Genetic Evolution of Hierarchical Behavior Structures

Brian G. Woolley and Gilbert L. Peterson
Air Force Institute of Technology
Department of Electrical and Computer Engineering
2950 Hobson Way, Bldg 640
Wright-Patterson AFB, OH 4543
(937) 255-3636
brian.woolley@ieee.org, gilbert.peterson@afit.edu

ABSTRACT
The development of coherent and dynamic behaviors for mobile robots is an exceedingly complex endeavor ruled by task objectives, environmental dynamics and the interactions within the behavior structure. This paper discusses the use of genetic programming techniques and the unified behavior framework to develop effective control hierarchies using interchangeable behaviors and arbitration components. Given the number of possible variations provided by the framework, evolutionary programming is used to evolve the overall behavior design. Competitive evolution of the behavior population incrementally develops feasible solutions for the domain through competitive ranking. By developing and implementing many simple behaviors independently and then evolving a complex behavior structure suited to the domain, this approach allows for the reuse of elemental behaviors and eases the complexity of development for a given domain. Additionally, this approach has the ability to locate a behavior structure which a developer may not have previously considered, and whose ability exceeds expectations. The evolution of the behavior structure is demonstrated using agents in the Robocode environment, with the evolved structures performing up to 122 percent better than one crafted by an expert.

Categories and Subject Descriptors
I.2.9 [**Artificial Intelligence**]: Robotics – *Autonomous Vehicles*; D.2.2 [**Software Engineering**]: Design Tools and Techniques – *Evolutionary Prototyping.*

General Term: Design

Keywords
Evolutionary Robotics, Genetic Programming, Behavior-Based Robotics, Unified Behavior Framework.

1. INTRODUCTION
Mobile robots inherently exist in dynamic environments and are expected to react well in unpredictable situations while performing their task(s). Currently, most robots employ some form of reactive behavior architecture [18]. To cope with the variety in the environment, agents are implemented with a broad set of skills, or behaviors. The goal of fusing several behaviors into a single complex behavior is to deliver a coherent sequence of actions that are ultimately more effective in a given environment than any single behavior [22]. Such attempts have proven to be a significant endeavor for two reasons. The first is that the code complexity of a behavior grows exponentially as additional traits are added. The second is that development of a behavior that tries to maximize some criteria while minimizing others is the optimization of a multi-objective problem [6].

To ease the complexity of designing and coding a behavior, a behavior framework is introduced such that simple and independent behaviors can be interchangeably arranged into an arbitrated hierarchy. The goal of using the framework is to allow for: parallel development of elemental behaviors and arbitration units, to restrict the complexity of implementation to the development of elemental behaviors, to encourage code reuse within the domain, and to allow the application of an evolutionary algorithm to discover sets of near-optimal behavior structures for the domain.

By using the unified behavior framework (UBF) [23] to provide behavioral logic to robots operating in the Robocode domain, various behavior structures are possible based on a pool of simple elemental behaviors and interchangeable arbiter components. While many formations are poor choices, some unexpected combinations may be quite good. By using the environment as an evolutionary pressure, an initial population of randomly formed structures is able to organize itself into coherent behaviors that are well suited to combat. Through the repetitive application of ranking each member and then evolving the population by application of a genetic programming algorithm, behavior structures emerge that are effective on an absolute scale [14].

2. BACKGROUND
The basis of this paper's research draws on previous work related to evolutionary computation, behavior based robotics, and the evolution of robotic controllers.

2.1 Evolutionary Algorithms (EA)
The class of stochastic, global search & optimization algorithms that use the repetitive application of seemingly simple rules to discover emergent behaviors are known as evolutionary algorithms (EA). Such techniques loosely imitate natural evolution and the Darwinian concept of *Survival of the Fittest* [11]. EA techniques are especially effective in large search spaces because, they have a random element that makes them less

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

susceptible to becoming trapped in a local minimum. Since evolutionary pressures are directing the search, they provide good solutions to a wide range of optimization problems that traditional deterministic search methods find difficult [12].

In nature, the evolutionary process occurs when the following four conditions are satisfied: 1) an entity has the ability to reproduce itself, 2) there is a population of such self-reproducing entities, 3) there is some variety among the self-reproducing entities, and 4) some difference in ability to survive in the environment is associated with the variety [14].

One particular subset of EA algorithms is genetic programming (GP). This subset is defined by its ability to manage the adaptation of complex structures. Typically the structures are hierarchical in nature, stored as trees, rather than sequentially as in genetic algorithms. Since the organization and ordering of a member's structure is important, it must be preserved during crossover (or sexual recombination). A single GP cycle, referred to as an epoch, consists of five major events: 1) a fitness evaluation of each member's ability to cope in the environment, 2) a ranked ordering of the population, 3) a period of recombination where the strongest members have the greatest probability of reproducing, thus propagating successful attributes, 4) an opportunity for mutation, which is optionally used to introduce variety and avoid local minima and 5) a pruning of the population size by removing unfit members. Once one epoch is complete a new epoch begins [6, 14].

Many times an environment is competitive and adversarial in nature, meaning that the members of a population must gain their fitness measure at the expense of another. Such competitive evolutions rank individuals in the population relative to their peers. This approach is beneficial because despite the members of the initial population being highly unfit, over a period of time, members evolve and rise to higher levels of performance as measured in terms of absolute fitness. What is interesting is that such a process is a self-organizing, mutually bootstrapping process that is driven only by relative fitness (and not by absolute fitness) [14].

2.2 Behavior Based Robotics

Research efforts in robotics through about 1985 focused almost exclusively on planning and world modeling [10] in an attempt to develop completely rational mobile robots [18]. The sense-plan-act approach, Figure 1a, proved inadequate in dynamic and unpredictable environments, where the robot finds itself in trouble when its internal state loses sync with the reality that it is intended to represent [2]. This is because anything approaching a real world model typically requires so much time to maintain and develop plans for, that the state of the environment changes before the actions can be carried out, effectively nullifying the action sequence. The main problem is that a traditional Lorenz control loop [19] directly links the rate at which a robot can evaluate its environment to the computational time requirements of the planning module.

The need to alleviate this planning bottleneck led tasks to be decomposed into collections of low-level primitive behaviors. This approach took on the self-contradictory term, reactive planning [10]. The ideas behind reactive planning stem from arguments such as Braitenberg's, who argues that the complex behavior of natural organisms may be the result of simple behaviors. Braitenberg further argues that by combining simple behaviors, more complex behaviors and attributes are possible [3]. In equivalent research Brooks claims that for many tasks, robots do not need traditional reasoning, only a tight coupling of sensing to action. He backs that claim with robust autonomous robots using the Subsumption architecture [4, 5].

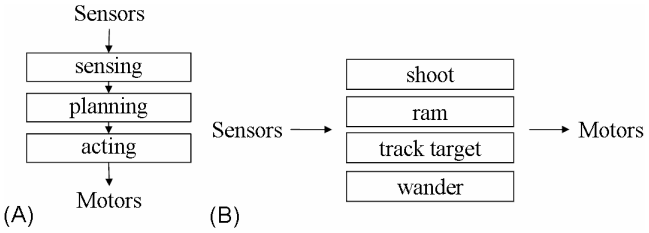

Figure 1: Two organization decompositions for robot control (A) Sequential execution of functional modules; (B) A task-based decomposition into parallel execution modules.

Subsumption is the canonical architecture that advocates for a layered control system based on task decomposition, an approach which is radically different from previous research. Figure 1 highlights this quintessential paradigm shift, with the sense-plan-act architecture shown in (A) and the new horizontal structure of Subsumption shown in (B). This parallel organization naturally promotes concurrent and asynchronous responses to sensor input. Each individual layer works to achieve its particular goal. Coordination between layers is achieved when complex actions (or higher layers) subsume simpler actions, or when low-level behaviors inhibit higher layers. From this work other distinct behavior architectures emerged: motor schemas [1], circuit architecture [13], action-selection [15], colony architecture [7], animate agent architecture [8], DAMN [17] and utility fusion [18].

Traditionally, a mobile robot design implements a single behavior architecture, thus binding its performance to the strengths and weaknesses of that architecture. The unified behavior framework (UBF) incorporates the critical ideas and concepts of these eight existing reactive controllers, demonstrating that each can be represented using a single straightforward framework. At is core the UBF uses an abstract behavior interface to define a family of behavioral algorithms that can be used interchangeably regardless of the underlying behavior architecture. Using the UBF, the developer (and in this case, the evolutionary algorithm) is not restricted to using any single behavior architecture [23].

Additionally, the UBF supports the construction of new behaviors as compositions of existing behavior modules, the reuse of subcomponents is also encouraged in the UBF via a mechanism modeled on the composite pattern [9]. The composite pattern allows new control structures to be formed as arbitrated hierarchies of existing behaviors, with the resulting structure acting as a single behavior unit [23].

The software design mechanisms of the strategy and composite patterns encourage a developer to use modular approaches that ease the complexity of designing, testing and implementing a collection of reactive behaviors, while providing the ability to form larger hierarchies of behaviors. This isolates code complexity to the atomic (or leaf) behaviors. The freedom to join existing behaviors as compositions encourages experimentation

with various structural arrangements of elemental behaviors, arbitration components and existing behavior structures [23].

Under the UBF, behavioral structures are formed by joining groups of behaviors together at arbitrated nodes. At a structure's lowest levels are the leaf behaviors, which capture the simple logical skills of the system. Each leaf passes an action recommendation to the joining node above. Each joining node uses its arbiter to consolidate the recommendations of its sub-behaviors into a single action recommendation that is passed to the joining node above. Such a hierarchical structure ensures that the root of the behavior will only present a single action recommendation to the robot controller. For longer descriptions of the arbiters and the behaviors used in this experiment, refer to section 3.5.

2.3 Evolutionary Robotics

Previous work that spans the boundary between evolutionary computation and robotics includes the evolution of a layered Subsumption architecture for robot control [20] and the evolution of controllers for racing a remote controlled car around a track [21]. This work demonstrates the ability of evolution to develop and tune individual architectures to a given domain.

This paper uses the common interface of the unified behavior framework [23] to expand the scope of the evolutionary search, allowing the fitness of several behavior-based architectures to be evaluated concurrently for a particular domain.

3. IMPLEMENTATION

The discussion of this experiment's design and implementation is presented first as a high level design followed by an explanation of the Robocode adaptation, the fitness function, the genetic program, and concludes with a description of the elemental behavior/arbiter components.

3.1 High-Level Design

Because the UBF behavior structures are trees, consisting of root nodes with arbiters and leaf behaviors, the mapping to a genetic programming representation is straightforward. The high-level design of the evolutionary system used to automate the discovery of effective behavior structures is centered on the *fitness function* and the *evolution engine*. An adaptation of the Robocode robot battle simulator forms the basis of the fitness function which interacts with the evolution engine via input and output files. Each epoch of the evolutionary process is established by the repetition of four execution stages: Stage I enacts the relative fitness function described in section 3.3.1 to evaluate the relative fitness of individuals in a population. Stage II is the evolutionary engine that advances a population by one generational time, i.e. from P(t) to P(t+1). Stage III enacts the absolute fitness function described in section 3.3.2 to measure a population's current level of fitness, in reference to an unchanging benchmark behavior. Stage IV is a simple parser that maintains a historical record of each population's evolutionary progress. This four step cycle is shown in Figure 2.

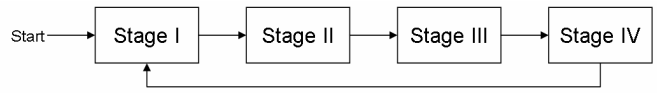

Figure 2: Cyclical progression of Stages I through IV.

3.2 Robocode Adaptation

Robocode [16] was chosen as a simulation environment because it provides a dynamic, straightforward environment for comparing different control architectures. However, it is not as useful for experimenting with established robot control architectures because rather than accepting motor commands, commands are discrete requests that set a robot to turn left 90 degrees, or travel a set distance and then stop. This motor interface is atypical of standard robot motor control mechanisms. For this reason, the motor interface of Robocode version 1.0.7 was adapted to allow for a velocity based approach, it now accepts commands that specify the desired velocity and rate of turn for the chassis as well as the turn rate for the gun turret and the radar. Once set, these rate based values persist until changed.

3.3 Fitness Function

The scoring mechanism provided in Robocode provides a quantifiable metric that indicates the relative fitness that two or more behavior structures have in a given environment. In this experiment the fitness function is configurable to operate in either a relative or an absolute fitness evaluation mode. The first is used during Stage I to rank the individuals in a population relative to each other. The second evaluation mode is used in Stage III to capture a population's absolute fitness relative to an unchanging benchmark behavior. This section concludes with a discussion of the noise parameters inherent in using a nondeterministic fitness function and presents the standards for this experiment.

$$(1)\quad R(k) = \frac{n \cdot score_k}{\sum_{i=1}^{n} score_i} - 1 \qquad (2)\quad Pr(k) = \frac{score_k}{\sum_{i=1}^{n} score_i}$$

An individual's rating and probability of selection are defined by equations (1) and (2) respectively, where ***n*** denotes the number of members in a population and ***k*** is a specific individual.

3.3.1 Relative Fitness

The relative fitness mode is the evaluation mode used during Stage I and ranks individuals in the population relative to their peers, regardless of their absolute fitness. The Robocode application is configured using the melee battle file and places ten robots on the battlefield for a twenty-five round, *all-for-one* melee. Because individuals advance their score by exploiting other members, the scores that result from this sequence provide a means of stratifying the members of a population relative to each other. Each member's rating, R(k), is calculated as the percent difference of a nominal score (total score/***n***); values above zero indicate superior combat skills while below zero ratings indicate an inferior level of performance. The probability of selection for an individual is based on their fraction of the total score.

3.3.2 Absolute Fitness

The absolute fitness mode is the evaluation mode used during Stage III to gain insight into how subsequent generations of a population are progressing over time by ranking it against a fixed benchmark behavior. This evaluation is used to observe the fitness of a population on an absolute scale and is never used to drive the direction of the evolution. In this mode, the Robocode application is configured to set the population's fittest member against the benchmark behavior for a twenty-five round, *one-on-one* battle.

In most cases this approach provides a good estimate of absolute fitness. However, in some cases, a population can discover structures that are particularly good at defeating the benchmark without being a globally optimal solution. For this reason, these values only serve as an indicator of how a population is progressing towards the notion of absolute fitness.

The benchmark behavior, as generated by a user expert, is shown in Figure 3 and consists of the behaviors Wander v3, Charge, Dodge and Fire v1 joined by an activation fusion arbiter. The benchmark's observed behavior has three operating modes: one that executes a random S-wander pattern across the battlefield while attempting to track and shoot opponents, another which aggressively charges towards a nearby weaker opponent with guns blazing, and an evasive behavior that emerges above the other two when the benchmark is taking fire from unseen opponents.

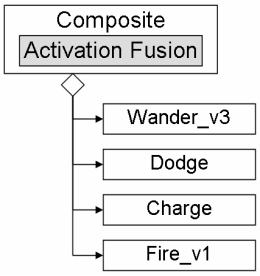

Figure 3: The control structure of the benchmark behavior.

3.3.3 Noise Parameter

The jitter inherent in the absolute fitness function is caused by the stochastic variance in the simulator's ability to accurately stratify members relative to each other. The nondeterministic progression of battles in Robocode is caused by random starting postures and the dynamic interaction of opposing behavior algorithms. The results of any battle will have some level of uncertainty, where the more rounds per battle, the smaller the uncertainty. To demonstrate this, a sequence of battles is created with the benchmark facing itself in combat. On average, when identical behavior structures are set against each other, neither one should score better than the other. When battles consist of five rounds each, the average relative fitness measured is 0.6% with a standard deviation of 40.2%. When battles consist of twenty-five rounds each, the average relative fitness is 0.5% and the standard deviation drops to 17.6%, the raw noise is illustrated in Figure 4a.

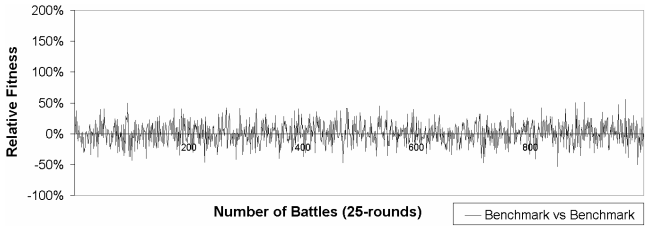

Figure 4a: Noise for Benchmark vs. Benchmark (25-rounds).

Although, increasing the number of rounds per battle reduces jitter and more accurately stratifies an individual's relative fitness, this approach is prohibitive due to time requirements. To keep the speed of the evolutionary cycles manageable, twenty-five round battles are established as the standard for this experiment, setting the fitness function's noise parameter at plus or minus 17.6% per battle. To smooth the representation of how sequences of battles are progressing, a ten-tap moving average is applied to smooth the results and establish a noise floor. Applying this filter to the data in Figure 4a establishes a noise floor expectation with a near zero average and a jitter of 5.45%. The effect of using this approach is illustrated in Figure 4b and is applied throughout the experiment.

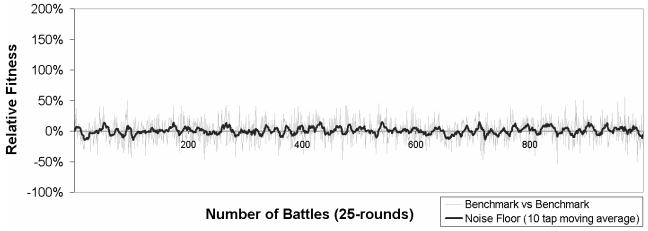

Figure 4c: A 10-tap moving average dampens variance, shows trends over time and establishes the experiment's noise floor.

3.4 Genetic Program (GP)

The hierarchical nature of behavior structures under the UBF allows a genetic program (GP) to perform a stochastic search of the solution space. The GP in this experiment maintains a fixed population of ten members and uses *elitism*, *mutation* and *generational recombination* to guide the search from an initial random population towards a set of behavior structures that are coherent for the domain. The GP's parameter settings are specified in Table 1.

Table 1: Parameters and settings of the genetic program.

Parameter	Symbol	Setting
Population Size	n	10
Elitism Rate (%)	E	10%
Mutation Rate (%)	M	10%
Generation Rate (%)	G	80%
Contributing Set Size	r	$G \cdot n$
Variance (%)	v	± 10%
Max Branching	b	4
Max Depth	d	7
Number of Generations	X	1000

The Elitism rate *(E)* provides the GP a means of propagating successful structures as they are discovered. By advancing a fraction of the population with highest fitness directly from population P(t) into P(t+1), the GP partially becomes hill climbing.

The Mutation rate *(M)* adds a random element to the search, attempting to avoid becoming trapped in local minimum. This fraction of the population P(t+1) are randomly generated behavior structure intended to maintain the genetic diversity of the population and promote exploration throughout the course of the search.

The Generation rate *(G)* specifies the rate of generational recombination. This fraction of the population P(t+1) are new behavioral structures formed by the crossover of members in the contributing set. Recombination is a two step process consisting of a *selection* step and a *crossover* step:

The selection process uses stochastic universal selection (SUS) [2] to choose the contributing members from the population P(t). SUS uses r equally spaced markers across the population's score distribution. The selection markers shift within the selection space based on the initial value (or seed). The seed is a randomly selected value between zero and 1/r.

During crossover, pairs of individuals are randomly selected from the contributing set of members and through the process of genetic recombination, each pair forms two new individuals that are ultimately introduced into the population P(t+1). During a crossover event, a randomly chosen branch from each contributing UBF tree is removed and given to the other. By swapping behavioral sub-structures, two offspring are created where the donated portion is replaced by the acquired structure.

The resulting offspring are then pruned at the maximum depth *(d)* to limit their complexity and are given additional variation *(v)* through fluctuations in the behavior weights held by each arbiter. The new generation of members is then introduced into the population P(t+1).

3.5 Description of Elemental Components

Using the UBF interface, thirteen elemental behaviors and seven arbiters are developed and tested as independent components. The functionality of each component is briefly described below and then used as the pool of genetic material from which members of the population are formed.

The behaviors are:

Charge—when another robot (with a lower energy level) is detected, this behavior causes our robot to turn towards the other and charge towards it, attempting to cause damage by hitting it.

Dodge—when hit by a bullet or by another robot, this behavior causes our robot to respond with an evasive maneuver based on the type of attack and afflicted quadrant.

Fire v1—has three operating modes. When no target is detected, the default mode turns the turret in a clockwise direction. When a target is detected, the target tracking algorithm causes the gun turret rotation to slow or reverse its direction in an attempt to continue tracking the target. In addition to target tracking, when the target is less than three degrees off boar site our robot will fire on another, the power committed to the bullet is reduced as a function of the target off boar site angle.

Fire v2—is exactly like Fire v1 with the exception that the maximum power is always committed to the bullet.

Return Fire—holds a grudge against another that has previously attacked our robot. When no specific target is set, the default mode behaves exactly like Fire v2 until our robot is shot or hit by another. When an aggressive opponent is specified, only that target is engaged. The aggressor remains the target until it is killed.

Scan Left—turns the gun turret and the radar counterclockwise.

Scan Right—turns the gun turret and the radar clockwise.

Short Range Fire—is based on Fire v1, but only fires at targets that are at close range and are less than fifteen degrees off boar site. Maximum power is always given to the bullet.

Sitting Duck—will always recommend that our robot stop all motion, including the motion of the gun and the radar.

Sniper Fire—is adapted from Fire v1 and is specialized to attack slow moving targets at long ranges. When a target is found to be stopped or moving slowly it recommends that our robot stop its movement and track the target until it is less than one half of a degree off boar site. Maximum power is always given to the bullet.

Wander v1—circumnavigates the perimeter of the board. Our robot's current velocity is maintained unless it is less than the minimum.

Wander v2—simulates Brownian motion by randomly executing a series of fifty degree arcs. When a wall is detected, the current velocity is flipped to reverse our direction.

Wander v3—performs a series of "S" turns. Random selection is used to set the length of the arc to be between thirty and one hundred twenty degrees before changing the turn direction. When a wall is found, the current velocity is reversed to change our direction.

The available arbitration techniques are:

Activation Fusion—is a semi-cooperative arbiter that uses a highest activation selection approach on a per motor command basis. Unlike highest activation, activation fusion builds a new action set, allowing the motor commands left unspecified by the behavior with highest level of activation to be set using the recommendations of behaviors with lower activation levels. When used with market based systems, this technique is easily referred to as utility fusion, but risks confusion with Rosenblatt's utility fusion [18] behavior architecture.

Command Fusion—is derivation of the motor schema architecture [1], a cooperative arbitration approach that uses summation and normalization of proposed motor commands to derive the resultant set of motor commands. The input of all contributing behaviors are used on a per motor command basis to form the resultant command vector.

Highest Activation—is a winner-take-all arbiter that returns the action set with the highest vote value. Inspired by the action-selection architecture [15], this approach provides a dynamic mechanism for competitive selection by allowing behaviors to indicate their urgency for activation. Associated behavior weights are used to internally tune global performance by scaling the votes of behaviors that either over or under vote. The concept of activation levels is synonymous with the concept of utility in market based systems.

Highest Priority—is a winner-take-all arbiter that returns the action set of the highest priority behavior indicating a desire to act, regardless of vote value. Like Subsumption [4, 5], the recommendations of lower priority behaviors only execute if higher priority behaviors abstain.

Monte Carlo—is a stochastic arbitration technique that uses fitness proportional random selection to activate one sub-behavior for a period of time. At the end of the period another random selection occurs, activating the chosen sub-behavior for the current period.

Null Arbiter—always passes an empty action back, regardless of the action set passed in. Using this arbiter deactivates the branch of control where it is applied.

Priority Fusion—is a semi-cooperative arbiter that uses priority based arbitration on a per motor command basis. Unlike the

highest priority arbiter above, priority fusion builds a new action set that allows the unspecified action fields of higher priority behaviors to be filled by lower priority action requests.

4. RESULTS

In this experiment, eight behavior populations are independently evolved over the course of 1,000 generations. While the initial populations are collections of randomly generated behavior structures and are generally unfit on an absolute scale, they introduce variety into the population. Through the repetitive ranking, selection and recombination of the members within a population, initially random structures organize themselves into populations of structures that are measurably effective on an absolute scale [14].

In this experiment each of the eight initial populations converges on relatively simple solutions that exploit similar aspects of the Robocode domain. This section discusses how the populations' absolute fitness progresses over time, then discusses the critical aspect of the Robocode domain that acts as the evolutionary pressure shaping the solutions, and finally concludes with a comparison of how the individual solution structures rate relative to each other.

The absolute fitness of each population is a measurement of the population's performance against the fixed behavior structure, which allows the progress of independent evolutions to be compared directly. The fitness rating is calculated as the percent difference of a nominal score; values above zero indicate superior combat skills while below zero ratings indicate an inferior level of performance. The trend graph presented in
Figure **5**a is a progression of the eight individual populations as they evolve over time.

The use of a fixed benchmark behavior to evaluate absolute fitness is somewhat misleading, because it reports high fitness ratings for configurations that are exceedingly effective against the benchmark without being a general solution.

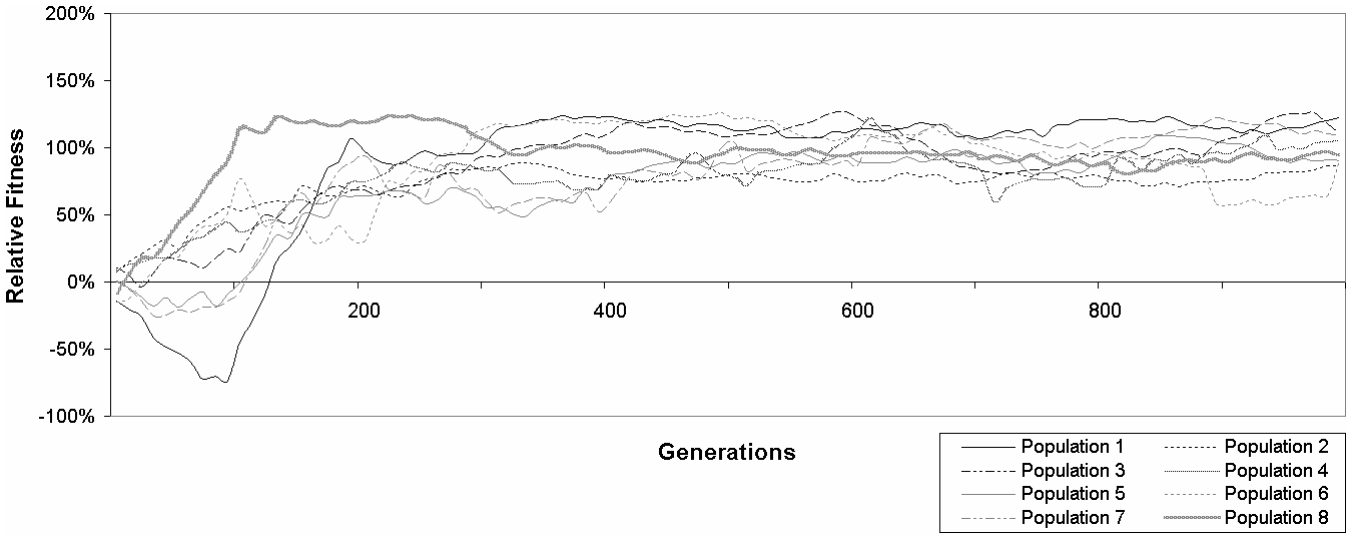

Figure 5a: Progression of eight individual populations, measured relative to the benchmark.

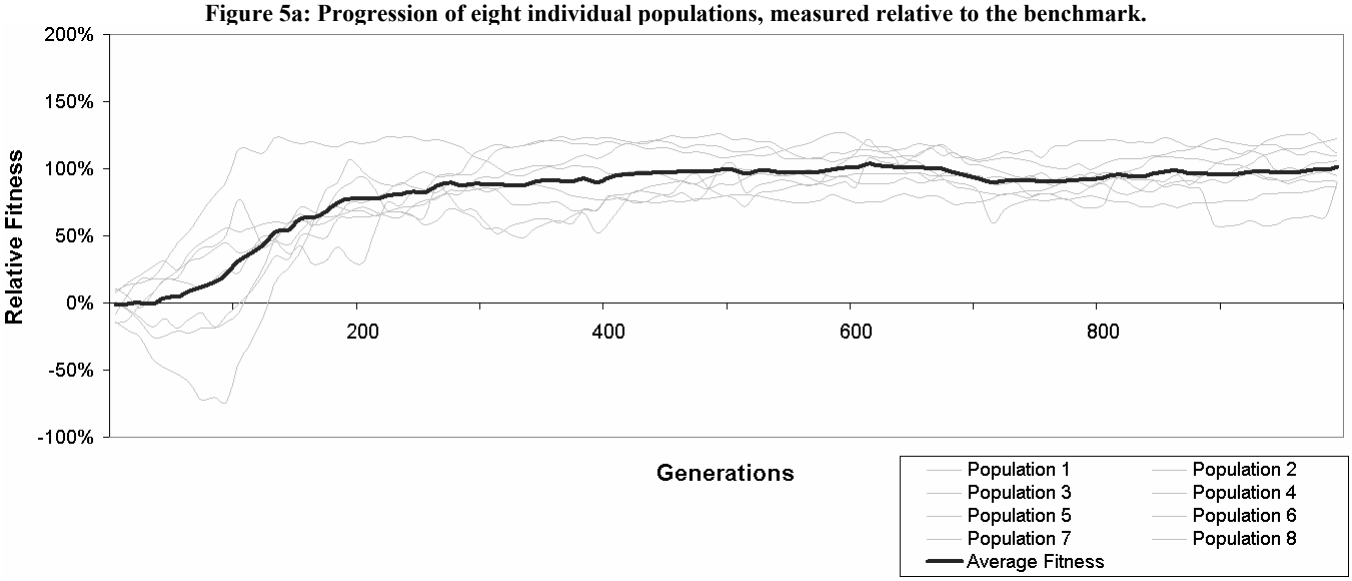

Figure 5b: Progression of average of fitness for all populations, measured relative to the benchmark.

This anomaly occurs in population 8, which initially favors a configuration that displays a high level of fitness against the benchmark (see generations 100 through 300 in Figure **5**a), but later abandons that family of configurations in favor of structures that are more successful in general. To reduce the affects of such anomalies and achieve a better indication of how the populations are progressing towards absolute fitness, the average progress of the eight populations is used. Figure **5**b presents the average progress of the eight populations as measured against the benchmark.

Looking at the progression of average fitness during the course of one-thousand generations, a notable period of improvement occurs during the initial two-hundred generations where fitness improves from a nominal rating of 0% to a rating of 78%. The remainder of the evolution is relatively stable, maintaining an average rating of 94% against the benchmark and ends with a rating of 101%.

While the evolution of eight independent populations converges on a variety of solutions, each structure captures a similar aspect of the Robocode domain. The populations naturally move towards somewhat passive solutions that are capable of attacking a target when conditions are favorable. This approach is effective because a robot must commit a fraction of its energy when shooting at an opponent. Like gambling, it benefits a robot to shoot when there is a reasonable expectation of hitting a target. If the shot misses, the committed energy is lost. If the shot hits a target, the target's energy is reduced by that amount and the shooter claims twice the energy committed. Observations made during the fitness evaluations in Stage III show that the aggressive nature of the benchmark behavior is self-defeating because it often fires from long distances where there is little expectation of scoring a hit. The more conservative behavior allows members to achieve high relative fitness ratings by simply evading the benchmark until it cripples itself by draining its own energy reserves.

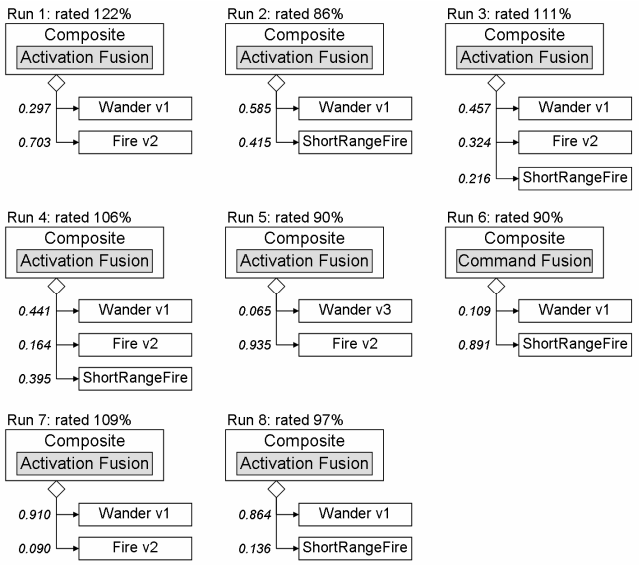

Figure 6: Behavior structures discovered from the evolution of eight randomly generated behavior populations.

The solution structures discovered by each of the eight populations are shown in Figure 6. At first glance, the common thread between the solutions is that they each employ a motion behavior and a tracking/shooting behavior joined by a fusion based arbiter. The use of a fusion based arbiter allows the robot to pursue multiple objectives simultaneously.

Conspicuously missing from the solutions above are the shooting behaviors: *Return Fire*, *Fire v1* and *Sniper Fire*. Having identified the importance of using a more conservative shooting approach, *Fire v1* and *Return Fire* are undesirable because they impose no range restriction and take unlikely shots at distant targets. The *Sniper Fire* behavior, a highly specialized behavior for shooting unmoving targets at long range, is likely to become obsolete because a population adopts continuous motion as a minimal requirement for survival.

Of the motion based behaviors, *Wander v2*, *Charge* and *Dodge* each fail to make an appearance in the solution set. *Wander v2*, which simulates Brownian motion, was intended to produce erratic movements that can not be effectively tracked by an opponent. In reality, it produces erratic motion in a localized area, making shots in the general direction more likely to score a hit. As noted above, somewhat passive behaviors are able to conserve their energy and achieve higher mortality rates, thus a behavior, like *Charge*, that moves our robot into an opponent's effective radius is also unfavorable. The absence of the *Dodge* behavior suggests that an ability to sustain continuous motion can act as a passive means of evading incoming attacks and indicates that such defensive measures are "good enough."

Observations of the solution structures in Figure 6 during battle shows that each is coherent, meaning that the behavior has the ability to perform basic elements of combat like tracking and shooting targets while moving within the battlefield without impeding its own progress towards the immediate goal and is able to consistently demonstrate a level of fitness that is superior to the benchmark. The real question is, "How good are these solutions on an absolute scale?"

To better understand how the eight solutions rank on an absolute scale, the eight solutions are compared in an eight-on-eight battle to discover the fitness of each solution structure relative to the others. This approach uses a series of 1,000 battles to create an inter-population fitness evaluation and the results are shown in Figure 7. Rather than separating into bands, where some solutions consistently achieve higher performance ratings than others, they are (with the exception of run 5) tightly interwoven, indicating that the solutions presented by the individual evolutions are equally matched. With a performance variance equal to the noise floor, seven of the resulting behavior structures are considered to be equivalent solutions.

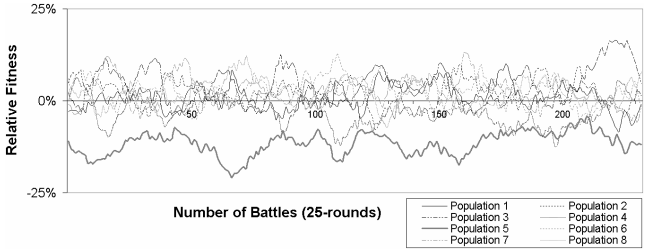

Figure 7: Relative fitness of the eight population runs, where seven of the solutions are considered equivalent.

The solutions presented by each run are relatively simple structures, lacking the depth and complexity typically associated

with genetic programming solutions. Each solution structure presents a clear pairing of one motion behavior with one or two shooting behaviors. The lack of multiple skills within successful structures indicates that the scope of the elemental behaviors is too large. The behaviors provided, while incomplete for the domain, prefer to act alone and do not act as generic operators that can be composed by an EA to form deeper and more intricate solution structures that have coherent outward operations.

5. CONCLUSIONS

The ability of the unified behavior framework (UBF) to simplify the development and testing of behaviors for a given domain is demonstrated through the use of a genetic program to automate the discovery of effective behavior structures from a pool of simple behavior and arbitration elements. In this experiment, a genetic program is used to discover combinations of elemental components that contribute to the robots motion and its ability to track and shoot targets. The ability of the UBF to support the composition and recombination of behavior structures by the genetic program validates its ability to form structures that are logically correct, if not semantically coherent for a given domain.

In robotic behavior based system development, the optimal solution is unknown and potentially changes with the introduction of new components. Along with the broad capabilities of the UBF, the use of a stochastic search discovers good solutions and is recommended as a useful tool for developing behavior based systems. The results show that this method is more effective than relying on raw human cleverness to achieve an optimal configuration directly. Additionally, the close relative fitness of the solution structures indicates that many equivalently good solutions exist within a domain, and that the approach is feasible for other robotic domains.

6. ACKNOWLEDGMENTS

The authors would like to acknowledge funding through AFRL/SNR Lab Task 06SN02COR from the Air Force Office of Scientific Research, Lt. Col. Scott Wells, program manager. The views expressed in this article are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

7. REFERENCES

[1] R. C. Arkin. “Behavior-based robot navigation for extended domains.” *Adaptive Behavior*, vol. 1, pp. 201-225, 1992.

[2] J. E. Baker. “Reducing bias and inefficiency in the selection algorithm.” *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and their Application*, pp. 14-21, 1987.

[3] V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. Cambridge, MA: MIT Press, 1984.

[4] R. A. Brooks. “A Robust Layered Control System for a Mobile Robot.” *IEEE Journal of Robotics and Automation*, vol. RA-2, pp. 14-23, 1986.

[5] R. A. Brooks. “New Approaches to Robotics.” *Science*, vol. 253, pp. 1227-1232, 1991.

[6] C. Coello Coello, D. Van Veldhuizen, and G. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. New York, NY: Kluwer Academic, 2002.

[7] J. Connell. “A Behavior-Based Arm Controller.” *IEEE Transactions on Robotics and Automation*, vol. 5, pp. 784-791, 1989.

[8] R. J. Firby, “Adaptive execution in complex dynamic worlds,” Ph.D. Dissertation, Yale University, YALEU/CSD/RR #672, 1989.

[9] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns*. Boston, MA: Addison-Wesley, 1994.

[10] E. Gat. “On Three-Layer Architectures.” *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, pp. 195 - 210, 1998.

[11] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[12] P. Husbands. “Genetic Algorithm in Optimization and Adaptation.” *Advances in Parallel Algorithms*, pp. 227 - 276, 1992.

[13] L. P. Kaelbling, “An Architecture for Intelligent Reactive Systems,” in *SRI International Technical Note No. 400*. Menlo Park, CA, 1986.

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

[15] P. Maes. “Situated Agents Can Have Goals.” *Robotics and Autonomous Systems*, vol. 6, pp. 49 - 70, 1990.

[16] M. Nelson. “Robocode Central.” *http://robocode.sourceforge.net*, 2006.

[17] J. Rosenblatt. “DAMN: A distributed Architecture for Mobile Navigation.” *the AAAI Spring Symposium on Lessons Learned for Implemented Software Architectures for Physical Agents*, pp. 167 - 178, 1995.

[18] J. Rosenblatt. “Utility Fusion: Map-Based Planning in a Behavior-Based System.” *Field and Service Robotics*, pp. 411 -418, 1998.

[19] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Dicipline*. Upper Saddle River, NJ: Prentice Hall, 1996.

[20] J. Togelius. “Evolution of a subsumption architecture neurocontroller.” *Journal of Intelligent and Fuzzy Systems*, vol. 15, pp. 15-20, 2004.

[21] J. Togelius and S. M. Lucas. “Evolving controllers for simulated car racing.” *Arxiv preprint cs.NE/0611006*, 2006.

[22] H. Utz, G. Kraetzschmar, G. Mayer, and G. Palm. “Hierarchical Behavior Organization.” *2005 International Conference on Intelligent Robots and Systems*, 2005.

[23] B. Woolley and G. Peterson, “Unified Behavior Framework for Reactive Robot Control,” Technical Report, Air Force Institute of Technology, WPAFB, OH, 2007.