How Should Intelligence Be Abstracted in AI Research ...
AAAI Technical Report FS-13-02

Guiding Evolutionary Learning by Searching for Regularities in Behavioral Trajectories: A Case for Representation Agnosticism

Krzysztof Krawiec
Institute of Computing Science
Poznan University of Technology
Piotrowo 2, 60-965 Poznan, Poland

Jerry Swan
Computing Science and Mathematics
University of Stirling
Stirling FK9 4LA, Scotland

Abstract

An intelligent agent can display behavior that is not directly related to the task it learns. Depending on the adopted AI framework and task formulation, such behavior is sometimes attributed to environment exploration, or ignored as irrelevant, or even penalized as undesired. We postulate here that virtually every interaction of an agent with its learning environment can result in outcomes that carry information which can be potentially exploited to solve the task. To support this claim, we present Pattern Guided Evolutionary Algorithm (PANGEA), an extension of genetic programming (GP), a genre of evolutionary computation that aims at synthesizing programs that display the desired input-output behavior. PANGEA uses machine learning to search for regularities in intermediate outcomes of program execution (which are ignored in standard GP), more specifically for relationships between these outcomes and the desired program output. The information elicited in this way is used to guide the evolutionary learning process by appropriately adjusting program fitness. An experiment conducted on a suite of benchmarks demonstrates that this architecture makes agent learning more effective than in conventional GP. In the paper, we discuss the possible generalizations and extensions of this architecture and its relationships with other contemporary paradigms like novelty search and deep learning. In conclusion, we extrapolate PANGEA to postulate a dynamic and behavioral learning framework for intelligent agents.

1 Introduction

In AI, a rational agent is expected to choose actions that maximize the likelihood of reaching the goal posed by the considered task. If the task is nontrivial, attaining the goal may require a long sequence of actions, with some of them providing little or no useful feedback from the environment. In such cases, the agent is obliged to perform a systematic (or more or less random) exploration of the environment.

The importance of both exploration and exploitation is widely admitted, and, as put by John Holland, agent learning can bee seen as a tension between these two (citing after (Mitchell 1998)). However, it is often assumed that the time spent on exploration is the price an agent has to pay for reaching the 'useful' parts of the environment (or search space in general), from which a more directed exploitation can start. Though rarely expressed so explicitly, this may suggest that exploration is an inevitable toll that we would be happy to do without. Contrary to this opinion, we will argue here that potentially useful knowledge can be gathered from almost any interaction with the environment. The main hypothesis of this paper is that the *dynamics* of agent interaction with the environment is a useful source of information that can be harnessed. More specifically, an agent that explores its environment exhibits behavior that can be *related* to the considered task (more precisely, to the performance measure used to evaluate agent's behavior). If there are means by which such relatedness could be assessed, they can be used to help the agent perform well at the task.

Consider the task of designing a mobile robot that should learn how to use its video sensor to locate a power socket (to approach it and charge itself from it). Assume robot A makes little use of the visual stimuli but intensely explores the environment, which at times causes it to attain the goal, i.e., to reach the socket. Robot B, on the other hand, clearly reacts to visual input (e.g., approaches or avoids brightly lit objects). However, because of this 'directional' behavior it gets 'distracted', explores less, and does not succeed in finding the socket in the prescribed time.

With respect to an objective performance measure (e.g., the success rate in multiple trials), robot A will be judged superior to robot B. But for a human observer, robot B clearly exhibits a more intelligent behavior. Moreover, that behavior has the potential of being malleable towards performing well at the task, in which case robot B would almost certainly beat robot A in performance (by being able to find a socket in more trials or in a shorter time within a single trial).

This thought experiment leads to two nontrivial observations. Firstly, adopting the external perspective on agent's behavior can provide insights that may be otherwise hard to obtain. For instance, if the robots have no location sensors, robot B may be unaware that the effects of its actions (movements) correlate with the presence of (or proximity to) visual stimuli, even if this is plainly obvious for an external observer. The second observation is that, to assess agent's intelligence (which may impact its prospective performance), one needs to scrutinize *all* aspects of behavior, including the aspects that seem directly unrelated to the assumed perfor-

mance measure.

Following these observations, we advocate here AI systems designed with such external perspective in mind and capable of discovering relationships between agent behavior and the goal of the task. Past AI research investigated various tools and frameworks for capturing such relationships. Hofstadter's seminal work on creative analogies (Hofstadter 1984) and the follow-up studies (Mitchell 1997; Marshall 2006) are probably the most famous examples of such attempts. In the next section, we propose an approach which demonstrates that straightforward machine learning (ML) techniques can be harnessed for an analogous purpose in the context of evolutionary learning. Based on that example, we elaborate on the variants and possible implications of this perspective.

2 Pattern guided genetic programming

Pattern-guided Genetic Programming (PANGEA) is an approach we proposed in (Krawiec and Swan 2013) for evolutionary search in spaces of computer programs. PANGEA is an extension of genetic programming (GP), a genre of evolutionary computation tailored to solve programming tasks (Koza 1992). The objective in an programming task is to synthesize a computer program that conforms certain desired behavior. The desired behavior is typically specified by providing a set of examples, each composed of program input and the corresponding desired output. In this respect, a programming task is thus identical with supervised machine learning from examples. Crucially, the quality (fitness) of a program depends exclusively on the *final* effect of its execution, i.e., its output.

The key rationale for PANGEA is that programs which yield output that does not match the desired output may produce some *intermediate outcomes* that can be useful for solving the task. In standard GP, intermediate results are not taken into account, because there is no means for assessing their relatedness to the considered task. As a result, a program that calculates a useful intermediate result but ends execution with a final outcome that is unrelated to the considered task, receives low fitness and may not pass the selection process. The potentially valuable pieces of program code (e.g., subexpressions, subprograms) are thus lost.

However, since the partial results have ultimately been calculated from input data that came with the task and using an instruction set that also belongs to task formulation, they can represent useful pieces of knowledge of potential value in assembling a complete solution. In particular, the outcomes calculated for multiple inputs (examples) can form some *patterns*, especially when collated with the desired outputs. A skilled human programmer or mathematician can discover such patters and exploit them to reach the goal, i.e. to design a program that meets the requirements of programming task. Moreover, humans are capable of defining what patterns are *desired* as an intermediate result. Consider designing a program that calculates the median of a vector of numbers. A reasonable first stage of solving this task is sorting the elements of the input vector. Therefore, an intermediate memory state (pattern) containing sorted elements of the input vector is desired for this task (and anticipated by a human programmer).

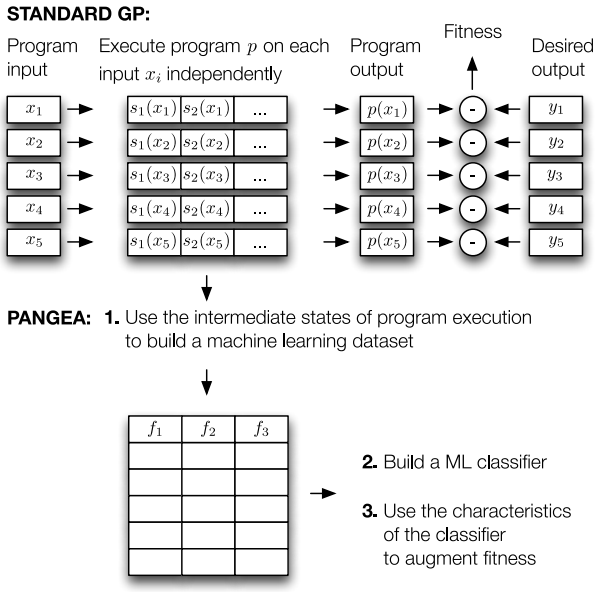

Figure 2.1: The top part illustrates the conventional GP fitness assessment. The program p being evaluated is applied independently to the input part x_i of every example, and the output $p(x_i)$ produced by p for that input is compared to the desired output y_i. The fitness assigned to p is the divergence of actual and desired output aggregated over all training examples. The bottom part shows the components added by PANGEA. The intermediate states $s_j(x_i)$ traversed by an executing program give rise to syntactic and semantic features (f_k) that form columns of a conventional machine learning dataset. The rows of the dataset correspond to the examples. A machine learning classifier is induced from the dataset and the properties of that classifier are used to adjust p's fitness.

PANGEA uses a machine learning algorithm to search for such patterns by inducing a classifier from the partial outcomes of program execution (Fig. 2.1). Information on the resulting trained classifier (in particular its complexity and accuracy) is then used to augment the fitness function. If this approach is able to reveal meaningful dependencies between partial outcomes and the desired output, we may hope to thereby promote programs with the potential to produce good results in future, even if at the moment of evaluation the output they produce is incorrect.

2.1 Technical description

PANGEA differs from the conventional GP only in the manner in which fitness is assigned to individuals in population (programs); apart from that, the method follows the conventional cycle of evaluation, selection, and breeding of individuals, as in most genres of evolutionary computation, including GP (Poli, Langdon, and McPhee 2008).

In GP, a program is evaluated by applying it individually

to all training examples and measuring how its output (i.e., the *final* effect of program execution) diverges from the desired output. The fitness of the program is typically a simple aggregate of these divergences (see the top part of Fig. 2.1). PANGEA additionally adjusts program's fitness by gathering and exploiting the information resulting from the intermediate stages of program execution. Prior to program execution, the interpreter is in an initial state, and execution of consecutive program instructions moves it to different states (depending on programming paradigm, a state may comprise registers, memory, working stacks, etc.). The sequence of states traversed by an executing program for a given input forms its *trace*. The last state in a trace, i.e., the state of execution environment after program completion[1], determines program output.

The traces capture program behavior for all training examples. PANGEA searches in the traces for patterns/regularities that reveal the generating program's hidden qualities by employing conventional machine learning algorithms that implement the paradigm of learning from examples and attribute-value representation (Mitchell 1997). To this end, we transform the traces into a machine learning dataset, where each row (example) corresponds to GP example, and every column is a feature derived in certain way from particular states of the traces. A feature reflects syntactic or semantic information derived from program traces at certain stage of program execution, seen 'across' the training examples.

As the objective of PANGEA is to assess how program behavior (captured in the features) relates to the desired output of the program, the dataset is extended with an extra column that defines the decision class attribute. The value of the class attribute is based on the desired output of the corresponding example (y_i in Fig. 2.1). The class attribute allows applying a supervised learning algorithm to the dataset and thus detecting and capturing regularities that relate to the desired output, and thus are relevant for solving the programming task.

Given the complete dataset with the class attribute, PANGEA induces from it a machine learning classifier (a decision tree in (Krawiec and Swan 2013)) and examines it with respect to two characteristics. Firstly, the classifier has certain inherent *complexity*, which in case of symbolic representation can be expressed as the number of symbols (tree size). Secondly, it commits a *classification error* on the training set, i.e. it erroneously classifies some of the examples. According to the *Minimum Description Length* principle (Rissanen 1978), these two characteristics, i.e., the size of the 'rule' and the number of exceptions from the rule, together determine the complexity of the mapping from the space of features onto the decision attribute. In our context, they tells us how easy it is to produce the desired output given the partial results gathered from program trace. If the features collected from program trace form meaningful patterns, i.e. capture regularities that are relevant to desired output, then the induction algorithm should be able to build a compact tree that commits few classification errors (and thus has short description length). Based on this premise, we extend the fitness formula with additional terms that depend on classifier complexity and classification error, so that the programs that lead to simple classifiers or classifiers that commit few errors are promoted in evolutionary search process. For details on this formula and other components of the methods, see (Krawiec and Swan 2013).

The above process of acquisition of program traces, transforming them into a supervised learning dataset, inducing a classifier from the dataset, and using the complexity and the error of the resulting classifier to adjust program fitness, are carried out for each evaluated individual independently. The role of the classifier is to assess program's prospective capability of solving the problem (i.e. producing the desired output value), however it does not form a permanent part of individual/solution and is discarded after fitness is assigned to a program.

2.2 Experimental verification

In (Krawiec and Swan 2013), we experimentally compared PANGEA to regular GP within the framework of stack-based GP variant known as Push (Spector and Robinson 2002). The comparison involved 12 instances of six programming tasks. Each task consisted in producing a correct output (a Boolean or integer value, depending on the task) for a given input array of numbers. In particular Boolean tasks, the evolved programs were to verify whether the array elements were all equal, whether the array was sorted in an ascending order, whether more than half of array elements were ones, and whether the array contained a copy of the first element. In the two integer tasks, a program was to calculate the maximum of the array, and count the number of zeroes in the array.

On most benchmarks, PANGEA attained much higher success rates, i.e., the fraction of runs that ended with success, often several times greater than standard Push; on the remaining benchmarks it was not worse than regular GP. We also verified that both classifier complexity and classifier error were essential for the method to perform well. The differences between PANGEA and the reference methods were statistically significant. Also, a detailed analysis of selected runs proved that the method operates as expected: programs that yield incorrect output but produce intermediate results that relate (as judged by the classifier) to the desired output often pass selection and thus contribute to the subsequent generations of individuals.

2.3 Implications and possible extensions

The experimental results demonstrate that the dynamics of agent's behavior, captured in features reflecting the intermediate states traversed by agent program, is a useful source of information that can be exploited to improve agent's performance by guiding its learning process. This is remarkable, knowing that in evolutionary learning, agent behavior is in great part random, particularly in the initial generations. Yet by linking that behavior to agent's goal using machine learning tools, useful search gradients can be elicited.

In PANGEA, the regularities detected in agent behaviors are used to aid an evolutionary learning process. They im-

[1] We consider only halting programs, so traces are finite.

pact individual's fitness and thus indirectly influence the likelihood of its behavioral traits being propagated to subsequent generations. Currently, the feedback received by agents-programs is very simple: they are being rewarded or punished by fitness adjustments. Relying on such low-information, 'poor' training signal is typical for solving learning and optimization tasks using traditional evolutionary algorithms. However, by analyzing in detail the induced classifier, one might determine not only how useful a program trace as is a whole, but also which particular trace elements (and thus the corresponding parts of program) relate to the desired output. This information could provide a richer feedback for an agent, which could react to it by applying more directional changes (e.g., modifying specific program instructions). Such extensions of PANGEA would probably not fit anymore within evolutionary computation, and other conceptual frameworks could be more appropriate.

In PANGEA, agent dynamics are captured and analyzed locally, i.e., for each agent-program independently. We expect that a comparative analysis of multiple agents from population could bring similar, if not greater, benefits. This is because certain behavioral patterns may not be prominent enough when expressed in traces of a single individual, or become evident only when confronted with behaviors of other agents.

3 Related concepts and framings

PANGEA can be linked to other concepts and approaches proposed in the contemporary AI and computational intelligence research. In not necessarily pursuing the search direction imposed by fitness function, but possibly detouring via other parts of the space of programs in search of interesting trace features, PANGEA pertains to novelty search (Lehman and Stanley 2011) and other approaches that use alternative drivers to propel search dynamics, e.g., (Schmidhuber 1991). In many cases, agents enticed to behave originally compared to their competitors proved to be surprisingly effective at solving the posed task (e.g., maze navigation in (Lehman and Stanley 2011)). The success of this form of open-endedness is due to, among others, the many-to-one mapping from the space of agent functions (genotypes in evolutionary terms) to the space of agent behaviors (phenotypes), which, when combined with novelty search, entices originality (while an ordinary, single-objective search process may more easily get stuck in a local optimum).

Recent advances in 'deep learning' (Ranzato et al. 2011) have addressed the problem of *a priori* feature selection by training networks in sequence, some of them possibly in an unsupervised manner, and using the output of a trained network as the features for a subsequent network. The 'deep' elements of agent architecture correspond in PANGEA to the intermediate states of program execution. PANGEA does not learn them in a fully supervised or unsupervised way, but promotes the agents that reach intermediate states which *relate* somehow to agent goal. Similarly to the core representatives of the deep learning paradigm, patterns in the intermediate states can be sought also in abstraction from the desired output (i.e., in a purely unsupervised manner), which opens the possibility of applying it to problems where the number of labeled training examples is for some reasons scant.

4 Discussion

In relation to the exploration-exploitation perspective outlined in the Introduction, PANGEA can be seen as a hybrid of exploratory, randomized evolutionary search and exploitative, deterministic, 'directional' machine learning, with the former one responsible for inventing novel, intermediate-level concepts, and the latter for relating them in a causal way to agent's goal and building so learning gradients. Importantly, the boundary between these two facets is blurred in this approach, because potentially any effect of exploration can be utilized to build a gradient that guides exploitation.

We find it particularly interesting that exploitation, which involves here such high-level notions like relatedness or causality, can be effectively realized with straightforward off-shelf machine learning tools. On the other hand, program traces are likely to exhibit more sophisticated regularities, like analogies, which the conventional machine learning algorithms and attribute-value representation may be unable to capture. For them, other mechanisms should be sought, with those proposed in (Hofstadter 1984; Mitchell 1993; Marshall 2006) being interesting options.

In a broader perspective, this research suggests that choosing a specific representation (a substrate) for processing percepts and issuing actions may be less important than the evolutionary computation community have traditionally assumed, and that the behavioral dynamics exhibited by such a substrate can be more essential to judge an agent's prospects. For the particular domain of genetic programming, this may for instance imply that program representation, a traditional line of divide between different GP genres (LISP-like tree structures, linear programs, graphs), may be not so critical. The importance of the behavioral (and dynamic, as opposed to static and structural) aspects has been raised in past AI and cognitive science research (Beer 1995; Gelder 1997), but relatively little has been proposed to harness the side effects of agent's behavior to guide its learning, particularly within the evolutionary framework.

Just as 'passing the Turing test' should be not be considered as a goal, but rather as a side-effect of a 'suitably architected' system (Swan and York 2012), we should be less concerned initially with 'solving the problem at hand' and invest *much* more computational effort in building up a system of correlates between agent environment and external mechanism. With particular respect to traditional machine learning and optimization, there is generally extreme descriptive paucity of the features of both environment and search trajectory. The prevalent 'let the dog see the rabbit' approach tends to proceed immediately to online search or applies a vectorized ML approach to a feature vector selected *a priori* by the experimenter. The further success of such approach is predicated on the application of considerable human ingenuity on a 'per-problem class' basis, as is evidenced by the daily practice of the majority of researchers in this area. Ideally, we would like to free the human from this tedious process by providing the learning agent with

additional learning gradients derived from agent's interaction with the environment (including the seemingly task-unrelated side-effects of that interaction, like in PANGEA), or from external knowledge sources.

5 Conclusion

In typical intelligent systems, of all the richness of interactions between agents and environment (or between other constituents of a system), only a minute fraction is usually traced and exploited. For that instance, in genetic programming considered in this study, only the final outcome of program execution is subject to evaluation. In a conventional layered neural network, only the activity in the output layer is serves for error estimation and implicitly influences the training signal. Other manifestations of agent activity, which do not strictly correlate with the task being solved, are usually ignored, or even penalized.

Contrary to widely held opinion, this is not necessarily desirable. Such manifestations, which by being not being directly rewarded can be deemed emergent (Altenberg 1994), carry potentially useful information about the characteristics of an environment (a programming task and the accompanying training data in this paper) and the nature of agent interaction with it. Moreover, they can be used to augment the training signal and so improve agent performance, which we demonstrated in this study.

Can the considerations presented above help us deciding how should intelligence be abstracted in AI research? Building upon the results summarized in this work and in consonance with other reports (Mitchell 2011), we argue that more emphasis should be put on the functional aspects (how agents behave) than on representations and substrates (how agent function is implemented). By extrapolating the architecture studied in this paper, we postulate that intelligence should (or at least can) be abstracted as processes that are good at recognizing the dynamical systems attractors that arise at one spatio-temporal scale, and presenting them in a compressed (or 'reified') form to similar processes one level higher up. In this paper, we demonstrated an architecture that involves only two such levels, the lower one realized by a machine learning algorithm that looks for regularities ('attractors') in agent dynamics, and the upper one implemented by an evolutionary process that drives the search according to the outcomes submitted by the lower lever. However, nothing forbids extending this architecture with additional levels.

Such behavioral, dynamical abstractions have the chance of becoming a 'common platform' for many architectural substrates typically used in AI. An important common feature of such substrates is that, apart from trivial cases, they are all *complex* in forming more or less sophisticated structures of components (predominantly hierarchies thereof). This holds for neural networks, symbolic representations, Markov decision processes, computer programs in GP, and many more. For such representations, the interactions between components and their outcomes should be considered more important than the components themselves. The rationale for that claim is that the components themselves do not 'do' anything: until one lets them interact, little can be said about their nature. This suggests that adopting the perspective characteristic for complex systems (Mitchell 2009) can be also useful in this context.

The relatively low popularity of dynamic (functional, behavioral) abstractions is probably due to their high complexity, which requires more advanced scientific apparatus. Indeed, modeling and analysis of dynamical processes (like trajectories of program behavior here) can be much more difficult than for static structures that gave rise to those processes (programs, networks, etc). However, as our experience with PANGEA demonstrates, a 'written transcript' of agent behavior can be analyzed with respect to the considered task using popular automatic data analysis tools, and the results of such analysis can be easily transformed into useful learning guidance. In practice, we do not necessarily need complete understanding of agent behavior to make certain aspects of that behavior useful for building a more effective intelligent system (though, obviously, having such insight can be helpful). Given the abundance of tools that mimic various aspects of human 'thirst for patterns', many of them much more sophisticated than the decision trees used in PANGEA (like (Hofstadter 1984)), we regard this abstraction and the architecture proposed in this study as an interesting alternative to conventional approaches.

Acknowledgments

K. Krawiec acknowledges financial support from the National Science Centre grant DEC-2011/01/B/ST6/07318. J. Swan acknowledges support from EPSRC grant EP/J017515/1.

References

Altenberg, L. 1994. Emergent phenomena in genetic programming. In *Evolutionary Programming – Proceedings of the Third Annual Conference*, 233–241. World Scientific Publishing.

Beer, R. D. 1995. A dynamical systems perspective on agent-environment interaction. *Artificial Intelligence* 72(1(2)):173 – 215.

Gelder, T. V. 1997. The dynamical hypothesis in cognitive science. *Behavioral and Brain Sciences* 21:615–665.

Hofstadter, D. 1984. *The Copycat Project: An Experiment in Non Determinism and Creative Analogies*. A.I. Mema. Massachusetts Institute of Technology, Artificial Intelligence Laboratory.

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

Krawiec, K., and Swan, J. 2013. Pattern-guided genetic programming. In *Proceedings of the fifteenth international conference on Genetic and evolutionary computation conference*, GECCO '13. Amsterdam, The Netherlands: ACM.

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

Marshall, J. B. 2006. A self-watching model of analogy-making and perception. *J. Exp. Theor. Artif. Intell.* 18(3):267–307.

Mitchell, M. 1993. *Analogy-making as Perception: A Computer Model*. Cambridge, MA: MIT Press.

Mitchell, T. M. 1997. *Machine Learning*. New York, NY, USA: McGraw-Hill, Inc., 1 edition.

Mitchell, M. 1998. *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press.

Mitchell, M. 2009. *Complexity: A Guided Tour*. Oxford University Press, USA.

Mitchell, M. 2011. Ubiquity symposium: Biological computation. *Ubiquity* 2011(February).

Poli, R.; Langdon, W. B.; and McPhee, N. F. 2008. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`. (With contributions by J. R. Koza).

Ranzato, M.; Susskind, J.; Mnih, V.; and Hinton, G. 2011. On deep generative models with applications to recognition. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, 2857–2864. Washington, DC,: IEEE Computer Society.

Rissanen, J. 1978. Modeling By Shortest Data Description. *Automatica* 14:465–471.

Schmidhuber, J. 1991. Curious model-building control systems. In *In Proc. International Joint Conference on Neural Networks, Singapore*, 1458–1463. IEEE.

Spector, L., and Robinson, A. 2002. Genetic programming and autoconstructive evolution with the push programming language. *Genetic Programming and Evolvable Machines* 3(1):7–40.

Swan, J., and York, W. 2012. Taking turing seriously (but not literally). In *AISB/IACAP 2012 World Congress*.