Learning Behavior Characterizations for Novelty Search

Elliot Meyerson[1,3] Joel Lehman[2] Risto Miikkulainen[1,3]
[1]The University of Texas at Austin; [2]IT University of Copenhagen; [3]Sentient Technologies, Inc.
ekm@cs.utexas.edu, jleh@itu.dk, risto@cs.utexas.edu

ABSTRACT

Novelty search and related diversity-driven algorithms provide a promising approach to overcoming deception in complex domains. The *behavior characterization* (BC) is a critical choice in the application of such algorithms. The BC maps each evaluated individual to a *behavior*, i.e., some vector representation of what the individual *is* or *does* during evaluation. Search is then driven towards diversity in a metric space of these behaviors. BCs are built from hand-designed features that are limited by human expertise, or upon generic descriptors that cannot exploit domain nuance. The main contribution of this paper is an approach that addresses these shortcomings. Generic behaviors are recorded from evolution on several training tasks, and a new BC is learned from them that funnels evolution towards successful behaviors on any further tasks drawn from the domain. This approach is tested in increasingly complex simulated maze-solving domains, where it outperforms both hand-coded and generic BCs, in addition to outperforming objective-based search. The conclusion is that adaptive BCs can improve search in many-task domains with little human expertise.

Keywords

Novelty Search; Behavior Characterization; Diversity; Search

1. INTRODUCTION

As optimization problems and domains become more complex they also tend to become more *deceptive*, i.e., it becomes increasingly difficult to craft an objective function that enables an evolutionary algorithm (EA) to discover the stepping stones ultimately leading to a desired solution [21, 28]. Because it is difficult to scale such objective-based search, there has been recent interest in diversity-driven algorithms [3, 13, 14, 19, 22, 26]. In such algorithms, the *behavior* of an individual is a vector representation of what the individual does during evaluation with respect to some given dimensions of interest; the *behavior characterization* (BC) is the function that maps each individual to its behavior. Diversity is then rewarded in a metric space of all possible behaviors, e.g., induced by Euclidean distance. Diversity-driven algorithms are less susceptible to deception, because they promote innovations, and discourage convergence towards qualitatively homogeneous populations.

Because diversity-driven algorithms are less focused on achieving particular goals, BCs can generalize across tasks drawn from the same domain, even when such tasks are deceptive. For the purposes of this paper, a *task* encodes a particular evaluation environment and fitness function, and a *domain* is a (possibly infinite) set of tasks sharing significant underlying structure and for which the same BC can be applied. As an example, consider the canonical novelty search domain: robot maze navigation. Given a maze (task), the goal is to find a controller that can lead a robot from the starting position to the goal in a limited time frame. Because of this time constraint, specialized controllers are required to solve different mazes.

The maze domain is a simple (and commonly used) proxy that captures the general tendency of domains to become more deceptive as they become more complex. For example, in circuitous mazes, minimizing the distance to the goal directly may cause convergence to a cul-de-sac, from which reaching the goal requires first traveling further away from it. To avoid such deception, fitness functions specialized to particular mazes can be constructed to guide evolution gently along the correct path. However, such specialized fitness functions are unlikely to generalize across broad classes of mazes, e.g., ones in which correct paths have different characteristics. On the other hand, the standard hand-coded BC for this domain, i.e., the final (x, y) position, is more robust across mazes, because it reflects exploration, and thus can be used to avoid deception in mazes. Applications in which the same BC can be applied across many related tasks are common in areas such as robot control [7, 8, 14], content generation [16, 18, 22], and classification [12, 26].

However, deriving an effective generalizable BC can be difficult. In most previous work, the experimenter provides a *hand-coded* BC specialized for the target task. Domain-agnostic *generic* BCs (e.g., applicable to any evolutionary robotics domain) have also been proposed [6, 7, 9, 20], but their performance varies by how well the assumptions underlying them hold in a particular task [23]. Interestingly, little work has focused on the problem of learning BCs [8, 18, 22]. This paper proposes a framework and methods for such learning. The goal is to perform well in a domain, i.e., to automatically discover which are the most important dimen-

GECCO '16, July 20 - 24, 2016, Denver, CO, USA

ISBN 978-1-4503-4206-3/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2908812.2908929

sions of behavioral diversity across its many tasks. Behavior samples produced by a default BC on a set of *training* tasks are used to learn a new BC, which then generalizes well across all tasks in the domain. This learned BC is validated on a set of *test* tasks drawn from the same domain.

Experiments combine these BC-learning techniques with novelty search [13, 14], and evaluate them across a range of increasingly-complex and deceptive many-task maze domains. The results show that the performance advantage of learned BCs over both hand-coded and generic BCs, as well as over fitness-based search, increases with the difficulty of the domain. Thus, this paper provides evidence that learning BCs is possible, useful, and aids in scaling novelty search to more complex real-world problems.

2. BACKGROUND

The approach in this paper builds on prior research in novelty search, generic BCs, and many-task domains.

2.1 Novelty Search

Novelty search [13, 14] is an EA which is not driven by optimization towards a fixed objective, but instead rewards individuals for demonstrating *behavioral novelty*. Such novelty is measured by comparing an individual's behavior both to other individuals in the current population and to those in an *archive* of past behaviors. The idea is that such novelty search, although less focused, may have a better chance of discovering complex and interesting behaviors, some of which may be solutions to the problem. Following previous precedent [14], in the implementation used here each evaluated individual is added to the archive with a fixed probability p. Additionally, to reduce computational cost and prevent oversaturation of regions in the behavior space, archives have a fixed maximum size. If the archive reaches its maximum size, new individuals added to the archive replace existing members with uniform probability, so that the archive does not contain only the most recently added behaviors.

Key to novelty search's performance in a given domain is the choice of BC. The BC is intimately connected to how novelty is quantified, which is the average distance of an individual's behavior to the k nearest neighbor behaviors of individuals in the population and archive. In most existing work, ad-hoc behavior characterizations are used, i.e., the BC is designed by the experimenter.

2.2 Generic Behavior Characterization

Generic BCs remove the need for expert domain knowledge in BC design. While their generality enables easy application to many domains, generality is also a liability: they may not take into account key aspects of particular tasks and domains. Several generic BCs have been introduced [7, 9, 20], with the conclusion that they were especially useful in multi-objective approaches that combine behavioral diversity with traditional objective-based fitness. One interpretation is that such a multi-objective approach does not require a nuanced sense of behavioral difference because its main aim is to encourage only enough diversity to prevent complete convergence. However, pure diversity-driven search may require more specialized ideas about behavior.

Most related to the approach proposed here, there is also some initial research into combining or learning BCs. Methods for combining multiple BCs were explored [6]; randomly switching between BCs improved performance over any single BC. In the approach most similar to the one in this paper, generic BCs were optimized through adapting a weight vector by calculating the mutual information between features and the given fitness function [8]. This weighting was updated periodically throughout evolution on a single task, not the many-task explored here; also, the approach relied on high-level task features that may not be available to an agent through its sensory-action space. The proposed method seeks to address some of these shortcomings, focusing on reducing the need for human domain knowledge or prespecified high-level features.

2.3 Generative Many-task Domains

The motivation for the many-task approach to evaluating evolutionary techniques is that tasks do not exist in a vacuum. Every task is drawn from some domain that includes several somehow-related tasks, and the experience acquired during evolution on one task can be leveraged to improve evolution's performance on future tasks. Illustrative examples of such domains include robots that learn to assemble many distinct products [17]; agents that learn to play different video games on the same platform [1]; and human-in-the-loop interactive evolution, in which solution quality depends on the particular human providing feedback [27].

Generative domains such as NK-landscapes are particularly useful because they can provide diverse challenges (and deception) across tasks [28]. In sequential decision-making, the many-task perspective has been suggested as a way to better validate approaches, preventing over-optimization of algorithms to a small set of tasks [2]. Generative domains have also been used for transfer in reinforcement learning [11]. For a given domain, the generative model that produces tasks may not be known explicitly. More realistically, one may only have access to a subset of tasks drawn from the model. In such cases, an approximate generative model can be inferred over which to optimize, or optimization can simply be performed over this fixed collection of tasks. As in the real world, one may not be able to choose which tasks are presented. However, for evaluating many-task approaches, explicit generative models are convenient. That way, a sufficiently large number of diverse tasks can be drawn for training and testing to clarify how the approach performs. The generative maze domains in this paper are based on those used to study the scalability of novelty search [14].

3. APPROACH

This section first introduces a many-task framework for learning BCs, and a new generic BC suited to this framework. Then, based on a formalization of BC optimality, two heuristics are introduced for learning BCs.

3.1 Many-task BC Learning Framework

Applying diversity-driven algorithms across a domain requires the specification of a BC, which affects the quality of search. This section describes a framework for learning effective BCs. The idea is that a set of training tasks can be leveraged to learn an effective BC, which can then be applied to any further tasks drawn from the domain. While such learning requires computation, the benefit is that a new BC must only be derived once per domain. In this way, if a large number of tasks are to be solved, the computational cost of learning becomes negligible in the limit.

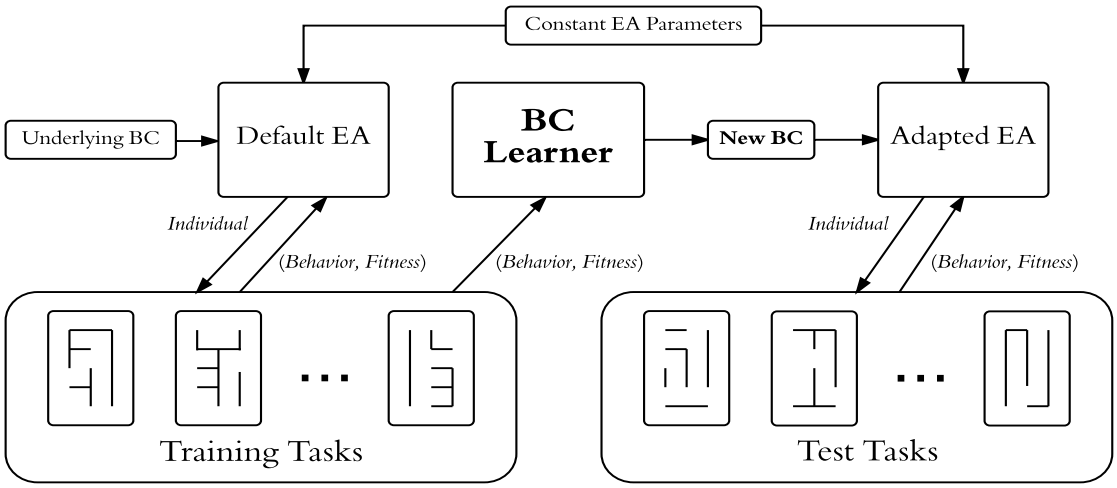

Figure 1: The batch version of the many-task framework for learning specialized BCs. A default EA is applied to the training tasks, which generates many $(Behavior, Fitness)$ pairs. A subset of such pairs is fed into the BC learner, which produces a BC specialized to the domain. This new BC is then evaluated on a set of test tasks, drawn either from the same (or a related) domain. Note that the BC learner observes only the results of the EA's interaction with the environment, i.e., it has no knowledge about the internals of the EA.

There are two ways to formulate BC learning: (1) an incremental version where the learned BC is updated after exposure to each task, and (2) a batch version, where the BC is learned only once based on a single set of training tasks. Only the batch version, which is the clearer setting for analysis, is investigated in this paper, and is depicted in Figure 1. In this framework there is a *default EA* and an *adapted EA*, which differ *only* in the BC that is used. The default EA is run on training tasks using some existing BC, called the *underlying BC*, and $(Behavior, Fitness)$ pairs from evaluated individuals are collected by the BC learner (in practice, only a subset of these pairs may be used). The BC learner then uses this information to produce a new BC. The new BC is used by the adapted EA, which is applied to the test tasks. Even if the training and test tasks are drawn from different distributions, the adapted EA with the new BC may still outperform the default EA if there is some structural relation. Potential measures of test set performance include averaging champion fitness across the test tasks, or the proportion of tasks on which a success threshold is achieved. Note that although novelty search is applied here as the underlying EA, this framework can be applied to any algorithm in which BCs guide search, e.g., MAP-elites [3, 22].

3.2 Stochastic Policy Induction

The BC learning framework described above requires an *underlying BC* that provides a set of base features that are consistent across tasks within a domain. This enables a weighting learned over the features of the underlying BC (see Section 3.4) to exploit this consistency. In theory, generic BCs are ideal candidates for underlying BCs, because by construction they require minimal domain-specific knowledge. However, most previously introduced generic BCs, like action history [9] and metrics over all task objects [8], result in features with different meanings when environmental components or evaluation lengths varies across tasks.

Stochastic Policy Induction for Relating Inter-task Trajectories (SPIRIT) addresses this shortcoming by providing an interpretable generic BC that is consistent across tasks within a domain. The main idea is that the BC produces the stochastic policy of a given form most likely to reproduce the state-action trajectory of an individual. The BC is a function only of an individual's sensory-action trajectory, so it does not rely on any privileged experimenter knowledge. In this paper, the simplest form of SPIRIT is used, termed SPIRIT$_o$. With SPIRIT$_o$, each element in a behavior vector represents the probability of the agent taking a particular action in a particular state. In higher-dimensional and continuous domains it may be helpful to use a more complex form of SPIRIT that can represent more expressive and compact policies, e.g., via decision trees, FSMs, or fuzzy systems. For these more complex representations, a suitable blueprint of a policy would be learned to maintain inter-task consistency. For SPIRIT$_o$, such structure learning is unnecessary.

Suppose $\{s_0, ..., s_{n-1}\}$ is the set of possible sensory states in the domain, $\{a_0, ..., a_{m-1}\}$ is the set of possible actions, and $\tau = ((s_0^\tau, a_0^\tau) = \tau_0, ..., \tau_t)$ is the agent's sensory-action trajectory recorded during evaluation. SPIRIT$_o$ induces a behavior characterization BC$_o$, such that BC$_o(\tau)$ produces a length nm vector describing a Markovian approximation of the agent's policy. The vector enumerates for each sensor-action pair (s_j, a_k) the proportion of times the agent took action a_k when in sensory-state s_j. More precisely,

$$\text{BC}_o(\tau)[jn+k] = \begin{cases} \frac{|\{i \,:\, \tau_i = (s_j, a_k)\}|}{|\bigcup_a \{i \,:\, \tau_i = (s_j, a)\}|} & \text{if } \exists\ a : (s_j, a) \in \tau, \\ \frac{1}{m} & \text{otherwise.} \end{cases}$$

Thus, if a possible sensory state is never encountered by an agent during its lifetime, actions from that state are assumed to have uniform probability. Euclidean distance is used to compare behaviors produced by BC$_o$. Note that SPIRIT$_o$ is similar to a previous generic BC method based on state counts [6], but with the advantage that it encodes an actual policy, providing an interpretable notion of what the agent does. This enables clearer comparison across tasks with different environmental components and evaluation lengths.

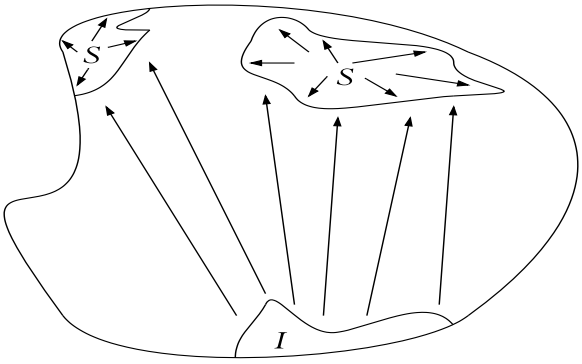

Figure 2: A desirable exploration pattern for a behavior-driven evolutionary process. Area I is the space of initial behaviors (i.e., those found in the initial population) achievable in the domain. Area S is the potentially disconnected space of behaviors that each yield success on at least one task in the domain. Informally, S-optimal behavior exploration may be achieved by discovering stepping stones from I to S and then exploring S as quickly as possible.

3.3 Optimal BCs and Exploring the Space of Success

This section proposes one possible formulation of BC optimality, motivating the heuristics proposed in Section 3.4 that intuitively target this definition.

The goal of BC learning is to find a BC that maximizes some notion of performance for a given EA in a given domain. In this paper, optimality is defined in terms of *success* in the many-task setting. An EA is deemed successful on a particular task if it evolves an individual meeting the domain-specific criteria for success, e.g., reaches a goal or crosses a specified fitness score threshold.

DEFINITION 3.1. *Let D be a domain and A an algorithm parameterized only by a behavior characterization B. B is* S-optimal *iff it maximizes the likelihood of A being successful on a task drawn randomly from the generative model of D.*

One way to view an S-optimal BC is one that enables an EA to most quickly cover a set S of behaviors, where for each task $t \in D$, there exists $s \in S$ such that s implies success in t with high likelihood. Suppose I is the set of behaviors of individuals from the initial population of A across all tasks in D for which A can achieve success. Then intuitively an S-optimal BC will guide the evolutionary process from I to S, and to explore S fully, as efficiently as possible (Figure 2). If S is relatively small compared to the entire space of underlying behaviors, a search for novelty that only explores S, and does so effectively, should be more efficient than search exploring the entire underlying behavior space indiscriminately. This observation leads to practical heuristics for learning BCs, which are discussed next.

3.4 BC Learning Heuristics

In this section two distinct learning heuristics are introduced, situated within the many-task BC learning framework. Each heuristic takes a different perspective on how successful individuals can be efficiently found for any task. The first, I2S, motivates evolution to progress from the space of initial behaviors I to the space of successful behaviors S, by focusing exploration within the dimensions of behaviors for which those in S vary most from those in I. The second, XS, motivates evolution to efficiently explore S itself, by focusing exploration within the dimensions of behavior varying most within S. Both I2S and XS are are implemented by applying a weighting to the features of the underlying BC.

Let $\mathcal{I}$ be a set of initial behaviors from successful runs, and $\mathcal{S}$ be a set of successful behaviors, where all such behaviors are collected during the training phase. Some underlying BC (see Section 3.1) produces these behaviors via a function $f : X \mapsto \mathbb{R}^l$, where each $x \in X$ contains the relevant information about an agent's interaction in its environment (e.g., for SPIRIT$_o$, X is the set of all possible sensory-action histories). The input to each BC learner is the pair $(\mathcal{I}, \mathcal{S})$; the output is a weight vector $w \in \mathbb{R}^l$, which states the learned behavioral importance of each feature. The vector w induces a new BC computed as $w \odot f(x)$, where $\odot$ is the Hadamard product, with resulting behaviors compared via Euclidean distance. Note that these methods assume the features of the underlying BC are normalized.

Let $\overline{i}$ be the mean over all $i \in \mathcal{I}$, and $\overline{s}$ be the mean over all $s \in \mathcal{S}$. I2S emphasizes features that differ most between successful behaviors and initial behaviors; the idea is that rewarding exploration along those features will generate stepping stones that lead from I to S. So, for I2S, w is computed by

$$w[k] = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} |s[k] - \overline{i}[k]|^{\lambda},$$

where λ controls how much stronger features are strengthened over weaker ones. $\lambda = 0$ implies a uniform w, i.e., the underlying BC is completely preserved. Only initial behaviors i from successful runs are included in $\mathcal{I}$, because this implies S is reachable from i. On the other hand, XS emphasizes the features that vary most *within* the space of success, to facilitate efficient exploration of S. So, for XS, replacing $\overline{i}$ with $\overline{s}$, w is computed by

$$w[k] = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} |s[k] - \overline{s}[k]|^{\lambda}.$$

When $\lambda = 2$, XS yields the variance of each feature within $\mathcal{S}$. XS characterizes S more accurately than I2S, so it should enable more efficient exploration of S when individuals with behaviors in S can be sampled. However, this advantage may be diminished when it is difficult to sample from S directly, e.g., when I and S are non-overlapping. Using SPIRIT$_o$ as the underlying BC (see Section 3.2), the complete learning pipelines SPIRIT$_{\text{I2S}}$ and SPIRIT$_{\text{XS}}$ are evaluated empirically in the next section.

4. EXPERIMENTS

The learning system described in Section 3 is applied to a range of generative versions of the canonical deceptive maze domain [13, 14, 15], to determine which of the heuristics are most effective, and on what types of problems.

4.1 Generative Maze Domains

Variants of maze domains have been the canonical testbed for novelty search since its inception [13, 15, 20, 23]. For this reason, a variety of generative maze domains, similar to those used by Lehman and Stanley [15], are explored here. In

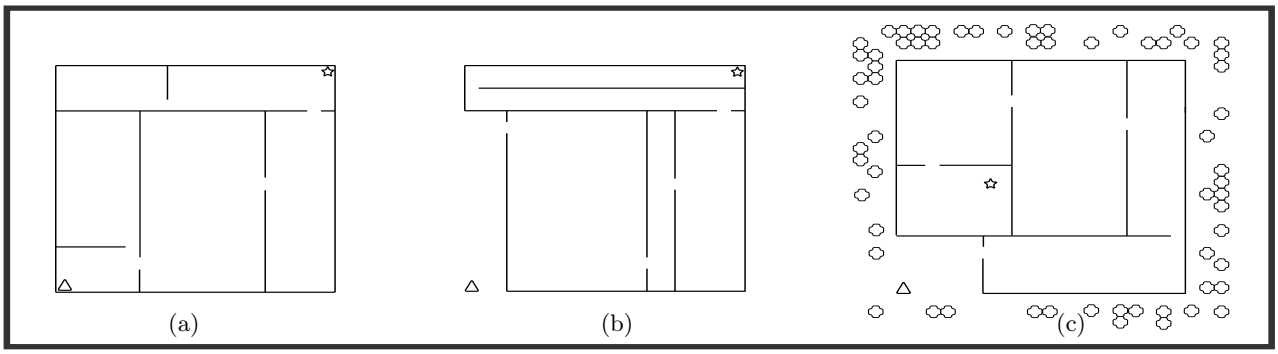

Figure 3: Examples of randomly generated mazes that illustrate the challenges that can arise in each domain: (a) Simple maze with complexity 38; (b) Complex Open maze with complexity 97; (c) Simple Open Forest maze with complexity 64 and $p_{\text{tree}} = 0.25$, with nearest trees shown. The triangle in the bottom left of each maze shows the agent's starting position and orientation, and the star indicates the location of the goal. Each maze instantiates several local optima for objective-based search, but the higher-complexity Open mazes are more challenging, with several of their local optima residing outside of the maze. The Open Forest maze can also hinder generic approaches to novelty search, because it includes diverse distracting stimuli.

the basic setting, the agent has ten binary sensors: six local wall sensors and four pie slice goal sensors. A goal sensor fires when the goal falls within its 90° cone; each goal sensor is centered in a different cardinal direction. The agent has three possible actions: turn 90° left, turn 90° right, and move forward one unit (look ahead to Figure 5 for sample sensor-action pairs). The agent acts until either it occupies the same cell as the goal (which counts as success in this domain) or the maximum number of steps (200) is exhausted.

Each maze domain's generative model produces mazes by recursive division [24]. In this paper, all mazes are 20 cells × 20 cells with five internal walls. Each model is parameterized based on the environmental features that its mazes require. One feature is complexity, which is defined as the length of a shortest path from the agent starting position to the goal, and is correlated with deception [15]. If a generative model has a minimum complexity requirement, then mazes produced with lower complexity are discarded. A model produces *Simple* mazes if there is no such requirement; it produces *Complex* mazes when the minimum complexity is 80. A second feature is whether or not mazes are open. When a model produces *Open* mazes, the outer walls of the lower left chamber are removed. Open mazes create additional difficulties, as the agent is free to explore the infinite plane, which can complicate search with a naive BC [14].

As a final feature, the maze model is extended to support *Forest* mazes. For a Forest maze, every cell distance at least 1 unit away from the 20 × 20 cell box in which the maze is situated has probability p_{tree} of being occupied by a tree. An agent in a Forest maze is augmented with six sensors equivalent to the local wall sensors, except they detect trees instead. Because wall and forest sensors are distinct, an agent can discriminate between the forest and the maze, which aids in navigation. As with walls, the agent cannot occupy the same cell as a tree, i.e., executing *forward* does nothing when a tree is in front of the agent. Similar to a previous generative domain [11] used in reinforcement learning, the Open Forest maze setting is challenging for novelty search, and for generic BCs in particular, because it adds distracting sensory and spacial exploration possibilities which are unrelated to solving the task. Figure 3 shows three sample generated mazes. All agents begin in the lower left corner facing North. For Forest mazes, the goal is randomly placed within the maze; for all other mazes, it is always located in the upper right corner. In the experiments, $p_{\text{tree}} = 0.25$.

4.2 Experimental Setup

The underlying EA for novelty and objective-based search in the experiments is the Direct Encoding of Neural Networks (DNN) neuroevolution algorithm [19, 20, 22], which generates neural network agent controllers, and has been previously applied to work exploring BCs. DNN is a simplified version of the popular NEAT algorithm [25]. The implementation in this paper is based on that specified by Mouret and Doncieux [19] (See the Appendix for settings).

In each BC learning setup, training data was first generated from a *source* domain and used to learn BCs that were evaluated in a *target* domain. BCs were tested in six target domains from Section 4.1: Simple, Complex, Simple Open, Complex Open, Simple Open Forest, and Complex Open Forest. For all target domains without trees, the source domain was Complex. For Forest target domains, two source domains were tried: Simple Forest to test the ability to ignore novel sensor readings, and Simple Open Forest. Thus, the Forest experiments test the approaches' ability to produce BCs that generalize to domains with higher complexity.

Novelty search was run with SPIRIT$_o$ as its BC on tasks drawn from source domains until a substantial amount of training data was collected. If success was achieved within the limit of 10,000 evaluations, then the behavior of the first individual evaluated was added to the set of initial behaviors $\mathcal{I}$, and the behavior of the successful individual was added to the set of successful behaviors $\mathcal{S}$. Then, from the pair $(\mathcal{I}, \mathcal{S})$, SPIRIT$_{\text{l2s}}$ and SPIRIT$_{\text{xs}}$ each output a new BC, which was evaluated in target domains to test performance and generalizability. 20,649 initial and successful behaviors were collected from the Complex source domain; 3,836 from Simple Forest; and 2,211 from Simple Open Forest.

SPIRIT$_{\text{l2s}}$ and SPIRIT$_{\text{xs}}$ setups were compared against several baselines: a traditional objective-based search with no novelty component (as in most EA's); novelty search with the hand-coded BC (final (x, y) location, with Euclidean dis-

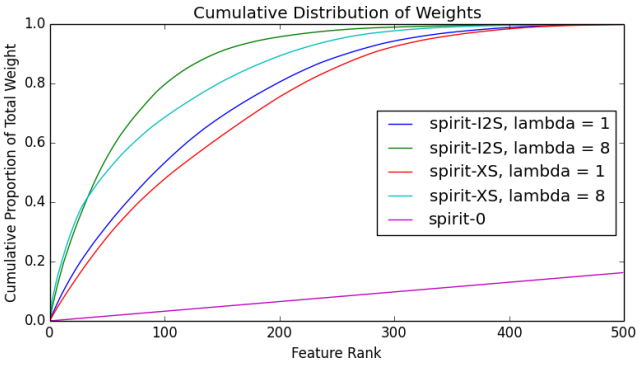

Figure 4: Cumulative proportion of weight by feature rank for the top 500 features of weight vectors produced by spirit$_{i2s}$ and spirit$_{xs}$ in the experiments from the Complex domain, for $\lambda \in \{1, 8\}$. In contrast to the uniform weights of spirit$_o$, in the learned weight vectors most of the weight is concentrated in only the top couple hundred features.

tance metric); novelty search with action history BC (with a Hamming distance metric), which has been found to be a promising generic BC [9]; and SPIRIT$_o$. Each setup was evaluated on 5,000 test tasks for each target domain (except SPIRIT$_o$ in Complex Open Forest, which was only evaluated on 959 test tasks due to computational time constraints).

4.3 Results and Analysis

The distributions of weights in the weight vectors learned by SPIRIT$_{I2S}$ and SPIRIT$_{XS}$ via samples from the Complex domain are depicted in Figure 4. The BC learning techniques concentrate the mass of the weight vector; that is, relatively few features take up most of the weight, especially when λ is high. In all performance tests $\lambda = 8$, and preliminary tests show robustness to changes to $\lambda \in [1, 10]$ for SPIRIT$_{I2S}$.

It is also informative to look at the features deemed most and least important by BC learners to better understand what these heuristics are accentuating. The strongest and weakest features for SPIRIT$_{I2S}$ from Complex mazes are depicted in Figure 5. These features generally match the intuition for what should be learned. In particular, the majority of the strongest features are sensor-action pairs that demonstrate overcoming deception, i.e., they go against the greedy strategy. Thus, exploring these features may encourage solving more complex mazes. On the other hand, the majority of the weakest features need not be explored. All the weakest features, except the eighth, depict sensor-action pairs that cannot occur in a shortest solving trajectory, i.e., their exploration is intuitively inefficient. In this way, these features coincide with the intuition of an effective BC learner.

Quantitatively, BC learning improves performance in all six domains, with SPIRIT$_{I2S}$ always outperforming SPIRIT$_{XS}$. Figure 6 compares the success rate of SPIRIT$_{I2S}$ and SPIRIT$_{XS}$ with the baseline methods across the six domains. The most dramatic improvements are in the Forest domains, though SPIRIT$_{I2S}$ outperforms all other methods in all cases. In target domains without trees, the strongest improvement for SPIRIT$_{I2S}$ is in the most difficult domain: Complex Open. This result further validates the idea that BC learning will be most useful in complex scenarios. It is also interesting that in the domains without trees, SPIRIT$_o$ significantly outperforms all other baselines, highlighting its usefulness as a generic BC. On the other hand, in the Forest domains, SPIRIT$_o$ is not so successful; it performs more similarly to the other baselines than to the top BC learning approaches.

As expected, in both Forest domains, each BC learning approach performs significantly better when trained with closed maze samples. With a closed maze, there is no observation of trees, so trees are completely ignored in the test tasks, making the new BC much more efficient than the underlying BC. In Open Forest training, trees may be encountered, but successful behaviors should contain few tree observations, resulting in a similar but somewhat attenuated effect. In Forest domains, BC learning successfully captures the idea that additional distractor sensors should not make any task more difficult. This idea of including sensors orthogonal to a task often arises in robot learning, as many sensors may have utility in some domains but not others.

5. DISCUSSION AND FUTURE WORK

The BC learning techniques in this paper used samples only of initial and successful behaviors to create new BCs; however, intermediate behaviors could also be leveraged for learning *dynamic BCs* [6] that change during evolution. For example, ancestral histories of successful individuals could be analyzed to derive temporal sequences of BCs more effective at guiding search into and through the space of successful behaviors. In the static setting I2S dominated XS, but such dynamic BCs could exploit synergies between I2S and XS, e.g., to initially explore using I2S and then to switch later in evolution to XS. This idea reflects the notion that I2S and XS capture complementary aspects of behavior space exploration. Another direction is to learn more objective-oriented *fitness features*, like those in shaping approaches [4, 5, 11], to complement the learned BC with optimization pressure. Shaping approaches have also shown how training on tasks that are easier than the target task may improve efficiency. The fact that Simple Forest training improved Complex Forest performance is encouraging in that training data did not have to be collected from the more difficult domain.

One critical direction for future work is extending BC learning to continuous and high-dimensional domains. A liability of the SPIRIT$_o$ representation is that its size grows exponentially in the number of possible state-action pairs. For this reason, more sophisticated representations, such as those used in reinforcement learning [10], are necessary for higher-dimensional contexts. To keep the dimensionality of SPIRIT low while still retaining enough relevant behavioral information, more compact policy representations such as decision trees or fuzzy systems can be used. Complementarily, SPIRIT could be combined with recent advances in deep learning to enable learning higher-level behavioral features. One approach is to train an autoencoder (or other unsupervised learning method) from SPIRIT representations of behaviors. The idea is that bottleneck layers of the autoencoder might encode a compressed space of relevant behaviors.

An advantage of the many-task framework is that it enables skirting the challenges of particularly difficult training tasks, because learning features from tasks more easily solved with the default BC may later generalize to enable solving more difficult tasks in the test set. In contrast, in the single-task case the default BC may be so ill-suited that solutions are never evolved, preventing at least the BC learners

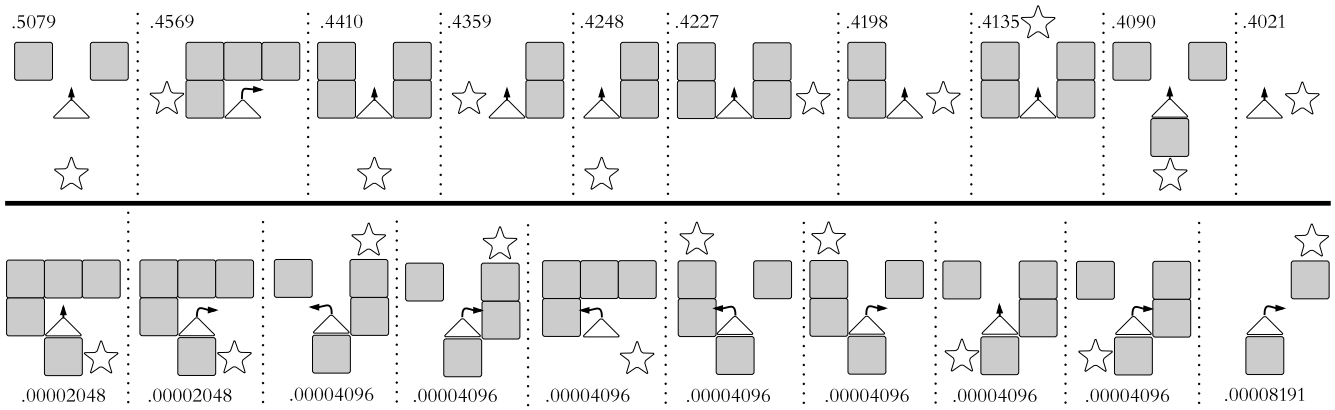

Figure 5: The strongest and the weakest features in the weight vector learned by spirit$_{i2s}$ via samples from the Complex source domain, along with corresponding weights ($\lambda = 1$). Grey squares indicate active local wall sensors; the star indicates directions of active pie slice goal sensors; the arrow on the tip of the agent indicates the action taken. (top row) Top ten strongest features. Each of these features, except the sixth and eighth, show a sensor-action pair whose exploration corresponds to solving deceptive tasks. (bottom row) Top ten weakest non-zero features. Except the eighth, each of these features show a state-action pair that cannot occur in a shortest solving trajectory, i.e., their exploration is intuitively inefficient. These results coincide with expectations of what kinds of features a BC learner should be emphasizing and deemphasizing.

in this paper from being effective. A deep learning approach more suited to the single-task case would be to, during evolution, train an explicit model of the fitness landscape from SPIRIT representations; the high-level features encoded by the layer preceding the regression output could also encode a reduced but relevant space of behaviors.

6. CONCLUSION

Motivated by reducing how much human knowledge is needed to apply diversity-driven algorithms to new problems, this paper introduces a general framework for learning BCs automatically. The experiments combine two approaches for BC learning with the novelty search algorithm, which enables applying novelty search more effectively in a general setting. The results show that such BC learning can outperform hand-designed BCs and completely generic BCs, especially when tasks become increasingly complex. In this way, this paper demonstrates the potential for BC learning to be an ingredient in effective automatic problem solving.

Aknowledgments. We would like to thank Alexander Braylan and the reviewers for their useful feedback. This research was supported in part by NSF grant DBI-0939454, NIH grant 1R01GM105042, and an NPSC fellowship sponsored by NSA.

7. REFERENCES

[1] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *JAIR*, 47:253–279, 2013.

[2] O. Coleman, A. Blair, and J. Clune. Automated generation of environments to test the general learning capabilities of ai agents. In *Proc. of GECCO*, pages 161–168, 2014.

[3] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

[4] S. Doncieux. Transfer learning for direct policy search: A reward shaping approach. In *Proc. of ICDL*, pages 1–6, 2013.

[5] S. Doncieux. Knowledge extraction from learning traces in continuous domains. In *AAAI Fall Symposium Series*, 2014.

[6] S. Doncieux and J.-B. Mouret. Behavioral diversity with multiple behavioral distances. In *Proc. of CEC*, pages 1427–1434, 2013.

[7] J. Gomes and A. L. Christensen. Generic behaviour similarity measures for evolutionary swarm robotics. In *Proc. of GECCO*, pages 199–120, 2009.

[8] J. Gomes, P. Mariano, and A. L. Christensen. Systematic derivation of behaviour characterisations in evolutionary robotics. *CoRR*, abs/1407.0577, 2014.

[9] F. J. Gomez. Sustaining diversity using behavioral information distance. In *Proc. of GECCO*, pages 113–120, 2009.

[10] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[11] G. Konidaris, I. Scheidwasser, and A. G. Barto. Transfer in reinforcement learning via shared features. *JMLR*, 13(1):1333–1371, 2012.

[12] E. Laredo, L. Trujillo, and Y. Martinez. Searching for novel classifiers. In *Proc. of Evostar*, pages 145–156, 2013.

[13] J. Lehman and K. O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *Proc. of ALIFE*, pages 329–336, 2008.

[14] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011.

[15] J. Lehman and K. O. Stanley. Novelty search and the problem with objectives. *GP Theory and Practice IX*, pages 37–56, 2011.

[16] J. Lehman and K. O. Stanley. Beyond open-endedness: Quantifying impressiveness. In *Proc. of ALIFE*, pages 75–82, 2012.

[17] G. Levitin, J. Rubinovitz, and B. Schnits. A genetic algorithm for robotic assembly line balancing. *Euro. Journ. of O. R.*, 168(3):811–825, 2006.

[18] A. Liapis, H. P. Martinez, J. Togelius, and G. N. Tannakakis. Transforming exploratory creativity with delenox. In *Proc. of ICCC*, pages 56–63, 2013.

[19] J.-B. Mouret and S. Doncieux. Using behavioral

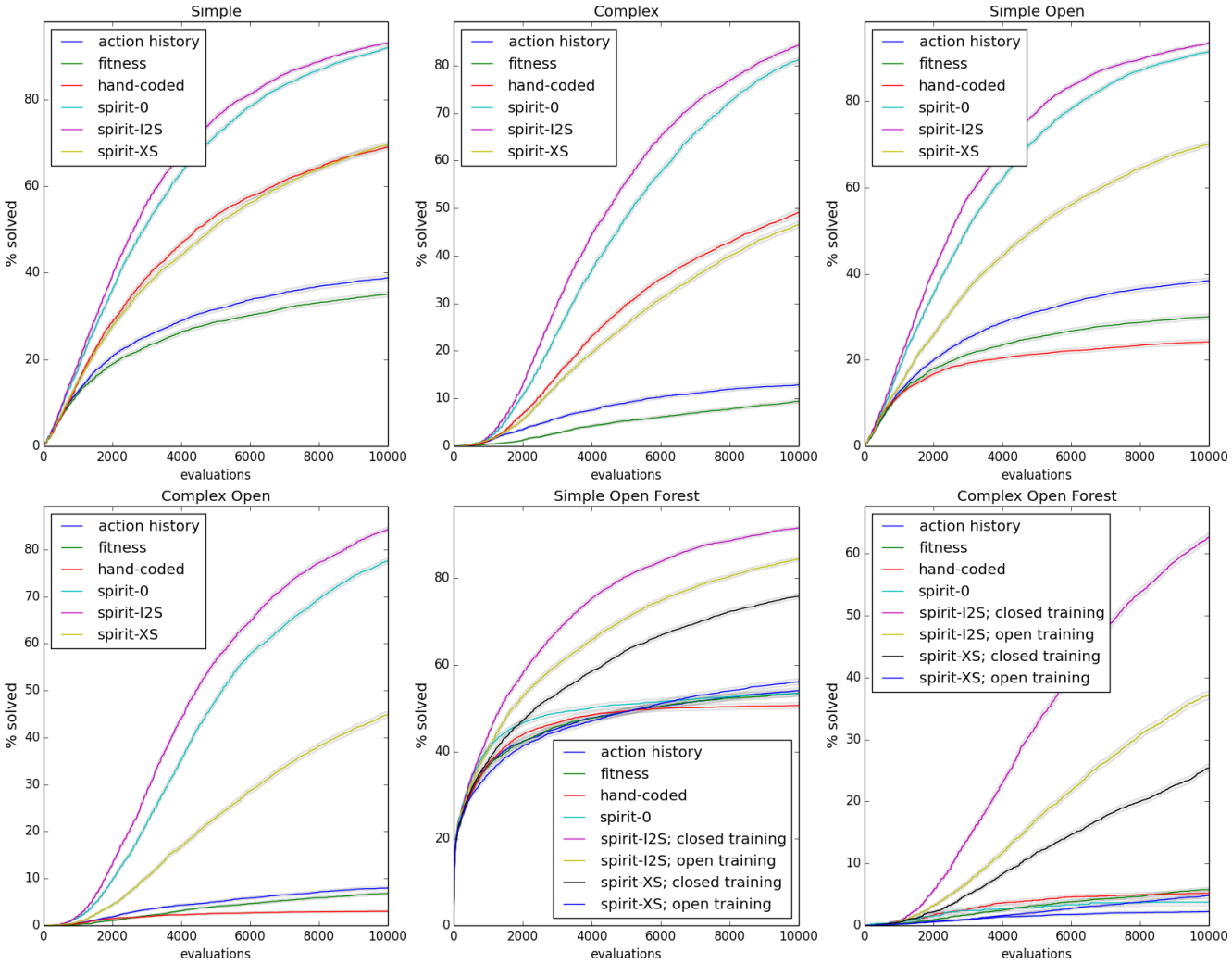

Figure 6: Learning curves comparing learned BCs to baselines in six models of the maze domain. Each line indicates percent solved of 5,000 test tasks (except only 959 for spirit$_o$ in Complex Open Forest due to time constraints), and the shaded area around them indicates standard error (very small due to large sample size). The spirit$_{i2s}$ method outperforms all other methods in each domain; its advantage scales with the difficulty of the domain. The most striking improvement is in Complex Open Forest, suggesting that BC learning will be most useful in more complex domains.

exploration objectives to solve deceptive problems in neuro-evolution. In *Proc. of GECCO*, pages 627–634, 2009.

[20] J.-B. Mouret and S. Doncieux. Encouraging behavioral diversity in evolutionary robotics: An empirical study. *Evolutionary Computation*, 20(1):91–133, 2012.

[21] A. L. Nelson, G. J. Barlow, and L. Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345–370, 2009.

[22] A. Nguyen and J. Clune. Innovation engines: Automated creativity and improved stochastic optimization via deep learning. In *Proc. of GECCO*, pages 959–966, 2015.

[23] J. K. Pugh, L. B. Soros, P. A. Szerlip, and K. O. Stanley. Confronting the challenge of quality diversity. In *Proc. of GECCO*, pages 967–974, 2015.

[24] A. M. Reynolds. Maze-solving by chemotaxis. *Physical Review E*, 81(6), 2010.

[25] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[26] P. A. Szerlip, G. Morse, J. K. Pugh, and K. O. Stanley. Unsupervised feature learning through divergent discriminative feature accumulation. In *Proc. of AAAI*, pages 2979–2985, 2015.

[27] H. Takagi. Interactive evolutionary computation: Fusion of the capabilities of ec optimization and human evaluation. *Proc. of the IEEE*, 89(9):1275–1296, 2001.

[28] D. Whitley. Mk landscapes, nk landscapes and max-ksat: A proof that the only challenging problems are deceptive. In *Proc. of GECCO*, pages 927–934, 2015.

APPENDIX

DNN Parameters. pop. size, 100; parents per gen.: 25; children per parent: 4; max evaluations: 10,000; add connection rate: 0.15; remove connection rate: 0.05; add node rate: 0.05; remove node rate: 0.05; min/max connection weight: -5.0/5.0; connection mutation rate: 0.1; connection mutation change range: [-1,1] uniform. **Novelty Search Parameters**. prob. add individual to archive: 0.01; archive size: 1,000; archive replace function: random; k: 15.