In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010), pp. 439--446, Portland, Oregon, July 2010.

Evolving Agent Behavior In Multiobjective Domains Using Fitness-Based Shaping

Jacob Schrum
Department of Computer Science
The University of Texas at Austin
Austin, TX 78712 USA
schrum2@cs.utexas.edu

Risto Miikkulainen
Department of Computer Science
The University of Texas at Austin
Austin, TX 78712 USA
risto@cs.utexas.edu

ABSTRACT

Multiobjective evolutionary algorithms have long been applied to engineering problems. Lately they have also been used to evolve behaviors for intelligent agents. In such applications, it is often necessary to "shape" the behavior via increasingly difficult tasks. Such shaping requires extensive domain knowledge. An alternative is fitness-based shaping through changing selection pressures, which requires little to no domain knowledge. Two such methods are evaluated in this paper. The first approach, Targeting Unachieved Goals, dynamically chooses when an objective should be used for selection based on how well the population is performing in that objective. The second method, Behavioral Diversity, adds a behavioral diversity objective to the objective set. These approaches are implemented in the popular multiobjective evolutionary algorithm NSGA-II and evaluated in a multiobjective battle domain. Both methods outperform plain NSGA-II in evolution time and final performance, but differ in the profiles of final solution populations. Therefore, both methods should allow multiobjective evolution to be more extensively applied to various agent control problems in the future.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Connectionism and neural nets*

General Terms

Algorithms

Keywords

Multi-objective optimization, Neural networks, Multi-agent systems, Shaping

1. INTRODUCTION

Multi-Objective Evolutionary Algorithms (MOEAs) have long been used to discover useful trade-offs for challenging engineering problems [3, 10, 18]. Recently, another application of MOEAs has emerged: agent control, i.e. evolving behavior for intelligent agents in complex and noisy environments [1, 12, 19]. Whereas hand-designing agent behavior is challenging and time-consuming, evolution has proven to be an effective means of discovering such behavior automatically. Although agent control problems are different from engineering design problems, traditional MOEAs have been successful at solving them as well.

However, such success has often depended on some form of "shaping". The term shaping comes from behavioral psychology [13] and describes how animal behavior can be trained via a series of increasingly complex tasks. Animal behavior can also be shaped via biological evolution if successive generations face increasingly complex challenges [14].

Such shaping has been modelled by the evolutionary computation community using incremental evolution [11, 12], and division of problems into subproblems [19]. A progression of tasks helps agents evolve intermediate behaviors that eventually lead to success in the overall task. However, defining effective sets of tasks is error prone, and requires significant domain expertise that is difficult to come by.

An alternative to these task-based shaping methods is fitness-based shaping, defined here as any method that changes selection pressures to favor behaviors that will ultimately lead to good performance. Such intermediate behaviors may be inferior to others in the current population in terms of objective scores, but are needed by evolution in order to reach the behaviors with the highest performance. Unlike task-based shaping methods, fitness-based shaping can be effective in general, without extensive domain knowledge, as will be shown in this paper.

For agent control problems, the evolved representation is a policy defining how the agent behaves. These problems are essentially Reinforcement Learning problems [17], except such problems are not typically formulated in terms of multiple objectives. It turns out that such a formulation is very useful when evolving complex behavior.

In standard multiobjective optimization, the desired result is an approximation of the Pareto front, from which specific solutions can be chosen by a "decision maker" [6]. However, in agent control problems the desired result is good behavior, which means that the solutions are competent in each objective, i.e. pass a certain threshold. Defining success in terms of thresholds makes dynamic objective management possible. This paper introduces such a method: Targeting Unachieved Goals (TUG).

The threshold for each objective defines a goal. When

GECCO'10, July 7–11, 2010, Portland, Oregon, USA.

enough of the objective scores for the population have surpassed the objective's goal, the goal is considered achieved. TUG deactivates objectives whose goals have been achieved, so that only objectives with unachieved goals are part of the multiobjective selection process. Removing objectives with achieved goals from consideration encourages better performance in the more challenging objectives, resulting in efficient and domain-independent shaping.

Another possible approach to shaping is Behavioral Diversity (BD). This method was originally proposed for a much narrower role, i.e. as a way to bootstrap evolution in single-objective problems by including a second, domain-specific objective measuring behavioral uniqueness [8, 9]. In this paper, this idea is extended to a problem that already has multiple objectives, and a domain-independent way of defining behavioral uniqueness is proposed.

Both TUG and BD result in better performing agents with more complex behaviors, yet each is suitable to different types of problems. These methods demonstrate that domain-independent shaping can help extend multiobjective evolution to more challenging agent control problems.

2. MULTIOBJECTIVE EVOLUTION

There are several evolutionary methods designed to solve multiobjective problems by approximating the Pareto front, including SPEA2 [24], PESA-II [4] and NSGA-II [5]. The algorithm used in this work is NSGA-II, though the shaping methods presented below should be applicable to other MOEAs as well.

In this paper, NSGA-II is used to evolve neural networks, which implement agent behavior. In this section, NSGA-II is briefly reviewed, followed by details on the evolution of neural networks.

2.1 NSGA-II

Like all MOEAs, NSGA-II is based on the principle of Pareto dominance. Vector $\vec{v} = (v_1, \dots, v_n)$ dominates $\vec{u} = (u_1, \dots, u_n)$ iff:

1. $\forall i \in \{1, \dots, n\} : v_i \geq u_i$, and
2. $\exists i \in \{1, \dots, n\} : v_i > u_i$.

The expression $\vec{v} \succ \vec{u}$ denotes that $\vec{v}$ dominates $\vec{u}$. Vector $\vec{v}$ within population $\mathcal{F}$ is said to be non-dominated if there does not exist any $\vec{x} \in \mathcal{F}$ such that $\vec{x} \succ \vec{v}$. The vectors in $\mathcal{F}$ that are non-dominated are said to be Pareto optimal, and make up the non-dominated Pareto front of $\mathcal{F}$.

NSGA-II sorts individuals with respect to domination. Individuals in the Pareto front have a rank of 1, indicating that they are the best in a multiobjective sense. Individuals that are only dominated by solutions with a rank of 1 are the next best, and are assigned a rank of 2, and so on. Given these rankings, NSGA-II repeatedly carries out an elitist $(\mu + \lambda)$ selection procedure [2]: μ parents give rise to λ children, and selection is performed on the combined parent/child population to get the next μ parents. Better-ranked individuals are favored in selection, with ties broken based on a metric called "crowding distance" [5].

Unlike the original NSGA-II, selection in this paper is not used to create the λ children of the $(\mu + \lambda)$ selection procedure. The child population is formed instead by cloning each individual member of the parent population, then probabilistically mutating them. This process is faster and sufficient for the current experiment.

2.2 Neuroevolution

Neuroevolution is the application of an evolutionary algorithm to artificial neural networks. In this paper, a neuroevolution method similar to NEAT (Neuroevolution of Augmenting Topologies [16]) is used, adapted to support multiobjective evolution. Networks are represented using a direct encoding that allows for arbitrary network topologies. They are stored as lists of neurons with weighted pointers to other neurons.

The initial population of networks consists of individuals with no hidden layers, i.e. only input and output nodes. Furthermore, these networks are sparsely connected in a style similar to Feature Selective NEAT [21]. Initializing the networks in this way allows them to easily ignore any inputs that are not, or at least not *yet*, useful.

In order to support multiobjective evolution, the selection process of NSGA-II is used instead of the fitness assignment and selection procedures of NEAT. Therefore, speciation and fitness sharing are not used.

After the cloning stage of NSGA-II, mutations are probabilistically applied to the cloned neural networks. As in NEAT, there are mutations to perturb the weights of existing connections, add new (potentially recurrent) connections between existing nodes, and splice new nodes along existing connections. Crossover is not used because preliminary experiments using a crossover operator similar to the one used in NEAT actually decreased performance.

The networks evolved by the above method serve as the brains of agents used in a multiobjective task. The inputs for a network come from the agent's sensors and the network outputs drive its actuators, as described in section 4. Through evolution, networks gradually emerge that implement intelligent behavior with respect to multiple objectives.

NSGA-II neuroevolution as described above is the *control* condition for experiments presented below. This version of NSGA-II is independently augmented with TUG and BD methods for shaping as described next.

3. SHAPING METHODS

The two methods presented in this section are used independently to implement multiobjective fitness-based shaping in agent control problems. Their performance is evaluated experimentally in section 5.

3.1 Targeting Unachieved Goals

The selection process in NSGA-II can be improved by dynamically managing which objectives are used in calculating the successive Pareto fronts of the population. This choice is guided by the principle that only objectives in which the population is having trouble performing should be part of the selection process; objectives in which the entire population is performing well can be ignored.

To know when to deactivate an objective, a numeric goal is defined for each objective. These values represent desired levels of performance, and are a way of specifying how well an agent would have to perform to be considered successful.

A goal is considered achieved once the average performance of the population in that objective has *persisted* long enough at a level above the value of the goal. Persisting above the goal means both the average performance and a recency-weighted average of that average have surpassed their objective's goal value. The persistence requirement assures that goal achievement is not the result of luck.

Formally, a recency-weighted average r_t at time t is updated according to $r_{t+1} \leftarrow r_t + \alpha(\bar{x}_{t+1} - r_t)$, where α defines what portion of the distance between r_t and the current actual average, denoted by $\bar{x}_{t+1}$, that r_t should be increased by. Thus the recency-weighted average moves slightly closer to the most recent average every generation.

Note that objectives can be reactivated if performance drops back below the goal. Goals reactivate as soon as the *actual* average drops below the goal, since the recency-weighted average catches up too slowly in comparison.

As to how the goal values are set, it turns out that appropriate values can be learned automatically with little domain knowledge. TUG defines a gradual way of increasing the goal values that does not overshoot the capabilities of agents within the domain. For each objective, TUG moves the current goal closer to the current maximum score in that objective every time the population achieves all goals. Formally, the update rule is $g_o \leftarrow g_o + \eta(o_{\max} - g_o)$, where g_o is the goal for objective o, $o_{\max}$ is the current maximum score within the population for objective o, and η defines what portion of the distance between the current goal and the current maximum the goal should be increased by.

The maximum score in an objective will always be above the average, which will always be above the goal at the moment it is achieved, so TUG will always *increase* goals. Because goals are moved towards the current maximum score, no goal will ever be set at a level which is unattainable. Furthermore, the goals are only increased when all of them are achieved. Whenever all goals are achieved, all recency-weighted averages are also reset, so that the population must prove it can perform well given the new goals.

In this manner, all initial goals can be set at low values easily attainable by the population. TUG then automatically increases the goal values based on the capabilities of the population, eventually leading to maximum performance. Therefore, TUG requires almost no domain knowledge.

Another way to achieve fitness-based shaping in multiobjective problems is by using a behavioral diversity objective, as will be described next.

3.2 Behavioral Diversity

Given a single-objective problem, one can encourage the population to explore the space of possible solutions by transforming the problem into a two-objective problem with a behavioral diversity objective [8, 9]. This method was originally proposed as a bootstrapping algorithm because the behavioral diversity objective helps the population find solutions when it would otherwise stagnate. However, the behavioral diversity objective actually shapes learning throughout the entire course of evolution. Although this method has so far been used to augment single-objective problems only, a behavioral diversity objective can just as well be added to a problem that is already defined using multiple objectives.

In order to use a behavioral diversity objective, first a way of assigning behavior vectors to each individual must be defined. The actual behavioral diversity objective score is then taken to be the average Euclidean distance between a genome's behavior vector and those of all other genomes in the population. Originally, Mouret et al. suggested that behavior vectors had to be domain specific [8, 9]. In contrast, this paper proposes a domain-independent way of using the behavioral diversity objective for any domain in which agent behavior is defined by a mapping from real-valued input vectors to real-valued output vectors.

Each generation, ten random input vectors are generated. Each input vector is presented to the control policy of each genome in the population in the same order (in case the policy representation supports some form of memory), and the output vectors are saved. Given these output vectors, the behavior vector for a specific genome is simply the concatenation of its output vectors. Defining behavior vectors in this way is simple and effective.

Interestingly, the randomly generated input vectors do not even need to represent possible or likely inputs, though it seems likely that differences in behavior will be better highlighted by input vectors that an agent might actually experience in the domain. Some potentially better approaches to defining behavior vectors are suggested in section 6. However, random input vectors serve well enough for this initial experiment. The details of the domain used in this work are described next.

4. BATTLE DOMAIN

The battle domain has previously been used to show the advantages of multiobjective evolution over the weighted-sum approach in challenging agent control tasks [11]. An illustration of the domain is shown in figure 1. Several monster agents evolve to attack a scripted fighter agent. The fighter swings a bat that damages monsters on impact, and the monsters damage the fighter on impact. Agents are knocked backwards whenever they take damage. All agents have limited health points, and die after taking too much damage, though the fighter immediately respawns, thus allowing evaluation to continue.

The version of the battle domain used in this paper is slightly more challenging than before because monster teams consist of only four members (in contrast to the original 15). The team composition is also different in that each monster has a copy of the same policy, i.e. the teams are homogeneous. Third, instead of task-based shaping via hand-designed incremental evolution, which was used in [11] to overcome the challenging scripted fighter agent, TUG and BD will be used for fitness-based shaping.

The behavior of the fighter is simple yet challenging. The fighter moves constantly forward, always swinging its bat and turning to pursue the nearest monster that it sees. The fighter can only see monsters in front of it, so it ignores near monsters that are behind it. If no monsters are in front of the fighter, it turns until it finds one. Because the fighter constantly swings its bat, a frontal attack by a monster is difficult, but possible with precise timing. However, teamwork can also help the monsters overcome this challenge.

Monsters are controlled via neural networks with 38 inputs: a constant bias; the angles to the fighter and each teammate; the differences in headings from the fighter and each of the teammates; indicators of when the monster is in front of the fighter, and when the fighter is reeling from taking damage; indicators for when the monster is close and very close to the fighter's bat; brief signals whenever the sensing monster is injured or damages the fighter, whenever *any* monster is injured or damages the fighter, and whenever each particular teammate damages the fighter. The last 15 inputs come from three sets of five sensors spread out in front of each monster. One of these sets can detect the fighter, the second other monsters, and the third the bat.

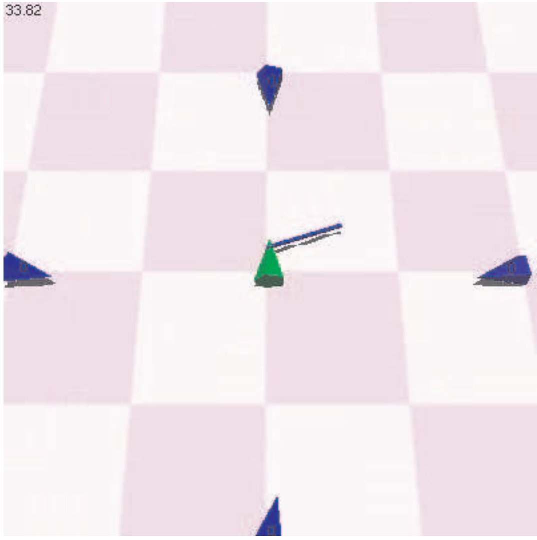

Figure 1: Battle Domain. The fighter (center) starts an evaluation surrounded by four monster agents. The fighter approaches the nearest monster while swinging its bat. If the bat hits a monster, the monster is damaged; if it receives enough damage, it dies. If a monster hits the fighter, the fighter is damaged. The monsters must evolve to deal damage to the fighter, avoid damage from the fighter's bat, and survive as long as possible. This is a challenging multiobjective task that is difficult to learn without some form of shaping.

Two network outputs define a monster's behavior: one controls the forward/backward impulse, and the second controls the left/right turning.

Using these inputs the monsters must fulfill multiple contradictory objectives. They must avoid the fighter's bat, since they die from too many hits. If death is unavoidable, then it should be avoided for as long as possible. If the fighter can kill monsters quickly, then it can keep shifting its focus to new monsters, and kill even more. Finally, monsters must maximize the damage that they as a group deal to the fighter. It is not individual scores, but the group score that matters. Therefore, the monsters must work together to maximize group score. Given these objectives, the following three fitness measures are designed to measure success:

1. *Maximize Damage Dealt:* Every time a monster contacts the fighter, the fighter loses 10 health points. The amount of damage dealt is attributed to the team, regardless of which individual dealt the damage.
2. *Minimize Damage Received:* Every time the fighter strikes a monster with its bat, the monster takes 10 points of damage, making for a resulting change in health of -10. The fitness attributed to the team is the average change in health across all individuals.
3. *Maximize Time Alive:* There are 600 time steps in each trial. For each individual monster, this objective measures the number of time steps that the monster is alive. The team score in this objective is the average across team members.

These objectives are similar to those in [11], except the intuitive *Maximize Damage Dealt* objective replaces the comparatively convoluted *Attack Assist Bonus* used in that study. *Attack Assist Bonus* assigned fitness to individuals near the fighter whenever any monster dealt damage to it. This measure was used to overcome the team credit assignment problem that results from evolving heterogeneous teams with individual-level selection for a task involving teamwork. Since the teams in this paper are homogeneous and selected at the team level, credit assignment is not an issue. The decision to use homogeneous teams is partly based on [20], which demonstrated that homogeneous teams are better for tasks requiring teamwork and altruism. Because of team-level selection, all the objectives described above are team-centric.

An initial set of goal values for these objectives indicates a reasonably good level of performance:

1. *Maximize Damage Dealt* $= 50$*:* The fighter has 50 health points, so this goal requires the monsters to kill the fighter at least once per trial. The fighter respawns after death, giving the monsters a chance to inflict more damage.
2. *Minimize Damage Received* $= -20$*:* Bat strikes deal 10 damage points each, so each monster should take no more than two hits on average. However, because this value is averaged across team members, it is possible to achieve this goal even if one team member dies (50 damage), since the average across the four team members could still be above -20.
3. *Maximize Time Alive* $= 540$*:* On average across team members, monsters must survive throughout 90% of the trial. It is still possible to achieve this goal even if some NPCs die in fewer than 540 iterations (the total number of iterations per trial is 600).

These goal values were found to be appropriate in preliminary runs. With regards to how these values are used by TUG, lower values could safely be used at the expense of slowing down evolution slightly.

5. EXPERIMENTS

Having described the fitness-based shaping methods to be evaluated and the multiobjective domain that they will be evaluated in, the next step is to describe *how* these methods will be experimentally evaluated.

5.1 Setup

In these experiments, three methods are tested: NSGA-II by itself (Control), NSGA-II combined with TUG, and NSGA-II combined with BD. Each method was evaluated in 30 trials. The parent and child population sizes were $\mu = \lambda = 52$, and trials executed for 500 generations. Every network was evaluated three times, and its objective scores were averaged across evaluations in order to get more reliable scores in the face of noisy evaluations.

With respect to goal achievement, the particular recency-weighted average step-size constant used was $\alpha = 0.15$. Goal achievement was monitored in all runs, though it only influenced evolution in runs using TUG. Also important to TUG was the step-size constant by which goal values increased, which was $\eta = 0.15$.

In runs using BD, completely random vectors within a limited range were used as the input vectors for generating the output vectors necessary to define behavior vectors.

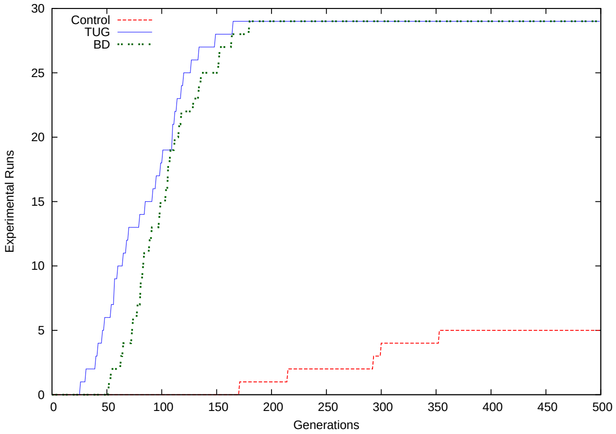

Figure 2: Number of successful runs (out of 30) for each generation for each method. A run is considered successful once its population has achieved all initial goals for the battle domain. Both TUG and BD runs succeed quickly in early generations with TUG only slightly better than BD, whereas the Control plot (i.e. plain NSGA-II) is at the bottom with only has a few successful runs.

5.2 Results

NSGA-II combined with either TUG or BD significantly outperforms NSGA-II in creating populations of individuals with successful and interesting behaviors.

The first measure of success is the number of generations required by the population to achieve all goals. All trials evolved past this point, but assuming good goal values, the point where all goals are achieved can be thought of as a reasonable stopping point for evolution. These results are shown in figure 2. TUG and BD perform similarly well, with TUG slightly outperforming BD. Both methods outperform plain NSGA-II by a large margin.

Only one TUG run and one BD run failed to achieve all goals within 500 generations. In contrast, 25 of the plain NSGA-II runs failed. These failed runs mean that average completion times cannot be compared. However, median tests can still be used, and they indicate that TUG is significantly faster than the control ($\chi^2(1, n = 60) = 52.2667, p < 0.001$) by a large amount ($\phi = 0.933$), and that BD is significantly faster than the control ($\chi^2(1, n = 60) = 52.2667, p < 0.001$) by a large amount ($\phi = 0.933$). The difference between TUG and BD is not significant ($\chi^2(1, n = 60) = 1.0667, p = 0.302$).

A second measure of success is the number of *individuals* in the population whose objective scores surpass the initial goal values of each objective. These results are shown for the final generation in figure 3. Once again, both TUG and BD significantly outperform plain NSGA-II ($p < 0.05$). Furthermore, in most generations TUG significantly outperforms BD ($p < 0.05$). This result is not surprising since BD intentionally allows poor-performing individuals to persist in the population if their behavior is significantly different from the behavior of all other individuals in the population.

TUG runs tend to have more successful individuals in the population, and in two runs all 52 networks in the population, after the final generation, surpassed all goals. However, the minimum number of successful individuals in a TUG run is less than BD's minimum. Observation of learning curves from individual TUG runs (figure 4) indicates that TUG runs go through cycles in which the number of successful individuals is maximized at the point where all goals are achieved, and then minimized soon afterwards when all goals are increased and reactivated. Therefore, it is best to terminate TUG at a point where all goals have just been achieved rather than after a fixed number of generations.

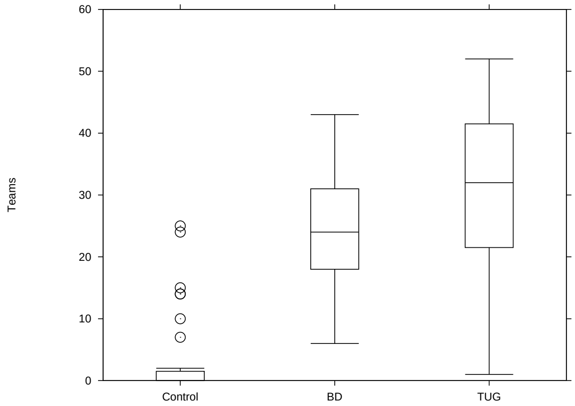

Figure 3: Box-and-whisker plots of the number of successful networks/teams in the final parent populations (each of size 52) for each method. These plots show the lower quartile, median, and upper quartile as the lower boundary, center line, and upper boundary of each box respectively, and the whiskers denote the furthest points within $1.5IQR$ of the nearest quartile, where IQR is the interquartile range. Points outside of the whiskers are outliers. A network is successful if its objective scores surpass all goal values defined in section 4. Both TUG and BD result in significantly more successful individuals within the population than NSGA-II alone (Control). All of the control's high scores are outliers. TUG runs produce populations with the most successful individuals, though they also have the most variance.

The methods can also be compared in terms of summary attainment surfaces [7]. Such a surface shows the portion of objective space that is dominated by solutions from a given percentage of runs of the same method. The surfaces shown in figure 5 are the best, median, and worst summary attainment surfaces for each of the three methods. The worst surface contains the region dominated in all runs, whereas the median surface indicates what portion of objective space is dominated by the Pareto fronts from 50% of runs. The best surface dominates a given region of objective space if *any* of the runs dominated that region. Each row shows TUG dominating more volume than the control, and BD dominating more volume than TUG.

The superiority of BD's Pareto fronts can also be measured in terms of hypervolumes, i.e. the portion of the objective space that is dominated by the Pareto front [23]. Hypervolume is one of few Pareto-compliant metrics [22], meaning that if one Pareto front completely dominates another, then it will have a larger hypervolume. The hypervolume plots show that BD fronts have the largest hypervolumes, followed by TUG and then the control (figure 6).

To gain further insight into these results, representative behaviors generated by each method will be analyzed next.

5.3 Behaviors

As expected from the results above, the behaviors evolved

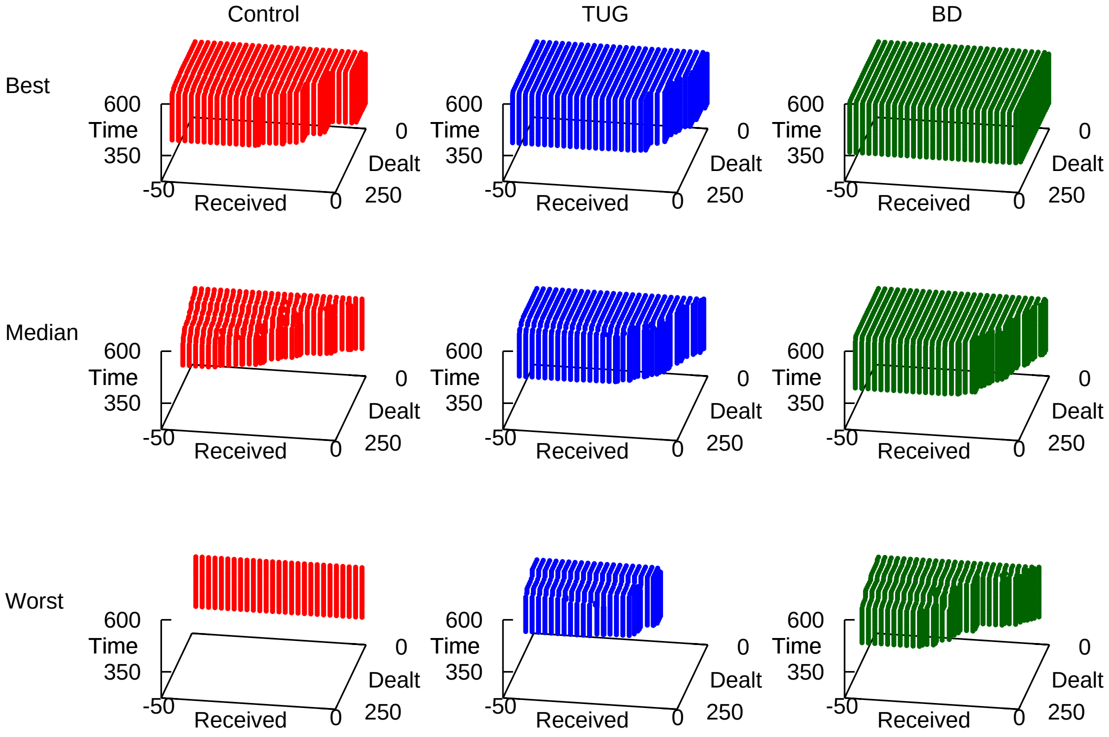

Figure 5: Summary attainment surfaces for each method. The columns correspond to Control, TUG and BD, and the rows show the best, median and worst summary attainment surfaces for each method. In terms of the best surface, TUG is only slightly better than Control, but BD dominates considerably more area than either. In terms of the median surface, the coverage clearly improves from the Control to TUG, and from TUG to BD. In terms of the worst surface, the Control is terrible, TUG is much better, and BD is the best.

with TUG and BD tend to be better than those evolved with plain NSGA-II. Movies of characteristic behaviors can be viewed at: http://nn.cs.utexas.edu/?fitness-shaping

The most effective behavior occurred in a BD run. Across the three trials that the team faced, it avoided all damage and thus stayed alive the whole time. However, this team is exceptional in that it also dealt an average of 240 damage, which amounts to killing the fighter nearly five times. This team's trick is to rush at the fighter, pause for just the right amount of time as its bat swings past, then rush in to attack the fighter repeatedly until it dies. Somewhat similar behaviors emerged in TUG and even in control runs, but they received damage because very precise timing is needed in order to avoid the bat. Apparently the behavioral diversity objective makes it easier for evolution to fine-tune network weights by encouraging networks to differentiate.

Another behavior that is good at both avoiding and dealing damage is based on a clever turning maneuver at the start of each trial. The monster moves towards the fighter while turning left, and after it barely dodges a bat swing the monster starts backing into the fighter. The monster then continues to strike while turning, and thereby manages to avoid the bat. This behavior is slightly less efficient than the one above due to the turning motion. Since the maneuver takes more time to execute, monsters cannot deal as much damage in the allotted time. This behavior was common in both TUG and BD runs, but not in control runs.

A set of behaviors popular in TUG runs, but not in BD or control runs, employed baiting motions by one of the monsters, similar to those observed in the original work in the battle domain [11]. A monster backs away from the fighter while turning such that it has a greater risk of being hit by the bat, yet also slows down the progress of the fighter so that teammates can sneak up from behind to attack. Though this behavior requires teamwork and is visually compelling, it is not as efficient as the first two behaviors because extra time is required to successfully move the fighter into a vulnerable position via baiting. There is also a greater risk that one of the monsters will be hit.

The TUG runs also evolved an effective coordinated counter-clockwise attack behavior. Because the fighter swings its bat from right to left, it is safer to attack it on its left side. Starting from the monster on which the fighter currently focuses, and moving counter-clockwise around the fighter, the next monster will always be in a position to easily attack the fighter on its left side. Therefore, the monsters will be able to repeatedly blindside the fighter. This behavior is generally effective, but tends to result in large damage received because it is hard to get it coordinated precisely.

Control runs were characterized primarily by reckless behaviors that would sacrifice life in order to deal damage, and cowardly behaviors that would run away to avoid damage. Often different members of the same population exhibit these opposing behaviors, which makes sense given that they represent different trade-offs between objectives. However,

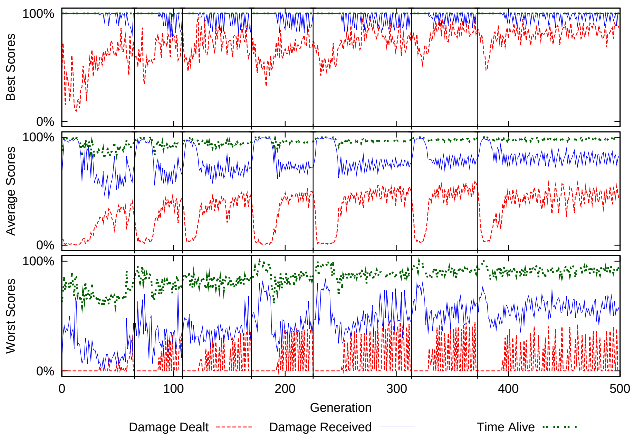

Figure 4: The best, average, and worst scores in each objective by generation for an individual run using TUG. Objective scores are normalized to a common range and measured as percentages of the maximum attained values. Vertical lines on the plot signify generations at which TUG achieved all goals, and thus had its goal values increased. Remember that all objectives are reactivated after goals are increased. The plot for Time Alive is always on top, since it is the easiest objective to perform well in. Damage Received is just beneath Time Alive, and its plot fluctuates in a similar manner because Time Alive is also affected by receiving damage. Damage Dealt is clearly the hardest objective to perform well in, since its plot is always beneath the plots of the other objectives. Damage Dealt drops after each vertical line because the reactivation of the other two objectives makes it hard to deal damage, yet as the goals for Damage Received and Time Alive are achieved, selection focuses more on Damage Dealt until all goals are achieved. Therefore, the population scores are always at their best on generations when all goals are achieved.

some control trials did achieve all goals, usually by approximating the coordinated attack behavior described above.

It is interesting that, for the most part, TUG and BD succeeded in different ways. Successful BD behavior consists of well-timed maneuvers for which only the actions of a single agent mattered. The behavioral diversity objective helped networks fine-tune their behavior such that they perform well on their own. TUG runs that produced similar behavior did not perform as well as BD, because the timing was not as good. In contrast, TUG runs produced groups of agents that exhibit teamwork, which is in some ways more interesting despite the fact that such teams did not score as well. In the battle domain, teamwork means risking oneself for the sake of the team. The result was more damage received due to teamwork. However, TUG agents do not mind such damage as long as the current damage received goal is still achieved. This orientation towards teamwork may be more useful than sheer performance in some applications, such as developing behavior for non-player characters in video games.

6. DISCUSSION AND FUTURE WORK

Both TUG and BD are successful in shaping evolution in a challenging multiobjective agent control problem. Interestingly, they do it in a different manner: TUG allows evolution to quickly and reliably find populations full of individuals that perform well across objectives, while BD leads to better final Pareto fronts.

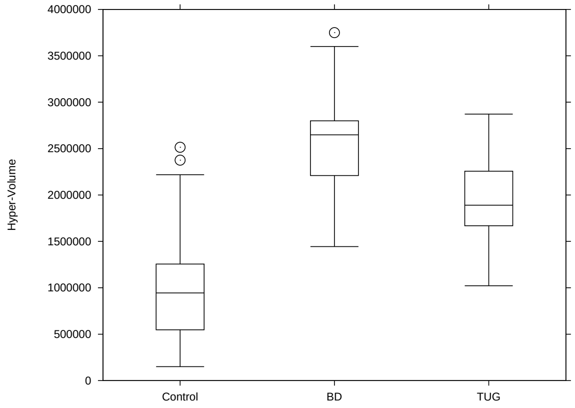

Figure 6: Box-and-whisker plots of hypervolume values for the three methods. Hypervolumes were calculated with respect to the following reference point: -10 for damage dealt, -60 for damage received, and 350 for time alive. This point was chosen because its values are just slightly smaller than the minimal values across points in all Pareto fronts. The Pareto fronts of the control condition dominate little area, with TUG dominating more and BD usually dominating the most.

Because its Pareto fronts are better, BD is the better method by most standards. However, BD was inferior to TUG in terms of the number of individuals in the population whose objective scores surpassed all goal values. Therefore, it is the better method in applications in which the entire population, or at least more individuals than make up a Pareto front, are required as the solution to a problem. For example, in the NERO game [15], the entire evolving population works together as a team to accomplish training tasks. Also, the original work in the battle domain [11] pitted larger heterogeneous teams composed of several different neural networks against the fighter. Therefore TUG may yet have important applications that BD cannot fulfill.

Since TUG and BD are largely orthogonal, it should be possible to use both at the same time. However, preliminary experiments indicate that a naïve combination of the two methods lowers rather than heightens performance. In these experiments, the behavioral diversity objective was always active. One issue to be addressed is whether TUG should ever deactivate the behavioral diversity objective, and if so, what its goal should be. Further analysis is required to understand how the methods interact in order to find ways for them to leverage each other instead of cancel out.

The generalized implementation of behavioral diversity proposed in this paper is also worth further study on its own. In this paper, ten randomly generated input vectors were used in each generation to produce outputs that made up the behavior vectors, but there are many ways to refine this approach. For example, the number of random input vectors can be changed to see whether a lesser or greater resolution make the method more effective. Also, a more structured method of choosing the input vectors could result in more meaningful behavioral distinctions. For example, instead of random input vectors, a syllabus of vectors tailored to the domain could be used. Another idea, which would not require any domain expertise, is to derive the behavior vectors by randomly sampling from input vectors that are actually experienced by agents during evaluation.

While the results in this paper lead to clear conclusions, it is important to keep in mind that so far they only pertain to the battle domain. Further experiments will need to be conducted in other domains to fully explore the strengths and weaknesses of TUG and BD. On the other hand, despite being designed for agent control problems, these methods of fitness-based shaping might also be useful in solving multiobjective problems in other types of domains.

In any case, provided the extensions above, it may be possible to improve the methods further and solve even harder problems in the future.

7. CONCLUSION

Two methods for shaping evolution of behavior in multiobjective agent control problems are presented. These methods require little to no domain knowledge, and are thus more attractive than previous methods.

First, Targeting Unachieved Goals speeds up evolution by focusing selection on the hardest objectives until the population achieves goals associated with each objective. With low initial goals, TUG automatically increases the challenge, meaning no domain expert is needed to set good goal values. Second, behavioral diversity, previously used in a domain-dependent way for single-objective problems [8, 9], speeds up evolution by encouraging exploration of new behaviors, which results in superior Pareto fronts. In this paper, behavioral diversity is measured in a domain-independent way, meaning that no domain expert is needed to define behavior vectors. Because domain experts are not needed, it should be easy to apply these methods to different domains.

TUG and BD were combined with NSGA-II, and each was shown to significantly outperform plain NSGA-II in a multiobjective battle domain. They have complimentary strengths, making them appropriate for different tasks. Both methods should extend multiobjective evolution beyond typical engineering problems to challenging agent control tasks.

8. ACKNOWLEDGMENTS

This research was supported in part by NSF grants IIS-0915038 and IIS-0757479, and by Texas Higher Education Coordinating Board grant 003658-0036-2007.

9. REFERENCES

[1] A. Agapitos, J. Togelius, S. M. Lucas, J. Schmidhuber, and A. Konstantinidis. Generating diverse opponents with multiobjective evolution. In *CIG'08*, 2008.

[2] T. Bäck, F. Hoffmeister, and H.-P. Schwefel. A survey of evolution strategies. In *ICGA IV*, pages 2–9, 1991.

[3] M. Chu, D. J. Allstot, J. M. Huard, and K. Y. Wong. NSGA-based parasitic-aware optimization of a 5GHz low-noise VCO. In *ASP-DAC'04*, pages 169–174, 2004.

[4] D. W. Corne, N. R. Jerram, J. D. Knowles, and M. J. Oates. PESA-II: Region-based selection in evolutionary multiobjective optimization. In *GECCO'01*, pages 283–290, 2001.

[5] K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. *PPSN VI*, pages 849–858, 2000.

[6] C. M. Fonseca and P. J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *ICGA V*, pages 416–423. Morgan Kaufmann, 1993.

[7] J. Knowles. A summary-attainment-surface plotting method for visualizing the performance of stochastic multiobjective optimizers. *ISDA V*, 2005.

[8] J.-B. Mouret and S. Doncieux. Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity. In *CEC'09*, 2009.

[9] J.-B. Mouret and S. Doncieux. Using behavioral exploration objectives to solve deceptive problems in neuro-evolution. In *GECCO'09*, pages 627–634. 2009.

[10] F. Pettersson, H. Saxen, and K. Deb. Genetic algorithm-based multicriteria optimization of ironmaking in the blast furnace. *Materials and Manufacturing Processes*, 24(3):343–349, 2009.

[11] J. Schrum and R. Miikkulainen. Constructing complex NPC behavior via multi-objective neuroevolution. In *AIIDE'08*, pages 108–113, 2008.

[12] J. Schrum and R. Miikkulainen. Evolving Multi-modal Behavior in NPCs. In *CIG'09*, pages 325–332, 2009.

[13] B. F. Skinner. *The Behavior of Organisms*. B. F. Skinner Foundation, Morgantown, WV, 1938.

[14] B. F. Skinner. The shaping of phylogenic behavior. *Journal of the Experimental Analysis of Behavior*, 24(1):117–120, July 1975.

[15] K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Evolving neural network agents in the NERO video game. In *CIG'05*, 2005.

[16] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002.

[17] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[18] Y. Tang, P. Reed, and T. Wagener. How effective and efficient are multiobjective evolutionary algorithms at hydrologic model calibration? *Hydrology and Earth System Sciences*, 10:289–307, 2006.

[19] N. van Hoorn, J. Togelius, and J. Schmidhuber. Hierarchical controller learning in a first-person shooter. In *CIG'09*, pages 294–301, 2009.

[20] M. Waibel, L. Keller, and D. Floreano. Genetic Team Composition and Level of Selection in the Evolution of Multi-Agent Systems. *Evolutionary Computation*, 13(3):648–660, 2009.

[21] S. Whiteson, P. Stone, K. O. Stanley, R. Miikkulainen, and N. Kohl. Automatic feature selection in neuroevolution. In *GECCO'05*, 2005.

[22] E. Zitzler, D. Brockhoff, and L. Thiele. The Hypervolume Indicator Revisited: On the Design of Pareto-compliant Indicators Via Weighted Integration. In *EMO'07*, volume 4403, pages 862–876, 2007.

[23] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms - a comparative case study. In *PPSN V*, pages 292–304, 1998.

[24] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *Evolutionary Computation*, 1999.