Liquid State Genetic Programming

Mihai Oltean

Department of Computer Science
Faculty of Mathematics and Computer Science
Babeş-Bolyai University, Kogălniceanu 1
Cluj-Napoca, 3400, Romania.
moltean@cs.ubbcluj.ro

Abstract. A new Genetic Programming variant called Liquid State Genetic Programming (LSGP) is proposed in this paper. LSGP is a hybrid method combining a dynamic memory for storing the inputs (the liquid) and a Genetic Programming technique used for the problem solving part. Several numerical experiments with LSGP are performed by using several benchmarking problems. Numerical experiments show that LSGP performs similarly and sometimes even better than standard Genetic Programming for the considered test problems.

1 Introduction

Liquid State Machine (LSM) is a technique recently described in the literature [3], [5]. It provides a theoretical framework for a neural network that can process signals presented temporally. The system is comprised of two subcomponents, the *liquid* and the *readout*. The former acts as a decaying memory, while the latter acts as the main pattern recognition unit.

Liquid State Genetic Programming (LSGP), introduced in this paper, is similar to both LSM and Genetic Programming (GP) [1] as it uses a dynamic memory (the liquid) and a GP algorithm which is the actual problem solver. The liquid is simulated by using some operations performed on the inputs. The purpose of the liquid is to transform the inputs into a form which can be more easily processed by the problem solver (GP). The liquid acts as some kind of preprocessor which combines the inputs using the standard functions available for the internal nodes of GP trees.

We have applied the LSGP on several test problems. Due to the space limitation we will present the results only for one difficult problem: even-parity. We choose to apply the proposed LSGP technique to the even-parity problems because according to Koza [1] these problems appear to be the most difficult Boolean functions to be detected via a blind random search.

Evolutionary techniques have been extensively used for evolving digital circuits [1], [4], due to their practical importance. The case of even-parity circuits was deeply analyzed [1], [4] due to their simple representation. Standard GP was able to solve up to even-5 parity [1]. Using the proposed LSGP we are able to easily solve up to even-8 parity problem.

Numerical experiments, performed in this paper, show that LSGP performs similarly and sometimes even better than standard GP for the considered test problems.

The paper is organized as follows: Liquid State Machines are briefly described in Sect. 2. In Sect. 3 the proposed Liquid State Genetic Programming technique is described. The results of the numerical experiments for are presented in Sect. 4.2. Conclusions and further work directions are given in Sect. 6.

2 Liquid State Machines - a Brief Introduction

Liquid computing is a technique recently described in the literature [3], [5]. It provides a theoretical framework for an Artificial Neural Network (ANN) that can process signals presented temporally. The system is comprised of two subcomponents, the *liquid* and the *readout*. The former acts as a decaying memory, while the latter acts as the main pattern recognition unit. To understand the system, a simple analogy is used. If a small rock thrown in a cup (pool) of water it will generate ripples that are mathematically related both to characteristics of the rock, as well as characteristics of the pool at the moment that the rock was thrown in. A camera takes still images of the water's ripple patterns. A computer then analyzes these still pictures. The result is that the computer should know something about the rock that was thrown in. For example, it should know about how long ago the rock was thrown. The rock represents a single bit from an input stream, or an action potential. The water is the liquid memory. The computer functions as the readout.

Translated into ANN's language the idea behind Liquid State Machines has been implemented [3], [5] as follows: Two ANNs are used: one of them plays the role of the liquid and the other is the actual solver. The inputs of the first network are the problem inputs and this network will reshape (modify) the inputs in order to be more easily handled by the second network. This second network, which is the actual problem solver, takes the inputs from some of the nodes (randomly chosen) of the first network. In this way, it is hopped that the structure of the second network (the actual problem solver) is simpler than the case when a single network is used for solving the problem.

3 Liquid State Genetic Programming

In this section the proposed LSGP technique is described. LSGP is a hybrid method combining a technique for manipulating the liquid and a GP technique for the individuals.

For a better understanding we will start with a short example on how a LSGP individual looks like for the even-parity problem. The example is depicted in Fig. 1.

The liquid can be viewed as a set (a pool) of items (or individuals) which are subject to some strict rules which will be deeply explained in the next sections. The liquid is simulated by using some operations performed on the inputs. The

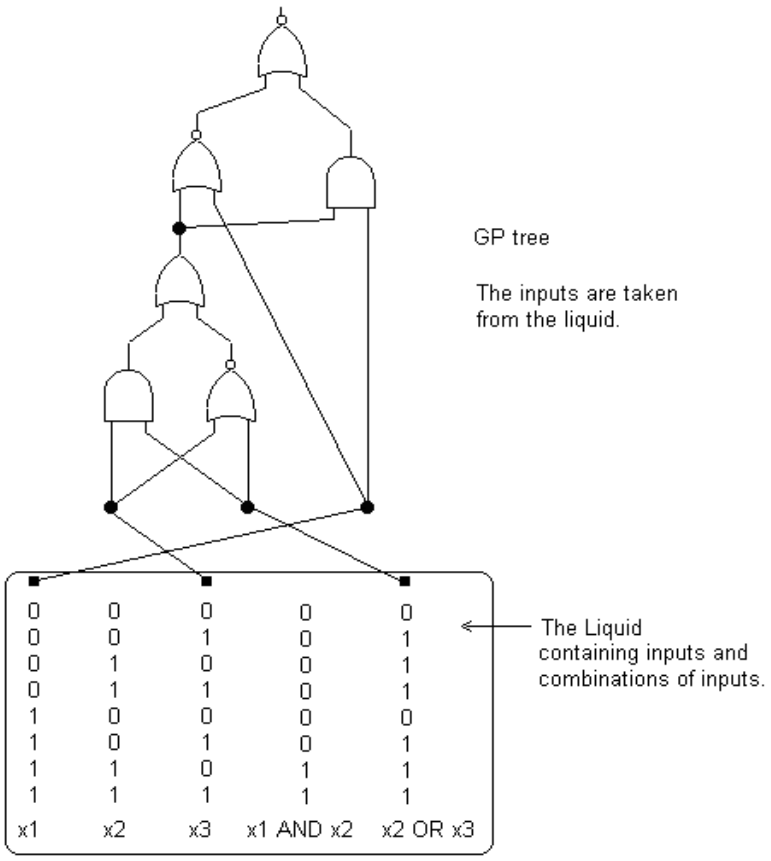

Fig. 1. A *liquid* and a GP program for the even-3-parity problem. The *liquid* contains 5 items. Three of them are the standard inputs for the even-3-parity problem and 2 of them are combinations of the standard inputs. The inputs for the GP program are taken from the *liquid*. The gates used by the GP program are the standard one: AND, OR, NAND, NOR

purpose of the *liquid* is to transform the inputs into a form which can be more easily processed by the problem solver (GP). The liquid acts as some kind of preprocessor which combines the inputs using the standard functions available for the internal nodes of GP trees.

The liquid and its accompanying rules can also be viewed as a simple GP algorithm that manipulates only the output of the tree rather than the entire tree. The state of the liquid will also evolve during the search process.

The GP algorithm is a standard one [1] and due to the space limitations will not be detailed in this paper.

3.1 Prerequisite

The quality of a GP individual is usually computed using a set of fitness cases [1]. For instance, the aim of symbolic regression is to find a mathematical expression that satisfies a set of m fitness cases.

We consider a problem with n inputs: x_1, x_2, $\ldots x_n$ and one output f. The inputs are also called terminals [1]. The function symbols that we use for constructing a mathematical expression are $F = \{+, -, *, /, sin\}$.

Each fitness case is given as an array of $(n+1)$ real values:

$$v_1^k, v_2^k, v_3^k, ..., v_n^k, f_k$$

where v_j^k is the value of the j^{th} attribute (which is x_j) in the k^{th} fitness case and f_k is the output for the k^{th} fitness case.

Usually more fitness cases are given (denoted by m) and the task is to find the expression that best satisfies all these fitness cases. This is usually done by minimizing the quantity:

$$Q = \sum_{k=1}^{m} |f_k - o_k|,$$

where f_k is the target value for the k^{th} fitness case and o_k is the actual (obtained) value for the k^{th} fitness case.

3.2 Representation of Liquid's Items

Each individual (or item) in the liquid represents a mathematical expression obtained so far, but this individual does not explicitly store this expression. Each individual in the liquid stores only the obtained value, so far, for each fitness case. Thus an individual in the liquid is an array of values:

$$(o_1, o_2, o_3, \ldots, o_m)^T,$$

where o_k is the current value for the k^{th} fitness case and $()^T$ is the notation for the transposed array. Each position in this array (a value o_k) is a gene. As we

said it before behind these values is a mathematical expression whose evaluation has generated these values. However, we do not store this expression. We store only the values o_k.

3.3 Initial Liquid

The initial liquid contains individuals (items) whose values have been generated by simple expressions (made up by a single terminal). For instance, if an individual in the initial liquid represents the expression:

$$E = x_1,$$

then the corresponding individual in the liquid is represented as:

$$C = (v_1^1, v_1^2, v_1^3, ..., v_1^m)^T$$

where v_j^k has been previously explained.

Example 1. For the particular case of the even-3-parity problem we have 3 inputs x_1, x_2, x_3 (see Fig. 1) and $2^3 = 8$ fitness cases which are listed in Table 1:

Table 1. The truth table for the even-3-parity problem

x_1	x_2	x_3	**Output**
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Each item in the liquid is an array with 8 values (one for each fitness case). There are only 3 possible items for initial liquid: $(00001111)^T$, $(00110011)^T$ and $(01010101)^T$ corresponding to the values for variables x_1, x_2 and x_3. Other items can appear in the liquid later as effect of the specific genetic operators (see Sect. 3.4). Of course, multiple copies of the same item are allowed in a liquid at any moment of time.

It is desirable, but not necessary to have each variable represented at least once in the initial liquid. This means that the number of items in the liquid should be larger than the number of inputs of the problem being solved. However, an input can be inserted later as effect of the insertion operator (see Sect. 3.4).

3.4 Operators Utilized for Modifying the Liquid

In this section the operators used for the Liquid part of the LSGP are described. Two operators are used: combination and insertion. These operators are specially designed for the liquid part of the proposed LSGP technique.

Recombination The recombination operator is the only variation operator that creates new items in the liquid. For recombination several items (the parents) and a function symbol are selected. The offspring is obtained by applying the selected operator for each of the symbols of the parents.

The number of parents selected for combination depends on the number of arguments required by the selected function symbol. Two parents have to be selected for combination if the function symbol is a binary operator. A single parent needs to be selected if the function symbol is a unary operator.

Example 2. Let us suppose that the operator AND is selected. In this case two parents (items in the liquid):

$$C_1 = (p_1, p_2, \ldots, p_m)^T and$$

$$C_2 = (q_1, q_2, \ldots, q_m)^T$$

are selected and the offspring O is obtained as follows:

$$O = (p_1 AND q_1, p_2 AND q_2, \ldots, p_m AND q_m)^T.$$

Example 3. Let us suppose that the operator NOT is selected. In this case one parent (item in the liquid):

$$C_1 = (p_1, p_2, \ldots, p_m)^T$$

is selected and the offspring O is obtained as follows:

$$O = (NOT(p_1), NOT(p_2), \ldots, NOT(p_m))^T.$$

Remark 1. The operators used for combining genes of the items in the liquid must be restricted to those used by the main GP algorithm. For instance, if the function set allowed in the main GP program is $F = \{\text{AND, OR}\}$, then for the recombination part of the liquid we can use only these 2 operators. We cannot use other functions such as NOT, XOR etc.

Insertion This operator inserts a simple expression (made up of a single terminal) in the liquid. This operator is useful when the liquid contains items representing very complex expressions that cannot improve the search. By inserting simple expressions we give a chance to the evolutionary process to choose another direction for evolution.

3.5 The LSGP Algorithm

Due to the special representation and due to the special operators, LSGP uses a special generational algorithm which is given below.

The LSGP algorithm starts by creating a random population of GP individuals and a random liquid. The evolutionary process is run for a fixed number of generations. The underlying algorithm for GP has been deeply described in [1].

The modifications in liquid are also generation-based and usually they have a different rate compared to modifications performed for the GP individuals. From the numerical experiments we have deduced that the modifications in the liquid should not occur as often as the modifications within GP individuals. Thus an update of the liquid's items will be performed only after 5 generations of the GP algorithm.

The updates for the liquid are as follows: at each generation of the liquid the following steps are repeated until the new *LiquidSize* items are obtained: with a probability p_{insert} generate an offspring made up of a single terminal (see the Insertion operator, Sect. 3.4). With a probability $1 - p_{\text{insert}}$ randomly select two parents. The parents are recombined in order to obtain an offspring (see Sect. 3.4). The offspring enters the liquid of the next generation.

A basic form of elitism is employed by the GP algorithm: the best so far GP individual along with the current state of the liquid is saved at each generation during the search process. The best individual will provide solution of the problem.

3.6 Complexity

A very important aspect of the GP techniques is the time complexity of the procedure used for computing the fitness of the newly created individuals.

The complexity of that procedure for the standard GP is $O(m * g)$, where m is the number of fitness cases and g is average number of nodes in the GP tree.

By contrast, the complexity of generating (by insertion or recombination) an individual in the liquid is only $O(m)$, because the liquid's item is generated by traversing an array of size m only once. The length of an item in the liquid is m.

Clearly, the use of the liquid could generate a small overhead of the LSGP when compared to the standard GP. Numerical experiments show (running times not presented due to the space limitation) that LSGP is faster than the standard GP, because the liquid part is considerably faster than the standard GP and many combinations performed in the liquid could lead to perfect solutions. However, it is very difficult to estimate how many generations we can run an LSGP program in order to achieve the same complexity as GP. This is why we restricted GP and LSGP to run the same number of generations.

4 Numerical Experiments

Several numerical experiments using LSGP are performed in this section by using the even-parity problem. The Boolean even-parity function of k Boolean

arguments returns **T** (**True**) if an even number of its arguments are **T**. Otherwise the even-parity function returns **NIL** (**False**) [1].

4.1 Experimental Setup

General parameter of the LSGP algorithm are given in Table 2.

Table 2. General parameters of the LSGP algorithm

Parameter	**Value**
Liquid's insertion probability	0.05
GP Selection	Binary Tournament
Terminal set for the liquid	Problem inputs
Liquid size	2 * Number of inputs
Terminal set for the GP individuals	Liquid's items
Number of GP generations before updating the liquid	5
Maximum GP tree height	12
Number of runs	100 (excepting for the even-8-parity)

For the even-parity problems we use the set of functions (for both liquid and GP trees) $F = \{\text{AND, OR, NAND, NOR}\}$ as indicated in [1].

4.2 Summarized Results

Summarized results of applying Liquid State Genetic Programming for solving the considered problem are given in Table 3.

For assessing the performance of the LSGP algorithm in the case of even-parity problems we use the success rate metric(the number of successful runs over the total number of runs).

Table 3 shows that LSGP is able to solve the even-parity problems very well. Genetic Programming without Automatically Defined Functions was able to solve instances up to even-5 parity problem within a reasonable time frame and using a reasonable population. Only 8 runs have been performed for the even-8-parity due to the excessive running time. Note again that a perfect comparison between GP and LSGP cannot be made due to their different individual representation.

Table 3 also shows that the effort required for solving the problem increases with one order of magnitude for each instance.

5 Limitations of the Proposed Approach

The main disadvantage of the proposed approach is related to the history of the liquid which is not maintained. We cannot know the origin of an item from the

Table 3. Summarized results for solving the even-parity problem using LSGP and GP. Second column indicates the population size used for solving the problem. Third column indicates the number of generations required by the algorithm. The success rate for the GP algorithms is given in the fourth column. The success rate for the LSGP algorithms is given in the fifth column

Problem	Pop size	Number of generations	GP success rate (%)	LSGP success rate (%)
even-3	100	50	42	93
even-4	1000	50	9	82
even-5	5000	50	7	66
even-6	5000	500	4	54
even-7	5000	1000	-	14
even-8	10000	2000	-	1 successful out of 8 runs

liquid unless we store it. We can also store the tree for each item in the liquid but this will lead to a considerable more memory usage and to a lower speed of the algorithm. Another possible way to have access to the liquid's history is to store the entire items ever generated within the liquid and, for each item, to store pointers to its parents. This is still not very efficient; however, it is better than storing the entire tree, because, in the second case, the subtree repetition is avoided.

6 Conclusions and Further Work

A new evolutionary technique called Liquid State Genetic Programming has been proposed in this paper. LSGP uses a special, dynamic memory for storing and manipulating the inputs.

LSGP has been used for solving several difficult problems. In the case of even-parity, the numerical experiments have shown that LSGP was able to evolve very fast a solution for up to even-8 parity problem. Note that the standard GP evolved (within a reasonable time frame) a solution for up to even-5 parity problem [1].

Further effort will be spent for improving the proposed Liquid State Genetic Programming technique. For instance, the speed of the liquid can be greatly improved by using Sub-machine Code GP [6], [7]. Many aspects of the proposed LSGP technique require further investigation: the size of the liquid, the frequency for updating the liquid, the operators used in conjunction with the liquid etc.

The proposed LSGP technique will be also used for solving other symbolic regression, classification [1] and dynamic (with inputs being variable in time) problems.

References

1. Koza, J. R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA, (1992)
2. Koza, J. R.: Genetic Programming II: Automatic Discovery of Reusable Subprograms, MIT Press, Cambridge, MA, (1994)
3. Maass, W. Natschlger, T., Markram, H.: Real-time computing without stable states: A new framework for neural computation based on perturbations, Neural Computation, Vol. 14 (2002) 2531-2560
4. Miller, J. F., Job, D., Vassilev, V. K.: Principles in the Evolutionary Design of Digital Circuits - Part I, Genetic Programming and Evolvable Machines, Vol. 1, Kluwer Academic Publishers (2000) 7-35
5. Natschlger, T., Maass, W., Markram H.: The "liquid computer": A novel strategy for real-time computing on time series. Special Issue on Foundations of Information Processing of TELEMATIK, Vol. 8, (2002) 39-43
6. Poli, R., Langdon, W. B.: Sub-machine Code Genetic Programming, in Advances in Genetic Programming 3, L. Spector, W. B. Langdon, U.-M. O'Reilly, P. J. Angeline, Eds. Cambridge:MA, MIT Press, chapter 13 (1999)
7. Poli, R., Page, J.: Solving High-Order Boolean Parity Problems with Smooth Uniform Crossover, Sub-Machine Code GP and Demes, Journal of Genetic Programming and Evolvable Machines, Kluwer (2000) 1-21
8. Prechelt, L.: PROBEN1: A Set of Neural Network Problems and Benchmarking Rules, Technical Report 21, (1994), University of Karlsruhe, (available from ftp://ftp.cs.cmu.edu/afs/cs/project/connect/bench/contrib/prechelt/proben1.tar.gz.)
9. UCI Machine Learning Repository (available from www.ics.uci.edu/∼mlearn/MLRepository.html)