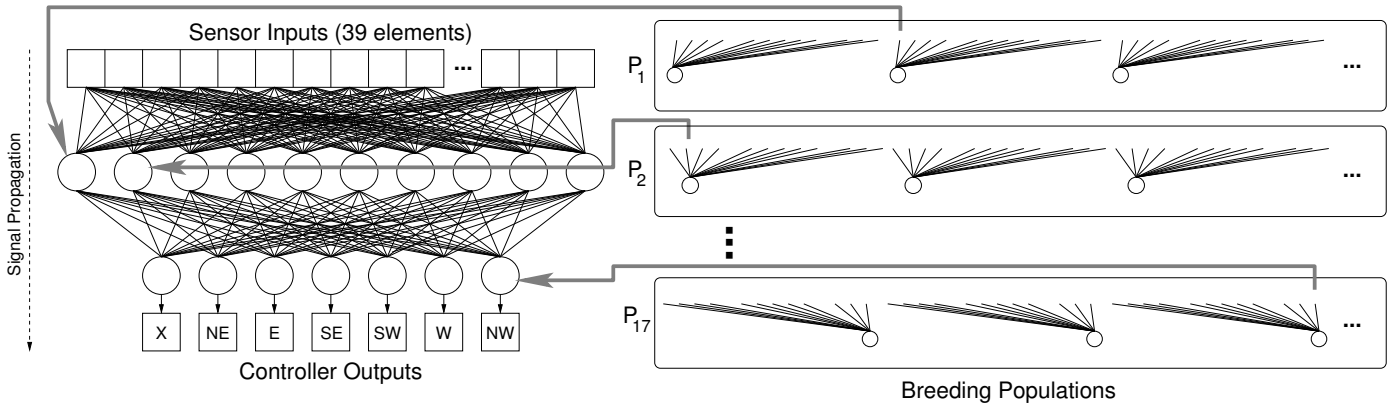

Fig. 5. **Neuroevolution with Enforced Sub-Populations (ESP).** In ESP a separate breeding population (P_1-P_{17}) is maintained for each neuron's position in the network. A network is constructed by drawing one neuron at random from each sub-population and assembling them according to a predefined architecture (grey arrows). Once a network has been evaluated for evolutionary fitness, that fitness value is ascribed back to each neuron that participated in the evaluation.

for evolutionary learning has been detected on any experiments using *Legion II* to date. Therefore learning was continued for 5,000 generations for all the methods described here, to put the learners well out on the "flat" of their learning curve and ensure that undertrained networks were not used for testing and comparison. A typical learning curve is shown in fig. 6.

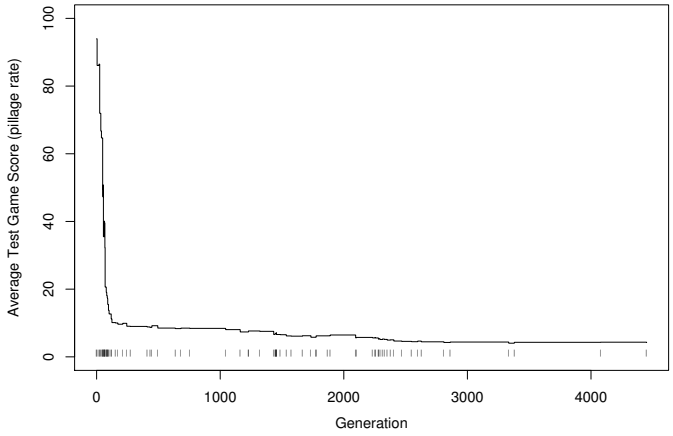

Fig. 6. **A typical evolutionary learning curve for *Legion II*.** The plot shows performance on the test set vs. generation for the median performer of 31 networks trained without virtual bagging. Lower scores are better. The tics under the curve indicate the generations when progress was made on the validation set; the curve is not monotonic because improvement on the validation set does not strictly predicate improvement on the test set. For this 5,000-generation run, the last improvement on the validation set was made at generation 4,450. Long runs are used because progress is roughly asymptotic, and it is essential to avoid using undertrained networks for comparing the methods under investigation.

Each method was tested on 31 independent runs, with independence achieved by using a different seed for the random number generator that controls both the neuroevolutionary algorithm and the choice of game set-ups for learning. Each of these runs provides a controller network that can be tested to provide a single sample point for statistical analysis. Statisticians generally recommend a minimum sample size of 30 such measurements when the distribution is not known to be normal (e.g. [9]); in the present work this was increased to 31 for the number of independent runs, so that there would always be a clearly defined median performer to serve as a principled choice when a specific "typical" example is needed (e.g., as in figure 6).

Each run used a non-repeating sequence of games for the evolutionary evaluations. The games differed in their placement of the cities, starting locations of the legions, and locations of the barbarians' arrivals. Such set-ups can be repeated by carefully marshalling the consumption of random numbers and resetting the relevant generators before starting a game. Thus it is possible to use a nominally "same game" to test all the neurons in a population. The actual course of play still varies because the different controllers make the legions behave differently, and the details of the barbarian behavior depend on where the legions happen to be when it is a barbarian's turn to move. However, fairness of comparison is maintained by equivalence of the starting positions of the cities and legions and the placement of the barbarians upon entry. Thus each neuron was evaluated for fitness while participating in a network that played "the same game" that all the other neurons played.

In order to improve the estimate of a controller network's generalized fitness, a different game was used for each of the three evaluations the neurons underwent during each generation. Different games were used for the evaluations in succeeding generations. This overall sequence of games was generated independently for each run by means of a different seed for the random number generator that created the games, so that the sample of performance measures obtained by the 31 runs is a sample over all possible training sets, rather than on a single training set.

At the end of each generation a *nominal best network* was defined by choosing the best-rated neuron from each sub-population. That network was tested on a validation set of 10 game set-ups, chosen randomly and independently for each of the 31 runs, but used repeatedly at each generation throughout a given run. If the nominal best network performed better on this validation set than the nominal best network from any previous generation, it was deemed to be the best

network found so far. At the end of a run, the most recent of such networks was returned by the algorithm as the controller produced by the run.

Notice that both the unbounded sets of training games and the fixed validation sets are part of the training algorithm, and are thus specific to an individual run. The training games were used to provide an evaluation of neurons for selection and breeding, and the validation set games were used to select which generation's nominal best network had the best generalization properties. However, for comparing results between independent runs it is necessary to use a fixed test set that is common to all of them. For this a set of 31 games was created by using a separate random number generator initialized with the same seed for every run. To avoid biases this seed was different from the seeds that controlled the selection of games in the various runs, and, in fact, different from any seed used during the development of the test bed and tuning of the learning parameters.

In principle the test set is to be used for a post hoc evaluation of the networks produced by the various independent runs. In practice it is convenient to use the test set during a run in order to produce a learning curve. No biases are introduced, because no algorithmic decisions are based on the result of evaluations on the test set games; such evaluations merely produce a number that is spilled to a file for later plotting.

As illustrated by figure 6, at each generation during a run when the nominal best network produced an improved score on the validation set, that network was also evaluated on the test set for plotting. Note that for a given run the performance against the validation set is monotonic due to the "best so far" mechanism. However, improvement on the validation set does not guarantee improvement on the test set, so the evaluations against the test set are not always monotonically improving. Since no training decisions can be based on evaluations against the test set, the test results obtained by the validation set's "best so far" mechanism were reported as the performance of the network produced by the learning algorithm, even if by chance better test scores were obtained at some earlier stage of training.

The average of the game scores on the 31 test games is reported as the performance of a controller, and consequently as the performance of a method on a single run. The collection of these scalars for the 31 runs for each method are plotted below for the various experiments, and used as the samples for inferential statistical analysis.

C. Effect on performance

A first experiment examined the relative effectiveness of virtual bagging versus the base neuroevolutionary method that does not exploit sensor symmetries. Both discrete and continuous VB were evaluated, and they took advantage of the full set of 12 rotations $\times$ reflections possible with a hexagonal map grid and corresponding six-sided sensor array.

The experiment shows that both methods of virtual bagging produced more effective agents than the base method, with CVB being the most effective (figure 7). The mean game scores achieved by the three methods were 4.511 for the base method, 3.401 for DVB, and 2.906 for CVB – recall that lower scores are better. The improvement of DVB over the base method was statistically significant at the 95% confidence level on the one-tailed Welch Two Sample t-test ($p = 2.038 \times 10^{-07}$), as was the improvement of CVB over DVB ($p = 6.352 \times 10^{-11}$).

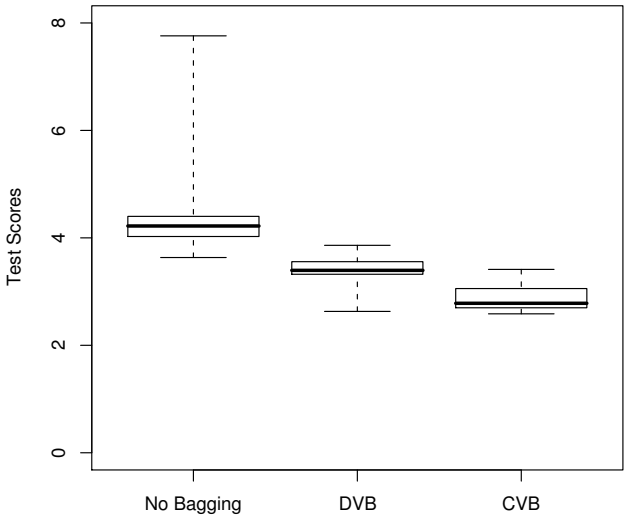

Fig. 7. **The effect of virtual bagging on performance.** Boxplots show the means, quartiles, and extremes for 31 independent runs of the learning experiment without bagging (left), with discrete-vote virtual bagging (DVB, center), and with continuous-vote virtual bagging (CVB, right). Lower scores are better. Both methods of voting provided statistically significant improvement over the base non-bagging method, but the continuous-vote method also provided a statistically significant improvement over the discrete-vote method.

Thus the experiment reveals that both methods of virtual bagging provide improved performance over the base non-bagging method on the *Legion II* problem, with CVB being the more effective of the two.

A second experiment further examined continuous virtual bagging to see whether it would still be useful for the common case of controllers for agents that can exploit bilateral symmetry but not rotational symmetry.

When both rotations and reflections are used, the same set of 12 is obtained regardless of which axis is used for the reflections. However, when the rotations are not used, different axes of reflection will give different sets of inputs. Since the agents in *Legion II* do not have any distinguished orientation other than for their graphical displays, all the reflections for this experiment were arbitrarily made by a north-south flip of the sensory input and choice of move.

The experiment shows that using CVB without the rotations still provides an improvement over the base method, though not as substantial an improvement as when both rotations and reflections are used (figure 8). The mean performance for CVB without rotations was 3.439, vs. the 4.511 base case and 2.906 for CVB with both rotations and reflections. The improvement of CVB without reflections over the base method was statistically significant at the 95% confidence level on the one-tailed Welch Two Sample t-test ($p = 5.461 \times 10^{-07}$), but

the improvement of CVB using both rotations and reflections was also statistically significant over the reflections-only case ($p = 9.976 \times 10^{-08}$).

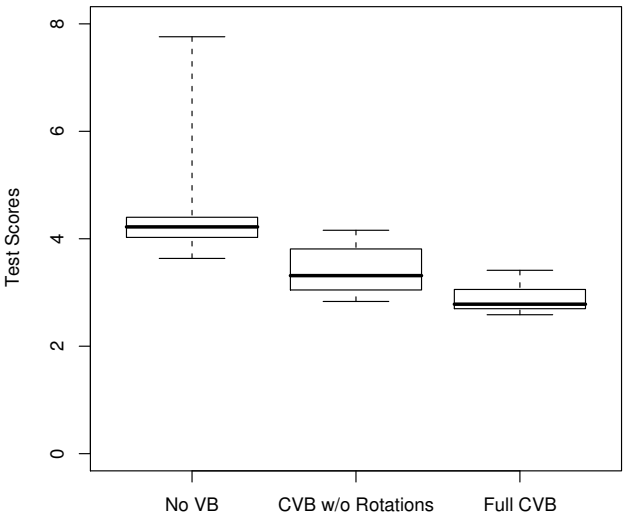

Fig. 8. **The effects of rotations and reflections.** Continuous-vote virtual bagging was evaluated with reflections only (center), as would be necessary for bilateral agents whose behavior would not be symmetric with respect to rotations of their sensory inputs. The results of the experiment without bagging (left) and with continuous-vote virtual bagging using both rotations and reflections (right) are shown again for comparison. The reflections-only method did not perform as well as when both rotations and reflections were used, but it still provided a statistically significant improvement over the base non-bagging method.

Thus the experiment reveals that both these methods of continuous virtual bagging provide improved performance over the base non-bagging method on the *Legion II* problem, though with the case of using both rotations and reflections being more effective than the case of using reflections only.

A final experiment examined the following conjectures. If VB is effective when used for both training and deployment, might it not also be effective if it were used during training, but not during deployment? Or similarly, might it not be effective if it were used during deployment, but not during training?

These conjectures are motivated by the fact that VB requires additional computation time over the base non-bagging method. For time-critical applications it would be useful to apply VB during training and omit it during deployment, if most of the advantages were still obtainable. Or, if computation time is not a bottleneck during deployment, it would be useful to speed up training by omitting the bagging then, and applying it only during deployment.

To evaluate the first conjecture continuous virtual bagging was applied during training (i.e., on both the training and validation set games) but disabled when testing on the test set. To evaluate the second conjecture CVB was disabled during training but applied when the resulting neural controllers were tested against the test set.

Unfortunately, the experiment showed that neither method was effective on the *Legion II* game, with both producing results that were worse than the base method (figure 9). The lack of effectiveness in the deployment-only case is especially surprising, since the arguments that motivated the use of virtual bagging in Section I would seem to be applicable in this situation as well.

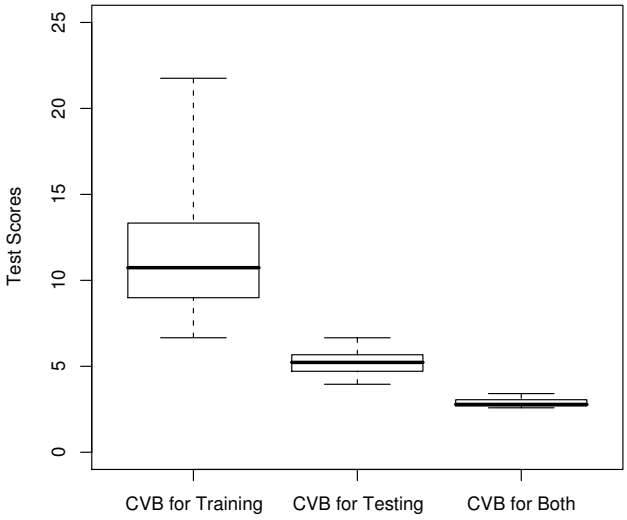

Fig. 9. **The effects of training and testing.** Continuous-vote virtual bagging was evaluated when used for training but not for testing (left), for testing but not for training (center), and for both. Neither perform as well as when both are used, or even as well as the non-bagging method (cf. figure 7). Note that the y-axis is compressed in comparison to the previous plots.

IV. Discussion and future work

The experiments show that virtual bagging can be useful for at least some control applications. It provides the best performance scores of any of the various methods used on the *Legion II* game to date. It also fully enforces the symmetrical behavior invariance, which could only be approximated by a closely related supervised learning method (cf. [3]). Its costs are the computational overhead of calculating the various rotations and reflections of the inputs and of processing each by the controller. Though network propagations on modern computer architectures are very fast, this n-fold multiplication of the effort could be prohibitive in applications that involve large numbers of agents or require computational resources for other tasks.

The continuous-vote method of virtual bagging was found to outperform the discrete-vote method. This is probably a result of exploiting the fact that neural networks can express confidence values in their choices via the activation levels of their various outputs [10]. Notice that while artificial neural networks were used for the current experiments, they are not strictly necessary for virtual bagging. Any mechanism that maps sensory inputs onto a choice of outputs should suffice. However, the superiority of CVB over DVB suggests that virtual bagging will be most effective if that mechanism represents the choice of outputs with graduated values rather than an all-or-nothing choice.

The current work has made use of stateless feed-forward networks for the controllers. Using virtual bagging with recur-

rent networks or other state-based controllers will require additional care. Presumably it will suffice to maintain a separate image of the current internal state for each rotation/reflection being used, so that computation can be continued from each of those distinctive states at the next time step, independently for each of the various rotations/reflections. However, this is a conjecture that will need to be evaluated experimentally.

Virtual bagging faces a bigger challenge in its application to continuous-output controllers, as opposed to the discrete-choice controllers used in the *Legion II* game. For example, if an agent's controller specifies the angle and velocity of a motion to be undertaken, it may suffice to use a mean value calculated by the various rotations and reflections – if the values are well clustered. However, if some of the values represent a choice to go around the left side of an obstacle and others represent a choice to go around the right side, a simple averaging could be disastrous. Thus the concept of virtual bagging will require further development before it can be applied to continuous-output controllers.

The benefits of using both rotations and reflections, vs. reflections only, suggests a "more is better" factor in the application of virtual bagging. Thus it may be profitable to examine the effectiveness of the technique in applications with a higher degree of rotational symmetry (e.g., octagonal sensor geometries) or a higher degree of reflectional symmetry (e.g., agents operating in deep water or deep space). Moreover, the technique should have applications other than controllers with egocentric sensors. For example, board games such as *Go* have state spaces that still make sense under rotations and reflections, so virtual bagging may be immediately applicable to existing *Go*-playing agents.

Finally, it can be observed that it may be possible to leverage the advantages of virtual bagging by adding on the similar but weaker advantages of supervised learning with artificially created training examples created by means of rotation and reflection, such as previously described in [3]. One potential mechanism for the combination is Lamarckian evolution [11], [12]. Lamarckian neuroevolution has already been applied to the *Legion II* game for experiments in inductive player modelling, but using it to combine virtual bagging and artificially generated examples is still a matter for future investigation.

V. Conclusions

Intelligent agents with sense-response controllers often have symmetries in their sensor architecture, and it is then possible to define behavioral invariants that would be observed across those symmetries by perfectly trained agents. This observation suggests that symmetrical invariants of sense-response behavior can be exploited by agents' controllers, making their responses more symmetrical and effective. This paper shows that exploiting such symmetries via *virtual bagging* yields both types of improvement in a game-like test environment, and suggests that exploiting sensor symmetries may provide similar benefits in other environments and with other learning and control methods.

Acknowledgments

This work was supported in part by NSF EPSCoR grant EPS-0447416. The images used in *Legion II*'s animated display are derived from graphics supplied with the game *Freeciv*, `http://www.freeciv.org/`. Special thanks to Raymond Mooney and Risto Miikkulainen for feedback during the development of this work.

References

[1] L. Breiman, "Bagging predictors," University of California, Berkeley, CA, Tech. Rep., 1994, department of Statistics Technical Report No. 421. [Online]. Available: http://citeseer.ist.psu.edu/breiman96bagging.html

[2] B. D. Bryant, "Evolving visibly intelligent behavior for embedded game agents," Ph.D. dissertation, Department of Computer Sciences, The University of Texas at Austin, Austin, TX, 2006. [Online]. Available: http://www.cse.unr.edu/~bdbryant/papers/bryant-2006-tr.pdf

[3] B. D. Bryant and R. Miikkulainen, "Exploiting sensor symmetries in example-based training for intelligent agents," in *Proceeedings of the 2006 IEEE Symposium on Computational Intelligence and Games (CIG'06)*, S. J. Louis and G. Kendall, Eds. Piscataway, NJ: IEEE, 2006, pp. 90–97. [Online]. Available: http://www.cse.unr.edu/~bdbryant/papers/bryant-2006-cig.pdf

[4] ——, "Neuroevolution for adaptive teams," in *Proceeedings of the 2003 Congress on Evolutionary Computation (CEC 2003)*, vol. 3. Piscataway, NJ: IEEE, 2003, pp. 2194–2201. [Online]. Available: http://www.cse.unr.edu/~bdbryant/papers/bryant-2003-cec.pdf

[5] ——, "Evolving stochastic controller networks for intelligent game agents," in *Proceedings of the 2006 Congress on Evolutionary Computation (CEC 2006)*. Piscataway, NJ: IEEE, 2006, pp. 3752–3759. [Online]. Available: http://www.cse.unr.edu/~bdbryant/papers/bryant-2006-cec.pdf

[6] ——, "Acquiring visibly intelligent behavior with example-guided neuroevolution," in *Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI-07)*. Menlo Park, CA: AAAI Press, 2007, pp. 801–808. [Online]. Available: http://nebl.cse.unr.edu/archive/papers/bryant-2007-aaai.pdf

[7] F. Gomez and R. Miikkulainen, "Solving non-Markovian control tasks with neuroevolution," in *Proceedings of the 16th International Joint Conference on Artificial Intelligence*. San Francisco, CA: Morgan Kaufmann, 1999, pp. 1356–1361. [Online]. Available: http://nn.cs.utexas.edu/keyword?gomez:ijcai99

[8] F. Gomez, "Robust non-linear control through neuroevolution," Ph.D. dissertation, Department of Computer Sciences, The University of Texas at Austin, 2003.

[9] R. R. Pagano, *Understanding Statistics in the Behavioral Sciences*, 2nd ed. St. Paul, MN: West Publishing, 1986.

[10] S. Renals and N. Morgan, "Connectionist probability estimation in HMM speech recognition," International Computer Science Institute, Berkeley, CA, Tech. Rep. TR-92-081, 1992.

[11] J.-B. Lamarck, *Philosophie zoologique*, Paris, 1809.

[12] D. Whitley, V. S. Gordon, and K. Mathias, "Lamarckian evolution, the Baldwin effect and function optimization," in *Proceedings of the International Conference on Evolutionary Computation*, Y. Davidor, H.-P. Schwefel, and R. Maenner, Eds., vol. 866, Jerusalem, Israel, October 1994.