Reports

Machine Learning and Inference Laboratory

Adaptive Anchoring Discretization for Learnable Evolution Model:

The ANCHOR Method

Ryzsard S. Michalski
Guido Cervone

MLI 01-3
P 01-2
May, 2001

School of Computational Sciences

George Mason University

ADAPTIVE ANCHORING DISCRETIZATION FOR LEARNABLE EVOLUTION MODEL: THE ANCHOR METHOD

Ryszard S. Michalski and Guido Cervone

Machine Learning and Inference Laboratory
George Mason University
Fairfax, VA 22030-4444

{michalski, gcervone}@mli.gmu.edu
http://www.mli.gmu.edu

Abstract

To apply a symbolic learning method to learning in a continuous representation space, the variables spanning the space need to be discretized. When the space is very large, a problem arises as to how to determine a discretization scheme for each variable that is both efficient and effective. This task is particularly important when applying Learnable Evolution Model to optimization problems with very large number of continuous variables. The presented method, called ANCHOR, starts with a low discretization precision of the variables, and then increases the discretization precision in the subranges indicated by the analysis of the descriptions learned using a lower precision. The method has been incorporated in the LEM2 system implementing the Learnable Evolution Model. Experiments with ANCHOR have demonstrated a significant advantage of the method over a fixed discretization method, and enabled LEM2 to optimize functions of large number of continuous variables very effectively.

Acknowledgments

This research has been conducted in the Machine Learning and Inference Laboratory at the School of Computational Sciences, George Mason University. The Laboratory's research has been supported in part by the National Science Foundation under Grants No. IIS-9906858 and IIS-0097476, and in part by the UMBC/LUCITE #32 grant.

1 Introduction

The recently introduced Learnable Evolution Model (LEM) employs Machine Learning to guide evolutionary computation (Michalski, 2000). Specifically, at each step of the evolutionary computation, it employs a machine learning system to generate hypotheses discriminating between groups of high and low fitness individuals, and then uses these hypotheses to generate new individuals. Although LEM does not impose any restriction on what kind machine learning method is used, in the current implementations we have chosen the AQ learning method due to several features particularly useful for LEM. One such feature is the ability to induce *attributional rules* that are more general than *atomic rules* typically generated by rule learning methods While atomic rules use only conditions in the form <attribute relation value>, attributional rules can use more elaborate conditions, such as: <attribute relation subset-of-values) or <attribute relation attribute> (Michalski, 2001). Another feature useful for LEM is that AQ can control the level of generality of the learned rules (Kaufman and Michalski, 2000).

If LEM is used for problems in which individuals are represented in terms of continuous variables, the machine learning program must be able to cope with such variables. In the AQ learning method, there are two approaches to handling continuous variables: (1) discretization, and (2) interval representation. The first approach requires continuous variables to be discretized. The second method handles continuous variables by creating conditions in the *form [attribute = lb..ub]*, where *lb* and *ub* are the lower bound and upper bound, respectively, of the interval of variable values. To satisfy the condition, the variable must take a value within such an interval.

The discretization method is easier to implement and can be more efficient if discretized domains are relatively small. The interval method may be more appropriate if the conditions on variables have to delineate very precise borders. It is however more difficult to implement. Moreover, since many problems include different types of variables, both metric (absolute and ratio represented as continuous), and non-metric (nominal or ordinal), then an AQ program using interval representation must simultaneously handle two representations, one for metric and one for non-metric variables. Such two-prong representation is required for handling internal disjunctions in the case of nominal variables, and is desirable for the sake of efficiency.

The discretization approach has been implemented in most of the AQ learning programs, in particular in AQ18 (Kaufman and Michalski, 2000) that has been employed in LEM2 system. The interval representation has been employed in AQ20 (Cervone, Panait, and Michalski, 2001). It is, however, an open problem which approach is most appropriate for LEM in which application domains.

This paper describes a discretization method that has been specifically developed for applying AQ18 program in the Learnable Evolution model. The method specifically addresses the problem of applying LEM to the optimization of functions of a very large number of continuous variables that can vary in arbitrary ranges. Clearly, if the problem solution requires a representation of one or more variables with high precision, then such a precision must be used in LEM/AQ for these variables. It is not known a priori, however, which variables must be represented with high precision and what this precision must be.

A straightforward solution to this problem would be to discretize all the variables with the maximal possible precision that one expects is needed for the given task domain. There are two problems with such a solution. One is that the guess regarding the needed precision may be incorrect, and the chosen precision will be insufficient. In this case, it may be impossible to determine the correct solution no matter how long the evolution process is continued. The second problem is that such a method may lead to unnecessary large variable domains, and by this can significantly impede the performance of the AQ learning system.

To avoid these problems, a new discretization method has been developed, called *Adaptive Anchoring Discretization* or ANCHOR. The term *adaptive* indicates that the precision of discretization is dynamically adapted to the problem during the process of evolution. The term *anchoring* signifies that the domains of attributes consist of consecutively more precise discrete values that are rounded to the nearest whole numbers (anchors). This feature is reflective of human preference for representing numbers/measurements simply, that is, with the minimum number of digits that is sufficient for a given problem.

The ANCHOR method proceeds by making discretizations that are consecutively more precise, but only in the ranges that are hypothesized to require a higher precision. This way, ANCHOR avoids excessive precision, and by that decreases the computational complexity of the learning process.

2 Method

The description of the ANCHOR method is split into three cases. One, when the domain of a variable contains only positive numbers; second, when the domain contains only negative numbers; and third, then the domain contains both types of numbers. Assume that a continuous variable x_i ranges over the interval [*min*,…,*max*], where *min* is the smallest possible real value, and *max* is the largest possible real value in the domain of x_i.

Case 1: **Positive Numbers Only** ($min \geq 0$)

(1) Determine 1st order approximation (FOA):

A. Determine LB = MIN and UB = MAX (where LB stands for the lower bound and UB for upper bound) of the currently considered range.

B. Replace values of x_i by the nearest *first order anchor* (FOA), defined as the best single digit approximation of x_i, which are called the *first order units* or *FOU*.

C. Determine FOU

If MAX > 1, then FOU(x_i) is defined as the digit 1 followed by the number of zeroes equal the number of digits in x_i before "." minus 1. For example, FOU(2359. 98) = 1000.

If MAX ≤ 1, then FOU(x_i) is defined as 0.1. For example: FOU(0. 3897638) = 0.1.

D. Replace x_i by FOA(x_i) = Round(x_i / FOU), where

Round(x_i /FOU) = $\lfloor x_i/FOU \rfloor$, if $x_i/FOU - \lfloor x_i / FOU \rfloor < 0.5$

Round (x_i /FOU) = $\lceil x_i/FOU \rceil$, if $x_i/FOU - \lfloor x_i / FOU \rfloor \geq 0.5$

E. Create a *normalized first order approximation* of x_i, NFOA(x_i), by isomorphically mapping the domain of FOA(x_i) into the range {0,1,…,d_i}.

The last step if done because AQ18 assumes that all variables have domains normalized to a set of consecutive non-negative integers starting with 0. Let us assume now that AQ18 has been applied to a training set, which in the case of LEM consists of high- and low-performing individuals (H-group and L-group, respectively). The resulting hypothesis discriminating between the H- and L-group is in the form of an attributional ruleset (Kaufman and Michalski, 2001; Michalski, 2001). According to the LEM method, the ruleset is then instantiated in different ways to generate new individuals (e.g., Cervone, 2000; Michalski, 2000). The individuals are used to create a new population, e.g., through an intergenerational truncation survival method (original and newly created individuals compete with each other and the k best are selected for the next population, where k is the population size parameter). Individuals in the population are arranged to create a fitness profile function.

F. If the range of the fitness profile function becomes *delta-small* (which is a parameter of the method) and the LEM termination condition has been satisfied, EXIT; otherwise, identify values of variables in the learned rules that need to be discretize further. These are NFOA(x_i) values that occur in atomic conditions of the ruleset (that is, occur as single values in the reference of a condition).

(2) *Determine 2nd order approximation* (*SOA)*:

A. For each value, v, identified above, determine the *second order* LB and UB, as the lower bound and the upper bound, respectively:

LB is defined as: if MAX > 1, then LB= v – 0.5; if MAX ≤ 1, then LB = v - 00.5.

UB is defined as: if MAX > 1, then UB= v + 0.5; if MAX ≤ 1, then UB = v + 00.5.

The expanded range of values is defined as [LB …UB).

B. Determine the second order unit (SOU) for the expanded range defined in A.

If MAX > 1, then SOU(x_i) is defined as the digit 1 followed by the number of zeroes equal the number of digits in x_i before "." minus 2 . For example, SOU(2359. 98) = 100.

If MAX ≤ 1, then SOU(x_i) is defined as .01. For example, SOU(0. 358798) = 0.01.

C. Replace values of FOA(x_i) in the ranges defined by the second order LB and UB, by the nearest second order anchor (SOA), which is the best double digit approximation of x_i (i.e., in terms of the second order units or SOU) in these ranges. The resulting domain is SOA(x_i) of the original domain of x_i.

As an example, consider the case of expanding the value x_i = 3000. In this case:
SOA(x_i) ={2000, (2500, 2600, 2700, 2800, 2900, 3000, 3100, 3200, 3300, 3400), 4000, 5000}

D. Create a normalized second order approximation of x_i, NSOA(x_i), by mapping isomorphically FOA(x_i) and SOA(x_i) in corresponding ranges into one range {1,…,d_i}.
Considering the previous example, NSOA(x_i) = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11, 12}.

E. Continue executing LEM until the range of the fitness profile function becomes delta-small. If the LEM termination condition is satisfied, EXIT; otherwise, go to 3.

(3) *Determine consecutively higher order approximations* in a similar fashion as above, until the *LEM stopping condition* is satisfied.

Case 2: **Negative numbers only** (MAX $\leq$ 0)

Transform x_i to $-x_i$ and apply the same procedure as above to the transformed x_i. The order of values in the domain of NSOA(x_i) will be opposite to the order of negative numbers, that is the higher value of NSOA represent a higher negative value.

Case 3: **Positive and negative numbers** (MIN < 0 and MAX > 0)

Apply the procedure described in Case 1 to the positive numbers and procedure described in Case 2 to the negative numbers. Assume that anchors for negative values are negative integers and anchors for positive values, are positive integers. Map the whole range isomorphically into {0,1…,d_i). The real "0" value of x_i is thus mapped into an integer between 0 and d_i.

3 An example

Suppose that the domain of x_I is a range defined by MIN = -2.3 and MAX = 14.7.

FOA(x_I) ranges over: {-2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}

NFOA(x_I) ranges over {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17}

The original 0 is thus represented by 2.

Suppose that it was found that FOA(x_i) needs to be approximated more precisely for values of NFOA(x_I) equal 5 and 6. In this case, SOA(x_i) ranges over:

{-2, -1, 0, 1, 2, (2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1, 3.2, 3.3, 3.4), (3.5, 3.6, 3.7, 3.8, 3.9, 4.0, 4.1, 4.2, 4.3, 4.4), 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15} and

NSOA(x_i) ranges over {0, 1, 2, 3, 4, …….. , 33, 34, 35, 36}

Suppose now that it was found that values of NSOA(x_i) equal 4, 14, 15 (values 2, 3.4 and 3.5 of SOA) need to be approximated more precisely. The third order approximation, TOA(x_i), would then range over the domain:

{-2, -1, 0, 1, (1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.4), (2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1, 3.2, 3.3) (3.35, 3.36, 3.37, 3.38, 3.39, 3.40, 3.41, 3.42, 3.43, 3.44), (3.45, 3.46, 3.47, 3.48, 3.49, 3.50, 3.51, 3.52, 3.53, 3.54), (3.6, 3.7, 3.8, 3.9, 4.0, 4.1, 4.2, 4.3, 4.4), (5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15} and

NTOA(x_I) would range over {0, 1, 2, 3, 4, …….. , 51, 62, 63}.

As this example indicates, the growth of the domain size of a variable is adjusted to the needs of the problem.

4 Implementation

The Anchor method was implemented using the dynamically resizable *vector* class found in the C++ STL. (Stroustroup, 1997). The vector is defined as

std::vector <double>

that indicates that each cell of the vector is a double precision floating point. The *std::vector* indexes each cell sequentially using an integer. The discretization process involves using the index of the cell containing the real value being discretized. The vector representation was chosen because it is fast and it allows one to insert or delete elements at real time. Consider Figure 1 that shows a fitness landscape spanned over a single variable.

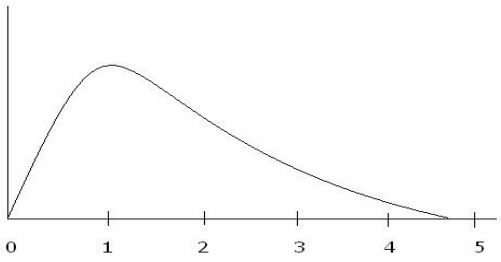

Figure 1. A fitness landscape over the range of a single variable.

At generation 1, the range of the variable [0-5] is discretized into 5 values by taking only the integer part of the real values. At this step, original integer values and discretized values are identical, as shown in Table 1.

Original values	0	1	2	3	4	5
Discretized values	0	1	2	3	4	5

A correspondence between original integer values and discretized values.

Table 1.

The original value is stored in the cells of the vector, and the discretized value is the index of each cell. Suppose now that AQ-learned rules indicate a need for representing the discretized value “1” with a higher precision. A new sequence of values [0.5 .. 1.4] around value “1”, that represents the first order approximation (Table 2).

Original values	0	0.1	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3	1.4	2	3	4	5
Discretized values	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

An example of second order approximation.

Table 2.

New cells are inserted for the new range, and the discretized value, which is the cell index, is automatically adjusted to reflect the new change. The vector representation facilitates cell removal, if the inverse operation is desired, that is, when some range is represented with an excessive precision.

5 Experiments: Comparing ANCHOR with χ^2-based discretization

To test the ANCHOR method, it was experimentally compared with a χ^2-based method (Kerber, 1992). Results below illustrate its performance in comparison to theχ^2-based method on the problem of minimizing the sphere function:

$$f(x) = \sum_{i=1}^{100} x_i^2$$

where each of 100 variables is bound between –5.123 and 5.123. The function has two maxima for each dimension, one at x_i = -5.123 and second at 5.123. Thus, there are in total 2^{100} possible solutions. Every solution requires three decimals of accuracy.

Six experiments were performed, each consisting of five runs that differed in their initial population. In each run, the population consisted of 100 individuals, and each experiment lasted for 10,000 births. Out of six experiments, five used the χ^2 –based method (with 5, 10, 50, 100 and 1000 discretization units for each of 100 variables), and the sixth experiment used ANCHOR. The results are presented in Figure 5.

The vertical axis represents the total elapsed time to find the solution in seconds, while the horizontal axis indicates the average accuracy of the solutions from five runs. The accuracy of each solution was measured here by the ratio of the highest fitness solution found in the given run and the globally maximum solution, expressed in percentage. As shown in Figure 5, increasing the number of discretization units in the χ^2 method yields a more accurate solution, but at the expense of the execution time. The execution time grew very rapidly with the number of discretization units.

For example, for 25 units the process took about 15 seconds, giving accuracy of 78%; and for 1000 units, it took about 240 seconds, giving the accuracy of 99% (these are averages of five runs). The best accuracy, 100%, was obtained by ANCHOR, in about 20 seconds. This means that ANCHOR's accuracy was 100% in all five runs.

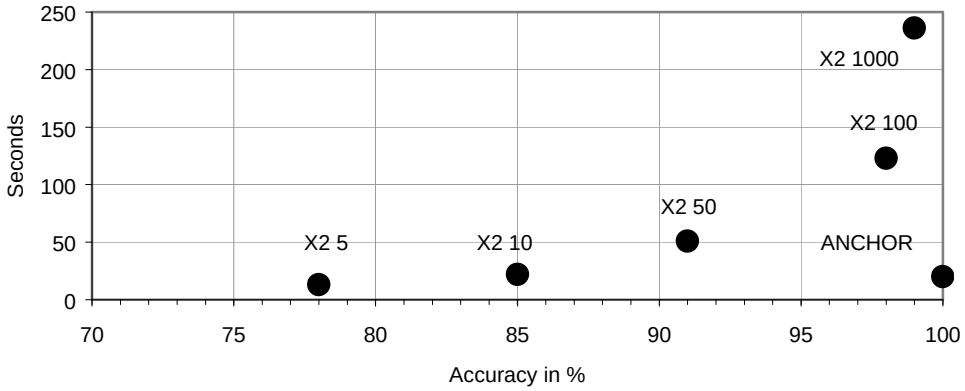

Figure 2. Accuracy and elapsed time for ANCHOR, and χ^2 method using 5, 10, 50, 100 and 1000 discretization units for each of 100 variables.

As shown in Figure 2, ANCHOR outperformed the χ^2 discretization method in accuracy, regardless of the precision of discretization (the number of discrete units). When the number of discretization units was 1000, which is sufficient to capture the exact value of the solution, the χ^2 method still did not produce the 100% accuracy, while its execution time was much longer than that of ANCHOR achieving 100% accuracy (240s vs 20 sec). ANCHOR is implemented in C++ (like the entire LEM2), and the experiments were performed on a SUN Sparc20 with 64MB of RAM, running Solaris 5.7. LEM2 implements both ANCHOR and χ^2–based method to facilitate experimentation.

4 Related Work

Discretization of continuous variables has been investigated extensively both in Machine Learning and in Statistics. Some methods are widely used, for example χ^2-based, which partitions ranges of variables into equal intervals (Kerber, 1992). The χ^2-based method was used in LEM 1. Experiments have shown that the method affects significantly the LEM1's execution efficiency when variables are discretized with high precision.

Ho and Scott (1997; 1998) proposed a method for discretization of continuous variables based on a measure of strength of association between nominal variables, called zeta. The zeta measure is based on minimization of the error rate when each value of an independent variable predicts a different value of a dependent variable.

Grzymala and Stefanowski (2001) proposed two methods: MODLEM-entropy and MOLEM-Laplacian. These methods are integrated into the rule induction algorithm based on rough set theory. The methods differ in the way they find the cut points. The first method uses an entropy function, while the second methods calculates the Laplacian accuracy. The error rate of rules learned by both methods was similar.

Main differences between ANCHOR and other known methods are that it is an iterative method in which the splits the variable range at a given iteration is based on the feedback from the previous iteration, and the discretization employs anchor points. Initially, the discretization precision is very low.

After rules have been learned using so discretized variables, they are used to identify subranges of the variable domain to be selectively discretized with a higher precision. These subranges are identified based on the rules learned with the lower discretization precision. Another important characteristic of ANCHOR is that splits are defined by anchors representing consecutively more precise intervals that are equal within given subranges.

5. Conclusion

The version of the ANCHOR method presented above and implemented in LEM2 is called ANCHOR/E (ANCHOR with *entire* domain representation). This version always maintains a representation of the entire domain of the original variable. An alternative method is ANCHOR/S (ANCHOR with *selective* domain representation) which represents only the subranges of the original domain that are hypothesized as likely containing the target solution.

A modification of ANCHOR/E needed to turn it into ANCHOR/S is straightforward: the ranges that are found not relevant in a given iteration of LEM/AQ would be ignored in the next iteration. This way, only ranges that are found relevant undergo a more precise discretization.

An advantage of the ANCHOR/E version is that allows the system to handle situations in which higher order discretization has been applied to inappropriate ranges, and to locally backtrack. The price for this advantage is the need to maintain larger domains, which makes the method more costly computationally. In contrast, ANCHOR/S uses reduced ranges, which increases its efficiency, but at the cost of preventing local backtracking. If it appears that backtracking is needed, LEM would invoke a start-over operation, repeating evolutionary computation with a different initial population using the first order discretization. To get a more precise evaluation of the relative merits of ANCHOR/E and ANCHOR/S versions, this issue needs to be investigated experimentally. An interesting problem for further research is to investigate the possibility of using ANCHOR method for other applications than LEM.

References

Grzymala, J, W., Stefanski, J., (2001). "Three Discretization Methods for Rule Induction," *International Journal of Intelligent Systems*, January.

Ho, K. M., Scott P.D. (1997). "Zeta: A Global Method for Discretization of Continuous Variables," *Proceedings of KDD-97, The Third International Conference on Knowledge Discovery and Data Mining*, Newport Beach, CA., pages 191--194, Menlo Park, CA., August. AAAI Press.

Ho, K. M. and Scott. P. D., (1998). "An efficient global discretization method," *Proceedings of PAKDD-98, The Second Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Melbourne, Australia.

Kaufman, K.A. and Michalski, R.S. (2000a). The AQ18 System for Machine Learning and Data Mining: User's Guide. *Reports of the Machine Learning Laboratory*, MLI 00-3, George Mason University, Fairfax, VA.

Kerber, R., (1992). "ChiMerge: Discretization of Numeric Attributes", *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pp. 123-127, San Jose, CA.

Michalski, R.S. (2000), LEARNABLE EVOLUTION MODEL: Evolutionary Processes Guided by Machine Learning," *Machine Learning* 38(1-2), pp. 9-40.

Michalski, R.S. (2001), ATTRIBUTIONAL CALCULUS: A Logic and Representation Language for Deriving Human Knowledge from Computer Data, *Reports of the Machine Learning and Inference Laboratory*, George Mason University, Fairfax, VA, MLI 2001-6.

Stroupstroup, B. (1997). "The C++ Programming Language," Addison-Wesley.

A publication of the *Machine Learning and Inference Laboratory*
School of Computational Sciences
George Mason University
Fairfax, VA 22030-4444 U.S.A.
http://www.mli.gmu.edu

Editor: R. S. Michalski
Assistant Editor: K. A. Kaufman

The *Machine Learning and Inference (MLI) Laboratory Reports* are an official publication of the Machine Learning and Inference Laboratory, which has been published continuously since 1971 by R.S. Michalski's research group (until 1987, while the group was at the University of Illinois, they were called ISG (Intelligent Systems Group) Reports, or were part of the Department of Computer Science Reports).