A Coevolutionary Approach to Representation Development

Edwin D. de Jong *edwin@cs.brandeis.edu*
Department of Computer Science
Brandeis University, Waltham, MA 02454-9110

Tim Oates *oates@cs.umbc.edu*
Computer Science and Electrical Engineering Department
University of Maryland Baltimore County
Baltimore, MD 21250

Abstract

The representation of a learning or search problem can greatly impact its performance. An alternative to hand-constructing an appropriate representation is to let the learning method adapt its own representation. We investigate this in a setup where building blocks and assemblies thereof are coevolved. Building blocks may employ other building blocks, thus leading to hierarchical constructions. In experiments, this is observed to lead to highly compact representations of sequences that are useful as building blocks. The method is able to solve problems requiring long sequences of primitive operators. Control experiments using a conventional evolutionary method were much less efficient, and did not find solutions to the problems. Limitations of the current method are discussed.

Keywords: Development of representations, Coevolution, Bias learning

1. Introduction

The representation of a learning or search problem[1] is known to greatly influence the efficiency with which the problem can be learned, see e.g. (Lenat & Brown, 1985). While careful design of the representation is important in practical applications of learning methods, the potential of this approach is limited by the skills and imagination of the practitioner, and it must be done anew for each learning problem. Here, we consider how a method for learning or search might develop the representation it employs as part of the learning problem in such a way that the representation is based on the problem.

[1] We consider a representation to be a set of variables and a corresponding interpretation that determines to which elements in the learning problem the variables refer.

2. Background

While some methods in machine learning such as Bayesian networks offer the potential to learn structure, most learning methods are based on some form of gradient descent, and operate by adapting the parameters rather than the structure of systems. One exception is Layered Learning (Utgoff, Stracuzzi, & Cochran, 2001), the idea of constructing new features out of previously selected features. This idea is also explored by Piater in his thesis (Piater, 2001). Another direction within machine learning concerned with transforming the initial representation is constructive induction, see e.g. (Kramer, 1994; Matheus & Rendell, 1989).

In evolutionary computation, the idea of finding solutions to large problems by combining building blocks has a long history, see e.g. (Holland, 1975). Many such methods view building blocks as high fitness schema, i.e. variable settings that, when completed to form solutions, have high fitness on average. However, it cannot always be assumed that one setting for a particular set of variables suffices. Recently, algorithms have become available that do not make this assumption, by considering multiple values for the variables in a building block, while still reducing the combinations that are considered, see e.g. SEAM (Watson & Pollack, 2000) and BOA (Pelikan, Goldberg, & Cantu-Paz, 1999). A key idea in SEAM is to evaluate a combination based on the performance of contexts that employ it. This idea is used here in a variable length setup. Related methods in genetic programming that use encapsulation include GliB (Angeline & Pollack, 1992) and ADFs (Koza, 1994). The second idea that characterizes our method is the use of strict tests to determine whether a new building block will be formed. By imposing strict tests, the inclusion of unnecessary elements in building blocks can be reduced. This improves the potential of the method to form many-layered hierarchical structures of elements as building blocks.

An important intuition behind the construction of building blocks is that by limiting the combinations of values that will be considered for some set of the variables in a problem, the search space is effectively reduced, since only certain parts of the space will be visited. This makes clear that the construction of building blocks or representations represents a *bias*. An interesting feature of this bias is that it is not defined entirely by the researcher, as is the case when a search space is limited by excluding certain elements, or skewing the distribution of the parts of the space that will be visited. Rather, the parts of the space that will be visited depend on the particular building blocks that are formed, and is thus partially determined by the problem. Thus, the development of representations can be seen as a form of bias learning (Baxter, 2000; Gordon & (eds.), 1995).

A popular form of bias learning is multi-task learning, where a sequence of related tasks is performed so that the learning method may pick up useful biases in early problems and employ them in later ones, e.g. (Baxter, 1995; Caruana, 1997; De Jong & Pollack, 2001; Peshkin & Jong, 2002). Here, bias learning is performed within a single learning problem.

3. A Coevolutionary Approach

In this article, we view the development of representations as a coevolution problem by coevolving building blocks and assemblies thereof. Coevolution refers to a branch of evolutionary computation where the evaluation of individuals is influenced by other evolving individuals[2]. Coevolution is of interest in that it provides a way in principle to let a search process develop its own evaluation criteria. For example, in the coevolution of a backgammon player (Pollack & Blair, 1998), the need for providing an evaluation function for backgammon strategies is circumvented by letting the players play against other evolving individuals, so

[2]More precisely, in coevolution this influence is not limited to scaling effects as it is in conventional evolution, but can affect the *ranking* of individuals.

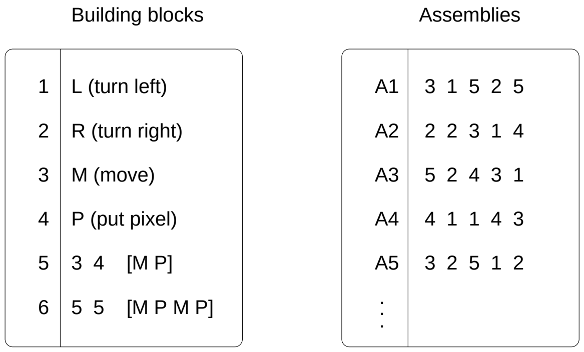

Figure 1. A coevolutionary setup for the development of representations. Assemblies consist of building blocks, and are evaluated based on their performance on the problem. Building blocks are evaluated based on their role in assemblies; if a frequent combination of two building blocks is the best combination for the role they fulfill in an assembly, the combination forms a new building block and enters the building block population. This leads to a hierarchy of building blocks, and allows to make large non-random variations of assemblies by means of a single mutation.

that only the rules of the game must be provided.

The way in which coevolution develops evaluation criteria is by basing evaluation on other evolving entities. Here, this idea is used to evaluate candidate building blocks. While complete individuals can be evaluated based on their performance, i.e. some task-related fitness or error measure, this is not the case for building blocks. The value of a building block follows from its potential to contribute to the fitness of individuals that employ it. Thus, building blocks are evaluated based on the extent to which they contribute to the assemblies that employ them.

3.1. Coevolving Building Blocks and Assemblies

The setup that will be employed here involves coevolution of building blocks and assemblies. Figure 1 shows an illustration. An assembly is a candidate solution, and consists of one or more building blocks. Initially, the set of building blocks corresponds to the variables of the problem. As the search progresses, new building blocks are formed by combining pairs of existing building blocks. This allows for the formation of a hierarchy of building blocks.

In detail, the mechanism operates as follows. Assemblies are evolved using a genetic algorithm with mutation and crossover. Interestingly, since the building blocks provide a potential to incorporate sequences of primitives of arbitrary length, the assemblies can be limited to a very small length (5 building blocks in the experiments here); this avoids the issue of bloat or code growth. As a result of this, the role of the assemblies in the setup may be viewed as providing a look-ahead search (Juille, 1999) that identifies effective combinations of building blocks, which in turn can become new building blocks.

After each generation, the most frequent pair of building blocks occurring in assemblies is considered for consolidation as a building block. However, in order to avoid the formation of spurious building blocks, a stringent test is imposed on candidate building blocks. To evaluate a candidate building block, we take an assembly that employs the pair of existing building blocks forming the candidate building block. We then consider all possible alternative combinations, i.e. all combinations of two existing building blocks other than the candidate combination. Only if the score of the assembly using the candidate building block is at least as good as the scores of the assemblies that are obtained by replacing the candidate building block with any of the alternative building blocks, and better than some of these scores, we create the building block and add it to the set of building blocks.

3.2. Bias Learning

Our motivation for investigating the development of representations as part of learning is that it may provide a way to address certain learning problems that involve large search spaces. Inevitably, any form of generalization corresponds to a bias that can make the learning problem highly effective for particular kinds of problems, but unsuitable for others (Mitchell, 1980).

To clarify for what type of problems the approach we describe here may be of benefit, we describe biases incorporated by the method. Three main forms of bias are employed, see figure 2:

- Modularity
- Repetitive modularity
- Hierarchical modularity

Modularity is the property that several variables in a problem are dependent on one another as to what their (near-)optimal settings, while the dependencies between the module and variables outside the module are weak compared to the former dependencies. Thus, the settings of variables inside a module can be considered without considering all possible combinations of the variables outside the module. This potentially

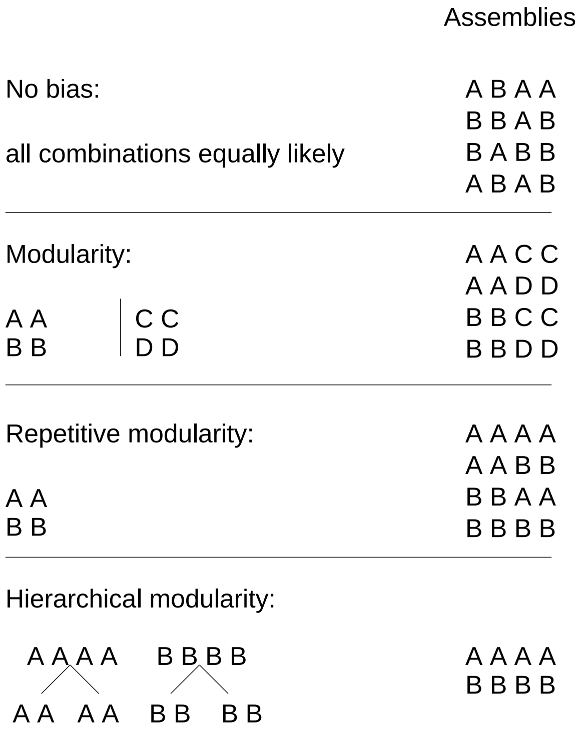

Figure 2. Three forms of bias are employed to make search in large spaces feasible for certain problems: modularity, repetitive modularity, and hierarchical modularity

provides a great increase in the complexity of problems that can be successfully addressed (Simon, 1968). A modularity bias refers to the assumption that modularity is present in the problem. By searching for pairs of elements in assemblies that have an optimal setting in the context defined by the assembly and consolidating such pairs, we employ a modularity bias.

A particular form of modularity bias is that of repetitive modularity; once a particular combination of elements is found to be useful, that combination will be used with a higher probability in other parts of assemblies. By making successful combinations of elements available as new building blocks, the probability that such building blocks will be inserted in other parts of assemblies is increased, thus resulting in a repetitive modularity bias.

New building blocks are formed by combining elements of assemblies, and assemblies are likely to use non-primitive building blocks. Thus, new building blocks that are formed are likely to be trees of existing building blocks. This produces a hierarchical modularity bias that makes it possible for assemblies to specify long, highly non-random sequences of primitive elements using only a few variables.

Finally, apart from the different forms of modularity that are used, an important form of bias is that of incremental construction; by starting with assemblies that employ primitive building blocks only, the search begins by considering short combinations of primitives. The intuition that incremental construction is required to arrive at long specifications has been noted in the literature (Harvey, 1992). This idea can only bring benefit to the extent to which the choices prior to description length increments can be maintained, thus it implies a modularity bias.

4. Task Description

The aim of this work is to contribute to the development of pattern recognition methods that can identify highly specific and complex patterns, i.e. patterns whose specification involves a large numbers of variables. Therefore, we define a concept learning task where a correct solution requires a long description. Concept descriptions are sequences of primitives (TURN LEFT, TURN RIGHT, MOVE, and PUT PIXEL). The interpretation of a sequence produces a bitmap. This bitmap is compared to the training image, and the fraction of coinciding pixels is used as a membership value for the concept to be learned. The difference between this membership value and the binary class label yields the classification error that is returned to the learning method. While the representation of concepts is somewhat involved, it allows for capturing aspects of the image in such a way that these aspects can be used to modify the concepts that will be considered in later stages of the search.

The training images are based on 16x16 bitmaps containing simple line drawings, see figure 3. At each generation, 20 positive training examples are generated, based on the target image, along with 20 negative training examples. To generate positive examples, each pixel in the image is inverted with P=0.1. For the negative examples, each pixel is on with P=0.5. Perfect solutions for these four problems require between 50 an 150 primitive operators, and thus satisfy the criterion that long descriptions are required for correct solutions.

5. Results

Figures 4 and 5 show the results of using the coevolutionary method for representation development that has been described. The graphs show the Hamming distance between the image specified by the assemblies and the target image, averaged over ten runs. This measure gives better insight into the results of learning than the classification error that the learning method itself receives, since the latter reflects the noise added to the images.

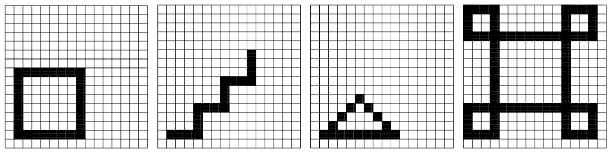

Figure 3. Test images used in the experiments: square, staircase, triangle, and daisy.

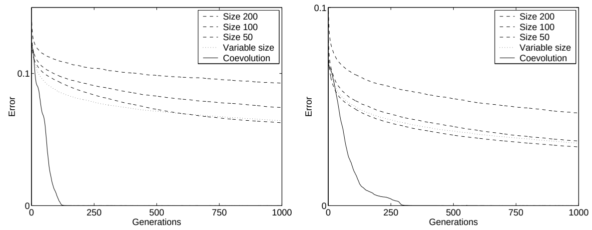

Figure 4. Results of the square (left) and staircase (right) experiments. The graphs show the average error over evolutionary time.

As a control, we took the same algorithm but removed the capacity to add building blocks to the given set of primitives. This renders the algorithm equivalent to a normal genetic algorithm. To compensate for the fact that the assemblies are too short to represent correct solutions, the first control uses a variable length setup, where the initial length of assemblies is 5 as in the coevolutionary case. In addition, we tried a fixed length setup for three different sizes (50, 100, and 200). As the results graphs show, developing representations as part of the learning process can result in great performance increases, and can render otherwise infeasible problems tractable.

To provide more insight into the operation of the method, we consider an example run. Figure 6 shows the error for the first run of the coevolutionary method in the daisy experiment. The graphs displays interleaved periods where progress speeds up and slows down. In control runs that were inspected, progress always slowed down, as in the averaged graphs. Our interpretation of the phenomenon is that once new building blocks are found, the search process can speed up by searching in terms of the new representations.

Inspection of the run showed that by generation 30, building blocks had developed that specify 8-pixel lines in the most compact way possible, see figure 7. Shortly

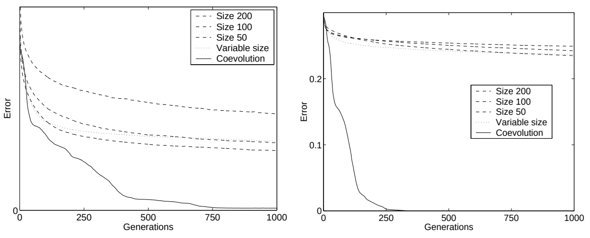

Figure 5. Results of the triangle (left) and daisy (right) experiments. The graphs show the average error over evolutionary time.

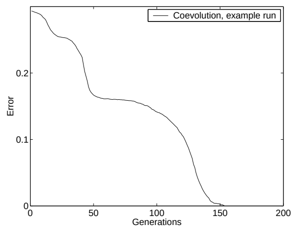

Figure 6. Example run of the daisy experiment.

after this, the error graphs displays a steep descent until generation 50, which resulted in assemblies specifying the complete inner square of the figure. The next and final descent corresponds to the addition of the four smaller squares at the corners of the image.

6. Discussion

A limitation of the current initial representation is that it can only represent fixed templates, i.e. particular configurations of pixels. One direction that could be explored from here is to employ primitives that provide the potential to perform computation by including loops, internal variables, and conditional statements. Such primitives are commonly used in genetic programming, and the encapsulation of subroutines in genetic programming is a well-known principle (Koza, 1994; Angeline & Pollack, 1992). The approach described here provides a way to determine the value of building blocks by assessing their role in assemblies, rather than by assessing their performance using the fitness function for complete individuals of the task. A key idea it employs is to base the decision of whether to include a particular new building block on a comparison with other combinations that could fulfill its role. The use of a strict tests in making this decisions reduces the number of unnecessary elements in building blocks. This is of particular importance in the formation of hierarchical structures.

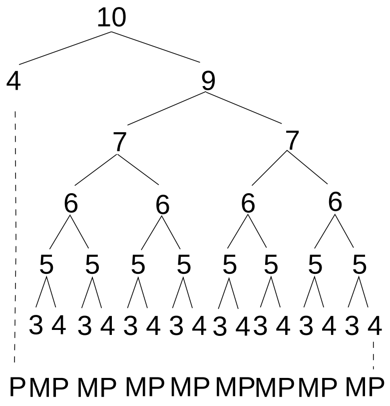

Figure 7. Building block developed in the example run which draws a straight line of length 9 (see text).

7. Conclusions

We have described a coevolutionary approach to learning in which representations develop as part of learning. The setup consists of an evolving population of assemblies that are evaluated based on their performance in the problem, and a co-evolving population of building blocks that these assemblies can employ. The distinguishing feature of the work is that building blocks are evaluated based on their role in assemblies in a variable length setup. In order to clarify for what type of problems the approach may provide an efficient method, we describe the biases employed by the method. These include modularity, repetitive modularity, hierarchical modularity, and incremental construction.

First experiments with the method have been described using pattern recognition tasks. The method was found to bring a substantial performance increase compared to control experiments where the set of building blocks was fixed to be the initial set of primitives. Building blocks were found that capture aspects of the problem in a compact way.

Research directions that could be explored in further research include refined criteria for determining whether a candidate building block is to be created, and extending the expressiveness of the primitives by providing them with the potential to perform computation.

Acknowledgement

The authors want to thank Richard Watson for useful comments on this paper. EdJ gratefully acknowledges an NWO Talent-fellowship.

References

Angeline, P. J., & Pollack, J. B. (1992). The evolutionary induction of subroutines. In *Proceedings of the fourteenth annual conference of the cognitive science society* (p. 236-241). Bloomington, Indiana, USA: Lawrence Erlbaum.

Baxter, J. (1995). Learning internal representations. In *Proceedings of the 8th annual conference on computational learning theory (COLT'95)* (pp. 311–320). New York, NY, USA: ACM Press.

Baxter, J. (2000). A model of inductive bias learning. *Journal of Artificial Intelligence Research*, *12*, 149-198.

Caruana, R. (1997). Multitask learning. *Machine Learning*, *28*, 41-75.

De Jong, E. D., & Pollack, J. B. (2001). Utilizing bias to evolve recurrent neural networks. In *Proceedings of the international joint conference on neural networks* (Vol. 4, p. 2667-2672).

Gordon, D., & (eds.), M. desJardins. (1995). Special issue on bias evaluation and selection. *Machine Learning*, *20*(1/2).

Harvey, I. (1992). The SAGA cross: the mechanics of recombination for species with variable-length genotypes. In R. Männer & B. Manderick (Eds.), *Parallel problem solving from nature* (Vol. 2, p. 269-278). Amsterdam: North-Holland.

Holland, J. H. (1975). *Adaptation in natural and artifical systems.* Ann Arbor, MI: University of Michigan Press.

Juille, H. (1999). *Methods for statistical inference: Extending the evolutionary computation paradigm.* Unpublished doctoral dissertation, Brandeis University.

Koza, J. R. (1994). *Genetic programming II: Automatic discovery of reusable programs.* Cambridge, MA: MIT Press.

Kramer, S. (1994). CN2-HCI: A two-step method for constructive induction. In T. Fawcett (Ed.), *ML-COLT workshop on constructive induction and change of representation* (pp. 33–39). New Brunswick, NJ.

Lenat, D., & Brown, J. (1985). Why AM and EURISKO appear to work. *Journal Artificial Intelligence*, *23*(3), 269–294.

Matheus, C. J., & Rendell, L. A. (1989). Constructive induction on decision trees. In N. S. Sridharan (Ed.), *Proceedings of the 11th international joint conference on artificial intelligence* (pp. 645–650). Detroit, MI, USA: Morgan Kaufmann.

Mitchell, T. M. (1980). *The need for biases in learning generalizations* (Technical Report No. CBM-TR-117). New Brunswick, New Jersey: Department of Computer Science, Rutgers University.

Pelikan, M., Goldberg, D. E., & Cantu-Paz, E. (1999). BOA: The bayesian optimization algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, & R. E. Smith (Eds.), *Proceedings of the genetic and evolutionary computation conference* (Vol. 1, pp. 525–532). Orlando, Florida, USA: Morgan Kaufmann.

Peshkin, L. M., & Jong, E. D. de. (2002). Context-based policy search: transfer of experience across problems. In *Proceedings of the icml-2002 workshop on development of representations.* Sydney, Australia.

Piater, J. H. (2001). *Visual feature learning.* Unpublished doctoral dissertation, University of Massachusetts Amherst.

Pollack, J. B., & Blair, A. D. (1998). Co-evolution in the successful learning of backgammon strategy. *Machine Learning*, *32*(1), 225–240.

Simon, H. A. (1968). *The sciences of the artificial.* Cambridge, MA: MIT Press.

Utgoff, P. E., Stracuzzi, D. J., & Cochran, R. P. (2001). *Many-layered versus few-layered learning* (Technical Report No. TR-01-14). Amherst, MA: University of Massachusetts Computer Science Deptartment.

Watson, R., & Pollack, J. (2000). Symbiotic combination as an alternative to sexual recombination in genetic algorithms. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, & H.-P. Schwefel (Eds.), *Parallel problem solving from nature, ppsn vi* (Vol. 1917). Springer Verlag.