An Extension of Hill-climbing with Learning Applied to a Symbolic Regression of Boolean Functions

Vladimír Kvasnička
Institute of Applied Informatics
Faculty of Informatics and Information Technologies
Slovak University of Technology
Bratislava, Slovakia
kvasnicka@fiit.stuba.sk

Ladislav Clementis
Institute of Applied Informatics
Faculty of Informatics and Information Technologies
Slovak University of Technology
Bratislava, Slovakia
clementis@fiit.stuba.sk

Jiří Pospíchal
Institute of Applied Informatics
Faculty of Informatics and Information Technologies
Slovak University of Technology
Bratislava, Slovakia
pospichal@fiit.stuba.sk

ABSTRACT

In this paper we discuss an application of simple stochastic optimization algorithm called *the hill climbing with learning* (HCwL) for a study of symbolic regression. A fundamental role in this approach plays the so-called probability vector $w=(w_1,w_2,...,w_n)$, where an entry $0 \le w_i \le 1$ specifies a probability that an *i*-th component of solution (e. g. a bit in binary representation) has a binary 1 value. An integral part of HCwL is a mutation process, where from a current solution x_{old} is created a new solution x_{new} by a stochastic mutation process. The used probability vector w (considered here as a special type of collective memory) serves as an auxiliary device for a construction of new mutated solution x_{new}; in particular, it predicts promising directions during its creation that are specified by the previous history of adaptation process.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic methods

General Terms

Algorithms

Keywords

Symbolic regression, Boolean functions, Hill-climbing with learning

1. INTRODUCTION

The aim of this paper is to discuss an extension of simple stochastic optimization, which is called the hill-climbing with learning (HCwL) to more complex optimization tasks than an optimization of standard binary objective functions. We outline basic principles of the hill climbing with learning, initially introduced by S. Baluja [1, 2, 3] and then developed in great details by the present authors and Martin Pelikan [8]. Moreover, Martin Pelikan [8] has extended HCwL to include pair correlations of entries of binary vectors. In this paper we will study an extension of optimization of binary function to a case of symbolic regression of Boolean functions. Illustrative examples will demonstrate that a concept of HCwL realized by a learning of probability vector represents a very serious acceleration element of simple hill climbing algorithm applied to nontrivial optimization problems.

2. HILL CLIMBING WITH LEARNING

2.1 An Optimization Of Binary Function By Simple Hill-climbing Method

Let *f* be a binary function (mapping)

$$f:\{0,1\}^n \to R \tag{1}$$

which assigns to each *n*-bit vector $x=(x_1,x_2,...,x_n)$ a real number $y \in R$, formally $y=f(\boldsymbol{x})$. Our task is to look for a vector $\boldsymbol{x}_{opt} \in \{0,1\}^n$ that corresponds to a global minimum of f over the domain $\{0,1\}^n$.

$$\boldsymbol{x}_{opt} = arg \min_{\boldsymbol{x}\in\{0,1\}^n} f(\boldsymbol{x}) \tag{2}$$

The most simple stochastic optimization method for solving this task is the so called hill-climbing method [7], which was extended by Baluja [1, 2, 3] and present authors by an introduction of the adapted (learned) probability vector, such extended method is called "hill-climbing with learning" (HCwL). A prototype of these stochastic optimization methods is the so-called *hill climbing* method (algorithmically very simple but usually not very effective). For our further purposes we outline this method in a modified form, where a simple adaptation process for the construction of neighborhood is already introduced and this adaptation process is controlled by a probability vector $w=(w_1,w_2,...,w_n)$.

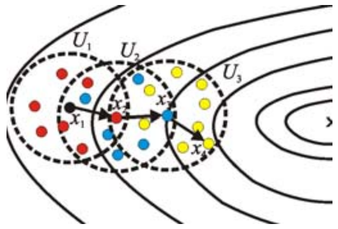

Figure 1. Schematic outline of the simplest stochastic optimization method called "hill-climbing". For simplicity the method is applied to a unimodal objective function with a minimum equal to its global minimum. In the first step, for a randomly generated solution x_1 there is created a neighborhood U_1 specified by (5), and then we determine its local minimum x_2 specified by (6). This new temporary solution is used in the forthcoming step as a "center" of neighborhood. Using local minimum as a "center" is perpetually repeated and the process is finished when in a current neighborhood a better solution than its "center" does not exist.

First, we introduce the so-called *mutation* of an n-bit vector $\boldsymbol{x} \in \{0,1\}^n$ onto another vector $\boldsymbol{x}' \in \{0,1\}^n$ with entries randomly determined as follows

$$x_i' = \begin{cases} 1-x_i & (if\ random < P_{mut}) \\ x_i & (otherwise) \end{cases} \tag{3}$$

(for i=1,2,...,n) where P_{mut} is a probability of flipping a single bit entry and *random* is a random number with uniform distribution from the semiopen interval [0,1). In other words, the mutation of $\boldsymbol{x}$ into $\boldsymbol{x}'$ is a sequential operation which changes stochastically (specified by the probability P_{mut}) single bit entries. Formally, mutation may be considered as a function O_{mut}

$$\boldsymbol{x}' = O_{mut}\left(\boldsymbol{x}; P_{mut}\right) \tag{4}$$

where the probability P_{mut}, on the right hand side, is a parameter of this function. A neighborhood $U(\boldsymbol{x};P_{mut})$ (specified by the probability P_{mut}) of a vector $\boldsymbol{x}$ is composed of n-bit vectors that are created by mutations of $\boldsymbol{x}$

$$U\left(\boldsymbol{x}; P_{mut}\right) = \left\{\boldsymbol{x}' = O_{mut}\left(\boldsymbol{x}; P_{mut}\right)\right\} \cup \boldsymbol{x} \tag{5}$$

A size – cardinality of the neighborhood is a very important parameter of the hill-climbing optimization, the neighborhood is usually composed of a few hundred elements. Within this neighborhood we look for a locally optimal solution $x_{loc,opt}$, which is used in the forthcoming step as a center of a new neighborhood $U\left(x_{loc.opt}; P_{mut}\right)$; in such a way we get a sequence of locally optimal solutions

$$x^{(0)} \to x^{(1)}_{loc.opt} \to x^{(2)}_{loc.opt} \to \dots . x^{(n)}_{loc.opt} \to \dots \tag{6a}$$

where

$$x^{(i+1)}_{loc.opt} = \arg\min_{x \in U\left(x^{(i)}_{loc,opt}; P_{ut}\right)} f\left(x\right) \tag{6b}$$

This iterative local optimization is finished when a few last solutions offer the same functional values (see Fig. 1). As was initially demonstrated by Baluja [1, 2], this extreme simple stochastic optimization algorithm offers in many cases results that are fully competitive with GA results.

2.2 Hill-Climbing With Learning

An interesting possibility how to introduce learning features to simple hill climbing algorithm has been considered by Baluja [1, 2, 3] and latterly by Kvasnička, Pelikan, and Pospíchal [7]. They introduced two concepts that effectively modify the hill climbing so that it achieves a close resemblance to genetic algorithms:

(1) A *probability vector* $\boldsymbol{w} = \left(w_1, w_2, \dots, w_n\right) \in [0,1]^n$, its entries $0 \le w_i \le 1$ determine probabilities of appearance of '1' entries in given i-th positions. For instance, if w_i=0(1), then x_i=0(1), for 0<w_i<1 the corresponding x_i is determined stochastically by

$$x_i' = \begin{cases} mutation \quad \left(random \le P_{mut}\right) = \begin{cases} 1 & \left(random \le w_i\right) \\ 0 & \left(otherwice\right) \end{cases} \\ x_i \qquad \left(otherwice\right) \end{cases} \tag{7}$$

It means that in this generalized mutation a single entry x_i of the binary vector $\boldsymbol{x}$ is randomly selected by a probability P_{mut} ; when this entry is selected, then it's binary value is set to 1 with a probability w_i. This means that entries of the probability vector $\boldsymbol{w}$ control a "mutation" process whether the corresponding bits will be set to one or not. It means that the used "mutation" process directs the created "mutated" vectors in a direction, which looks very promising according to the previous history of adaptation process. Let this random generation of n-bit vectors with respect to the probability vector $\boldsymbol{w}$ be formally expressed by a function R

$$\boldsymbol{x}' = R\left(\boldsymbol{x}; \boldsymbol{w}, P_{mut}\right) \tag{8}$$

A neighborhood $U(\boldsymbol{x};w,P_{mut})$ is composed of randomly generated n-bit vectors with respect to a probability vector w is determined by

$$U\left(\boldsymbol{x}; \boldsymbol{w}, P_{mut}\right) = \left\{\boldsymbol{x}' = R\left(\boldsymbol{x}; \boldsymbol{w}, P_{mut}\right)\right\} \tag{9}$$

We shall postulate that its cardinality is always kept fixed, $|U(\boldsymbol{x};\boldsymbol{w},P_{mut})|=N$. When the entries of the probability vector $\boldsymbol{w}$ are slightly above zero or below one, then a "diameter" of $U(\boldsymbol{x};\boldsymbol{w},P_{mut})$ is very small, all its elements are closely related to an n-bit vector $\boldsymbol{x}$ unambiguously determined by the probability vector $\boldsymbol{w}$ and the probability P_{mut}. In the case of simple hill climbing method this corresponds to a situation when the mutation probabilities w_i are very small. On the other hand, if probabilities w_i are near to 1/2, then n-bit vectors constructed by (7-8) span relatively large domain with a center (n-bit vector) roughly deterministically constructed so that x_i=0 (if w_i<0.5) and x_i=1 (if w_i>0.5).

(2) The *learning* of probability vector $\boldsymbol{w}$ is introduced with respect to a best solution found in the neighborhood $U(\boldsymbol{x};\boldsymbol{w},P_{mut})$.

$$\boldsymbol{x}_{opt} = arg \min_{\boldsymbol{x}' \in U\left(\boldsymbol{x}; \boldsymbol{w}, P_{mut}\right)} f\left(\boldsymbol{x}'\right) \tag{10}$$

The probability vector w is modified-updated (learned) by the so-called *Hebbian rule* (well-known learning rule widely used in artificial neural networks)

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \lambda\left(\boldsymbol{x}_{opt} - \boldsymbol{w}\right) \tag{11}$$

where λ is the learning coefficient (a small positive number) The learning rule (11) has a very simple geometric interpretation, the right-hand side of (11) is nothing but a convex combination of two vectors $\boldsymbol{w}$ and $\boldsymbol{x}$. This means that a resulting vector of the convex combination should lie on the straight-line connecting "points" $\boldsymbol{w}$ and $\boldsymbol{x}$ (see Fig. 2) close to the point $\boldsymbol{w}$ (λ is a small positive number, $0<\lambda\ll 1$). In other words, it means that learning rule (11) shifts the probability vector w towards local best solution $\boldsymbol{x}_{opt}$.

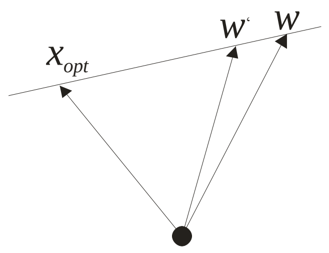

Figure 2. Geometric interpretation of the learning rule (11), the updated probability vector $\boldsymbol{w}'$ lies inside the region on straight-line determined by "points" $\boldsymbol{x}$ and $\boldsymbol{w}$.

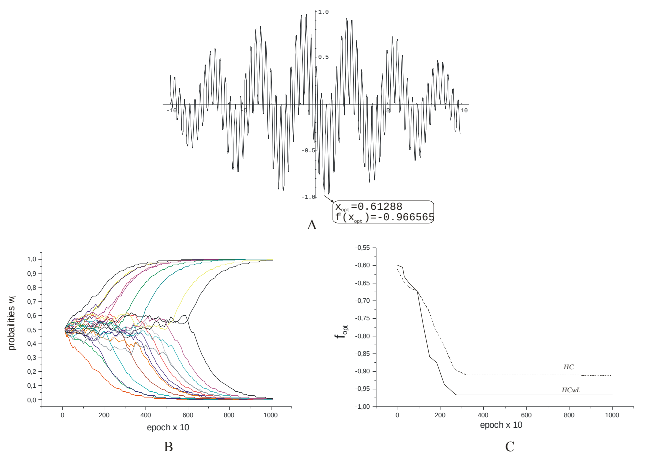

Figure 3. (A) Diagram of multimodal test function f (x) is composed of 56 local minima; the global minimum is marked on the diagram. (B) The progress of the components w_i of probability vector $\boldsymbol{w}$, which is initialized by 0.5 values for all components. During the evolution individual components are asymptotically approaching the binary value 0 or 1. We see that after about 300 epochs components of the vector $\boldsymbol{w}$ are always determined by a given component in the correct solution. This fact may be interpreted as a *collective memory* for the evolution of the population replicator, which manages the reproductive process of mutation on the basis of previous history of the adaptation process. Diagram C displays a plot composed of locally optimal solution x_{opt} . After roughly 250 epochs, these locally optimal solutions are equal to the globally optimal functional value f (x_{opt}) = 0.96657. Of course it might be expected that this evolution without learning (see line HC) usually does not converge to the correct global minimum, but to a local minimum with a higher value.

A few notes about the parallelization of the present method follow. The parallelization is simply achieved by using a pool of probability vectors instead of one probability vector. For all probability vectors independent neighborhoods $U(\boldsymbol{x};\boldsymbol{w},P_{mut})$ are randomly generated. Finally, all probability vectors are updated by (11) with respect to the local optimal solutions assigned to all probability vectors. In order to introduce an interaction between parallel running procedures, it is possible to use an analogue of crossover operator applied for a couple of randomly selected probability vectors. This introduces an exchange of information between subproblems so that at the end of the whole procedure all probability vectors are almost identical. Why the parallel version of HCwL would outperform its single nonparallel version? Independent methods may converge to different regions of the search space, this means that an introduction of "intercommunication" between probability vectors ensures increase of search diversity across the whole search space S. There exists a similar situation for genetic algorithms, where parallel versions usually outperform nonparallel genetic algorithms.

The suggested method is illustrated by a simple multimodal function: $f(x)=e^{-0.01x^2}\sin(8x)\cos(10x)$, determined on an interval $-10\le x\le 10$. As we may see, in the given interval this function has 56 minima, from which one is global $f(x_{opt})$= -0.966565, x_{opt}=0.61288 (see diagram A, Fig. 3). Plots of probabilities w_i are outlined in diagram B, there is easy to see that these single probabilities spontaneously tend either to 1 or to 0. In the midcourse these probabilities fluctuate around 1/2, but after a sufficiently great number of epochs, they each tend either to 1 or 0.

2.3 Symbolic Regression of Boolean Functions

The second illustrative application of HCwL will be presented for symbolic regression of Boolean functions. For a given training set $A_{train}=\{\boldsymbol{x}/\boldsymbol{y}_{req}\}$ we look for a model function $G(\boldsymbol{x};\boldsymbol{\omega})$, where $\boldsymbol{x}$ are variables and $\boldsymbol{w}$ are freely adjusted parameters, such that these model function G perfectly reproduces the training set. Let us define an objective function

$$E(\boldsymbol{\omega})=\sum_i\left(G(\boldsymbol{x}_i,\boldsymbol{\omega})-\boldsymbol{y}_i^{(req)}\right)^2+\kappa_1 n.v.+\kappa_2 n.e. \qquad (12)$$

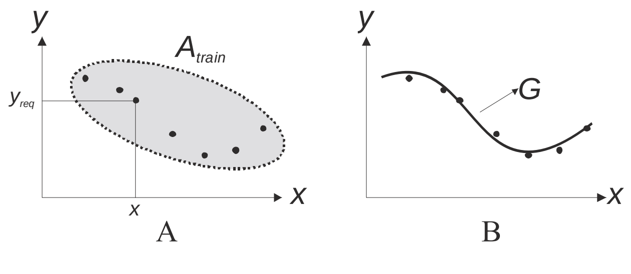

Figure 4. A schematic outline of regression. Diagram B represents the so-called training set composed of a finite set composed of couples of argument and required functional value. Diagram B illustrates a concept of regression, we look for such a model function G, which predicts for given arguments the required functional values as well as possible. In general, the regression may be understood as a special

inductive generalization, we look for such a model function G (which is simultaneously not only very flexible, but also simplest as possible). Then we believe that the resulting model function well predicts the "required" functional values outside the training set. In an opposite case, when we use complicated/sophisticated model function with many degrees of freedom, such a model function is not very well "working" for points outside the training set, an ability of the model function to generalize (i.e. predict functional values outside of training set) is very poor.

where κ_1 a κ_2 are small positive numbers – penalization constants, the first penalization term prefers solutions with a smaller number of vertices (n.v.), while the second one prefers solutions with smaller number of edges (n.e.). Then a symbolic regression consists of a minimization of (12) with respect to parameters $\boldsymbol{w}$ such that a difference between calculated $\boldsymbol{y}_{cal} = G(\boldsymbol{x},\boldsymbol{\omega})$ and required output values $\boldsymbol{y}_{req}$ are minimal. This is an outline of classical version of regression, which is called the ***parametric regression***, where a model function is known, we look only for parameters of this function in such a way that a required - measured and predicted functional values are as close as possible. Another type of regression is the so-called ***symbolic regression***, where we look for a model function *G* such that it perfectly reproduces the training set. Many generations of numerical mathematicians dreamed about symbolic regression, but only John Koza [6] surmounted all obstacles that should be solved by applying an evolutionary technique called the ***genetic programming***. The goal of the present subsection is to show that HCwL approach is effectively applicable to the solution of the problem of symbolic regression for Boolean function as well.

The basic concepts of the present approach are Boolean functions that are represented by acyclic syntactic (derivation) trees

$$f:\{0,1\}^m \to \{0,1\}^n \tag{12a}$$

or

$$(y_1,...,y_n) = f(x_1,...,x_m) \tag{12b}$$

An assumption of acyclic syntactic tree ensures their simple recursive calculation of functional values; going successive from input vertices (representing input variables) through inner vertices to output vertices (representing functional variables), we may unambiguously and successively calculate the output variables.

In graph theory [5] a fundamental property of acyclic oriented graph is proved: an oriented graph $\boldsymbol{G}=(V,E)$ is acyclic if its vertices may be indexed such that

$$\forall(v,v')\in E:\quad \varphi(v) > \varphi(v') \tag{13}$$

An indexing, which satisfies the above simple condition is called the ***canonical indexing***. Following this theorem each oriented acyclic graph may be canonically indexed. Vertices of canonically indexed graph may be divided into the following three disjoint subsets: (1) ***input vertices***, these vertices are adjacent only to outgoing edges, (2) ***inner vertices***, these vertices are adjacent simultaneously to incoming and outgoing edges, and (3) ***output vertices***, these vertices are adjacent only to incoming edges.

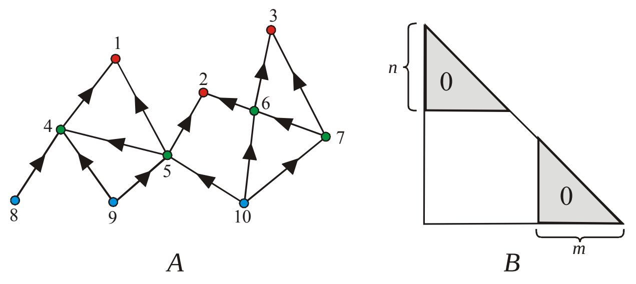

Figure 4. An illustrative example of indexing of acyclic oriented graph such that the above theorem (13) is fulfilled. An acyclic oriented graph (A) is canonically indexed, where output vertices are $V_{out} = \{1,2,3\}$, inner vertices are $V_{trans} = \{4,5,6,7\}$ and input vertices are $V_{in} = \{8,9,10\}$. Since output as well as input vertices are not mutually interacting, this implies that the left-upper and right-bottom parts of adjacency matrix (B) are empty, the ´1´ may appear only in middle unshaded part.

In order to specify Boolean function by a lower-triangle adjacency matrix $A = (A_{ij})$, we have to specify single output and inner vertices as single Boolean connectives, see Fig. 5. In general, these vertices have at least one outgoing edge and one or more incoming edges. We use binary and ternary connectives of disjunction, conjunction and exclusive disjunction (since an application of this last connective is usually very effective in simplifying syntactic trees of Boolean function). The diagonal elements of this matrix can be used for specification Boolean connectives of individual vertices. In our approach to symbolic regression we use this simple coding, unfortunately it is not binary and therefore it involves some additional effort how to properly code Boolean connectives in a binary way. Applying simple recurrent calculation in a bottom-up style, we may simply construct binary variables assigned to all vertices in syntactic tree (see Fig., 5).

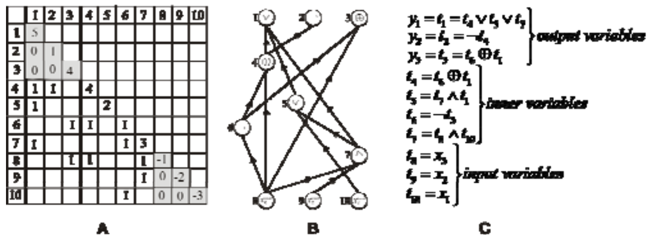

Figure 5. Diagram A represents an adjacency matrix assigned to the Boolean function specified by syntactic tree from diagram B. Lower-triangle adjacency matrix (with binary nondiagonal elements) may be generated randomly in such a way that for each column we randomly generate one, two or three unit nondiagonal elements ´1´ (this number specifies an in-valence of the given vertex). Empty diagonal elements are used for a specification of Boolean connectives that are assigned to single (inner of output) vertices. A mutation of syntactic trees coded by lower-triangle adjacency matrix consists in a simple process when a vertex is randomly

selected and then randomly changed its type (i. e. Boolean connectivity assigned to this vertex is randomly selected, this should be accompanied also by a changed type of Boolean connectivity), see Fig. 6.

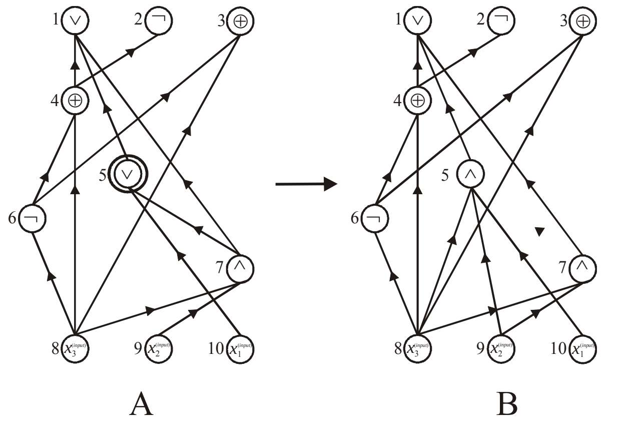

Figure 6. Illustrative example of mutation process of a syntactic tree. In diagram A an output or internal vertex is randomly selected and then in diagram B connective is mutated together with its input edges.

The present approach for symbolic regression will be illustrated by a Boolean function of the so-called full adder summator

$$\begin{array}{r} x_1 \\ x_2 \\ x_3 \\ \hline y_1\, y_2 \end{array} \tag{14}$$

where x_i and y_j are binary input/output variables. Applying simple Boolean-algebra symbolic considerations we get for (14) a Boolean function $(y_1,y_2)=f(x_1,x_2,x_3)$, its output variables as specified as follows

$$y_1=(x_1\oplus x_2)x_3+x_1x_2 \tag{15a}$$

and

$$y_2=(x_1\oplus x_2)\oplus x_3 \tag{15b}$$

We note that the Boolean function of full adder summator is specified by its simple table of its functional values. In Fig. 7 are displayed basic results for symbolic regression performed by HCwL; diagram B shows a plot of objective function, it is monotonously decreasing, starting from 125 epochs it corresponds to a global minimum 0.55 (for penalization constants ω_1=0.1 and ω_2=0.01, the resulting syntactic tree is composed of three internal vertices and ten edges, see diagram C). Diagram B displays plots of probabilities w_{ij} in the course of adaptation process, in a similar way as in the previous illustrative example; probabilities asymptotically converge either to 1 or to 0. Summarizing this illustrative example of symbolic regression of Boolean functions that are represented by acyclic syntactic trees, we see that also for this relatively complicated optimization task we are able to formulate an effective version of HCwL.

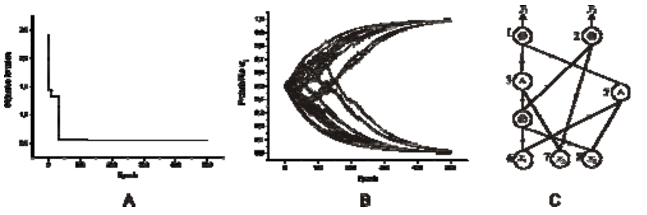

Figure 7. Illustrative examples of symbolic regression of full adder Boolean function (15). Diagram A is a plot of objective function (11) with penalization constant κ_1=0.1 and κ_2=0.01, its asymptotic value (achieved after 125 epochs) is 0.55. It corresponds to a syntactic tree displayed in diagram C. Diagram B shows plots of probabilities w_{ij}, , we see that these plots converge either to 1 or to 0.

3. CONCLUSION

In this work we have shown that the concept of collective memory, which was introduced more than half a century ago by French sociologist M. Halbwachs [4], as a fundamental specification of social groups, might be simulated by HCwL. In particular, probability vector represents a special type of collective memory, which is based on the course of adaptation process. We have demonstrated that this concept of collective memory can be used as an effective accelerator element of an adaptive system. The collective memory generalizes experiences and knowledge of agents in the history of this adaptive process. This was illustrated by simple examples of symbolic regression, where collective memory is constructed incrementally and gradually introduces some determinism to mutations in the reproductive process based on previous experience of the adaptation process.

To conclude, we emphasize our belief that the concept of Halbwachs collective memory provides effective theoretical approach for accelerating adaptations processes of HCwL, this approach allows to bridge the intergenerational transmission of the information barrier to future generations by making use cultural mechanisms. Moreover, we believe that similar approaches may be derived also for more complex system of distributed artificial intelligence (e. g. for multiagent systems composed of quasi-independent elementary units with adaptive control devices, which systems may be understood as prototypes of social systems).

The suggested enlargement of HCwL method may be understood as an acceleration mode of its adaptation process based on the history of adaptation process. In social sciences there is now very popular Halbwachs´ concept of „collective memory“, which is based on a central idea that the collective memory increases a cohesiveness of a given social group. If we compare on the one side the use of concept of collective memory in social sciences, with its use in artificial intelligence on the other side, we may observe that there exist many common properties that substantially simplify a discussion of their common interaction and interpretation. One of the main goals of this paper is to apply the general formalism of HCwL to support a novel alternative look at a meaning of concept of collective memory in social sciences, where it makes possible to overcome intergeneration “knowledge barriers” and provides an ability to predict future states of system from its history.

4. ACKNOWLEDGMENTS

This contribution was supported by the VEGA (Slovak Scientific Grant Agency) of the Ministry of Education of the Slovak Republic (ME SR) and of the Slovak Academy of Sciences (SAS) under the contract No. VEGA 1/0458/13 and VEGA 1/0553/12.

5. REFERENCES

[1] Baluja S., Caruana R.: *Removing the Genetics from the Standard Genetic Algorithm.* Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, CMU-CS-95-141.

[2] Baluja S.: *Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning.* Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, CMU-CS-94-163.

[3] Baluja, S. :Using a priori Knowledge to Create Probabilistic Models for Optimization, *International Journal of Approximate Reasoning*, Volume 31 (2002), Issue 3, pp 193-220. IJAR

[4] Halbwachs, M.: *On collective memory*, The University of Chicago Press, Chicago, IL,1992.

[5] Harary, F.: *Graph Theory*, Addison-Wesley, Boston, 1969.

[6] Koza, J.: *On the programming of computers by means of natural selection.* A Bradford book, MIT Press, Cambridge (1992)

[7] Kvasnička, V., Pelikán, M., Pospíchal, J.: *Hill Climbing with Learning. An abstraction of Genetic Algorithm*, Neural Network World, **5** (1996), 773-796.

[8] Pelikan, M., Goldberg, D.E., Lobo, F.G.: *A survey of optimization by building and using probabilistic models.* Computational optimization and applications, 21(2002), 5-20.