Local Search with Probabilistic Modeling for Learning Multiple-Valued Logic Networks

Shangce GAO†, Student Member, Qiping CAO††, Nonmember, Masahiro ISHII†, and Zheng TANG†‡, Members

SUMMARY This paper proposes a probabilistic modeling learning algorithm for the local search approach to the Multiple-Valued Logic (MVL) networks. The learning model (PMLS) has two phases: a local search (LS) phase, and a probabilistic modeling (PM) phase. The LS performs searches by updating the parameters of the MVL network. It is equivalent to a gradient decrease of the error measures, and leads to a local minimum of error that represents a good solution to the problem. Once the LS is trapped in local minima, the PM phase attempts to generate a new starting point for LS for further search. It is expected that the further search is guided to a promising area by the probability model. Thus, the proposed algorithm can escape from local minima and further search better results. We test the algorithm on many randomly generated MVL networks. Simulation results show that the proposed algorithm is better than the other improved local search learning methods, such as stochastic dynamic local search (SDLS) and chaotic dynamic local search (CDLS).

key words: multiple-valued logic, network learning, local search, probabilistic modeling, combinatorial optimization problems

1. Introduction

Multiple-Valued Logic (MVL) has been the subject of much research over many years [1]. Besides reduction in chip area, MVL offers other benefits, such as potential for image processing [2] and speech recognition [3]. In recent years, MVL has been combined into neural networks by constructing multi-valued neural element (MVN) [4] or associative memory (AM) [5] for effectively solving image recognition problems [6]. However, the advance of MVL technology depends much on the development of devices that are inherently suitable for MVL operation. Recently work on MVL using new devices, for example, resonant tunneling transistors, resonant tunneling diodes and super-pass transistors has been reported in the literature [7]–[9]. This opens the possibility of integrating MVL in a VLSI system. MVL circuits can be designed based on the learning of the relationship that transforms inputs to outputs. With the relationship, the circuit network will yield a particular response to a specific input when given a set of input-output pairs of examples. For learning such a relationship, several MVL networks with learning capability have been proposed [10]–[12]. Focusing on the MVL network construction, Z. Tang et al. [13] showed that the network can be created based on the canonical realization of MVL functions, where a set of basic operators, either \{MAX, MIN, and Literal operator\} or \{SUM, Multiplication and Piecewise linear operation\} can be utilized. Furthermore, various methods have been proposed for learning MVL networks.

As an efficient gradient descent method, the backpropagation algorithm (BP) [14] exhibits good learning properties. Nevertheless, since the derivatives of the node functions are required in BP, and for most inputs the resultant derivatives are zero, learning cannot be performed efficiently. Moreover, the gradient-based approach used in BP suffers from slow convergence and being entrapped at local minima in the parameter space [15]. Considering the limitations of BP, many researchers have found genetic algorithm (GA) [16] to be a better candidate [17]–[19]. Unlike the BP, the objective function for GA need not be differentiable or even continuous since it based on the mechanics of nature genetics which is capable of rapidly locating near-optimal solution to difficult problems [20]. Nevertheless, the efficiency of GA is restricted by premature convergence and poor ability of fine-tuning near local optimum points [21]. Other GA approaches have also encountered problems such as initial numbers of MVL neurons were obtained by trial and error, the initial values in the population were generated randomly according to a fixed range and unable to use prior knowledge before learning as reported in Tang’s work [12]. These drawbacks induced by the two of the most common approaches in literature only goes to show the need for a different and more simple learning algorithm to overcome the restrictions and limitations for parameter tuning.

The idea of local search has been extensively explored for many years. Most recently, Cao et al. [22] made use of local search (LS) for training MVL networks. LS is a direct and simple learning method which constructs a layered MVL network based on canonical realization of MVL functions, defines an error measure between the actual output value and teacher’s value and updates randomly selected parameters of the MVL network if and only if the updating results in a decrease of the error measure. Unlike the GA, LS does not have complicated evolutionary operators such as the crossover and mutation. It is simpler to be implemented because there is no tedious calculation or complicated process. The memory capacity is also smaller. Even though, the inherent local minima problem restricts its efficiency. Some concerns regarding the solving of this problem have been proposed. By introducing stochastic or chaotic dynamics into LS, the improved algorithms SDLS [23] or CDLS [24]...
were able to prevent trapping local minima and find global minimum eventually.

Over the last decades, there has been an ever increasing interest in the area of the probabilistic modeling (PM) [25], aiming at overcoming the drawbacks of usual recombination operators of evolutionary computation algorithms. This class of algorithms is also known as estimation of distribution algorithms [26]. There are several PMs in the literature [27]–[34] with alternative probabilistic models, depending on the intended degree of relationship among the variables. These algorithms, which have a theoretical foundation in probability theory, are also based on populations that evolve as the search progresses. The PMs attempt to model the distribution of promising solutions and then produce the next generation by sampling the estimated distribution modeling. After every iteration, the distribution is reestimated.

In this paper, the idea of PM is incorporated into LS to overcome the local minimum problem of LS, and the proposed PMLS is proposed for learning MVL networks. In contrast to the SDLS and CDLS, the proposed algorithm does not introduce the stochastic dynamic or chaotic dynamic into the original LS, and therefore does not attempt to prevent the system getting stuck at local minima. Rather, once LS converges to a local minimum, it aims to generate a new better starting point by PM for further LS search which is expected to be guided to a promising area. As a result, the proposed PMLS can escape from local minima and further search better results. The performance of the PMLS is tested and evaluated by simulating a large number of MVL networks. Simulation results demonstrate the efficiency of PMLS when compared with SDLS and CDLS.

The contributions of this paper are as follows: (1) a novel competitive LS method combined with PM is proposed for learning MVL networks and very good results are obtained, which is a contribution to the available literature on the MVL; (2) the PM which essentially is a population-based and multipoint search method is applied to the single-point search method–LS and thus helps LS escape from local minima, which is the contribution to both PM and LS literatures from the methodological perspective; and (3) an experimental comparison of the proposed method with other local minimum problem solving techniques, such as SDLS and CDLS, is also provided.

The remainder of the paper is organized as follows. In Sect. 2, we describe the LS for learning MVL networks. In Sect. 3, we propose the LS combined with PM for solving its inherent local minimum problem. Then we validate the proposed PMLS model by applying it to learn a large number of MVL networks in Sect. 4. Finally some general remarks are given to conclude the paper.

2. LS for Learning MVL Networks

The MVL network [22] is a three-layered feed-forward network as shown in Fig. 1. It is constructed based on the Allen–Givone Algebra [35] where a set of operators \( \{\text{MAX}, \text{MIN}, \text{Literal operator}\} \) are adopted. The network can be described as follows. The input layer in the network represents the variables from the set \( X = \{x_1, x_2, \ldots, x_n\} \) of a MVL function, and each of which can only take on values from \( R = [0, 1, \ldots, r - 1] \). The first layer in MVL network has \( n \times m \) nodes, where \( n \) denotes the number of variables, while \( m \) reflects the number of nodes in the second layer. Each node in the first layer implements a literal operator, that is:

\[
x_i(a_{ij}, b_{ij}) = \begin{cases} 
    r - 1 & a_{ij} \leq x_i \leq b_{ij} \\
    0 & \text{otherwise} 
\end{cases}
\]  

where \( x_i(a_{ij}, b_{ij}) \) \((i = 1, 2, \ldots, n)\) is the output of the node associated with the variable \( x_i \), \( a_{ij} \) and \( b_{ij} \) are window parameters assuring that \( a_{ij}, b_{ij} \in [0, 1, 2, \ldots, r - 1] \) and \( a_{ij} \leq b_{ij} \) for \( i = 1, 2, \ldots, n \) and \( j = 1, 2, \ldots, m \). As the values of \( a_{ij} \) and \( b_{ij} \) change, the literal function varies accordingly, thus exhibiting various forms of literal functions. The node function used in the second layer is the \( \text{MIN} \) operator, defined as:

\[
\text{MIN}_j = \text{MIN}(c_j, x_1(a_{1j}, b_{1j}), \ldots, x_n(a_{nj}, b_{nj})) 
\]

\[
= c_j \land x_1(a_{1j}, b_{1j}) \land \ldots \land x_n(a_{nj}, b_{nj}) 
\]  

where \( \text{MIN}_j \) \((j = 1, 2, \ldots, m)\) represents the output of the \( j \)-th node in the second layer. The operator \( \text{MIN} (\land) \) returns the smallest value of its arguments. \( c_j \) is a biasing parameter of \( \text{MIN}_j \). In the third layer, all outputs from the former layer are taken as the inputs of a \( \text{MAX} (\lor) \) operator as shown in the following:

\[
O = \text{MAX}(\text{MIN}_1, \text{MIN}_2, \ldots, \text{MIN}_m) 
\]

\[
= \text{MIN}_1 \lor \text{MIN}_2 \lor \ldots \lor \text{MIN}_m 
\]  

The \( \text{MAX} (\lor) \) operator returns the largest value of its arguments. Meanwhile, the resultant value \( O \) is transferred to the next layer as the final output of the network. For a more
comprehensive interpretation on the MVL network, refer to [22].

Since the network incorporated with the MAX, MIN and literal nodes gives functional completeness [22], any MVL function can be realized in the network by adjusting the system parameters:

\[ V = [c_1, a_{11}, b_{11}, a_{21}, b_{21}, \ldots, a_{1n}, b_{1n}, \ldots, c_m, a_{1m}, b_{1m}, a_{2m}, b_{2m}, \ldots, a_{mn}, b_{mn}]^T \]  \hfill (4)

As a single search technique, local search (LS) has shown its ability in coping with this task [22]. It starts from an initial solution, \( v \) and repeats replacing \( v \) with a better solution in the neighborhood until no better solution is found. The neighborhood is a set of solutions obtained from \( v \) with a slight perturbation. By interactively adjusting \( V \) through the search, we are able to minimize the error. The error for MVL network learning can be defined as the difference between the actual system output and the expected output:

\[ E = \sum_p (O_p - T_p)^2 \]  \hfill (5)

where \( O_p \) and \( T_p \) represent the \( p \)-th actual output value of the network and the teacher’s value corresponding to the \( p \)-th input pattern \((x_1, x_2, \ldots, x_n)_p\), respectively. Generally, the teacher’s values \( T \) for all input patterns are given by the MVL truth-table. In LS, the search direction it takes can be either one of the many \( L = m(2n+1) \) directions which represents the number of elements in the neighborhood, \( N \). Here, \( L \) denotes the total length of the system parameter vector \( V \). The direction vector \( e'_{i_k} \) at \( k \) iteration can be defined as such:

\[ e'_{i_k} = (0, 0, \ldots, 0, 1, 0, \ldots, 0)^T \]  \hfill (6)

Based on this understanding, a sequence of iterations can be generated. For \( k = 0 \), the iterations are initiated with \( V_k \), then moves along the direction,

\[ E(V_k + \Delta_k e'_{i_k}) < E(V_k) \]  \hfill (7)

\[ E(V_k - \Delta_k e'_{i_k}) < E(V_k) \]  \hfill (8)

When such a point is found, the iteration is declared successful, and the subsequent can be defined as

\[ V_{k+1} = V_k + \Delta_k e'_{i_k} \]  \hfill (9)

\[ V_{k+1} = V_k - \Delta_k e'_{i_k} \]  \hfill (10)

The step size \( \Delta_k \) is generally set as 1 for all \( k \). On the contrary, the iteration can be termed as unsuccessful if no such point is found. Instead local search performs forward and the next iteration would be the same as the current point,

\[ V_{k+1} = V_k \]  \hfill (11)

The essential advantages of the local search technique are its simplicity and easy adaptability for most problems. The shorter computation time of this learning is mainly attributed by the methodology of having to use only the local information to decide the next move.

The LS can thus be summarized as follows.

<table>
<thead>
<tr>
<th>Algorithm 1 – LS for MVL Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Begin procedure</td>
</tr>
<tr>
<td>initialize ( V_k ); set ( k = 0 );</td>
</tr>
<tr>
<td>calculate ( E ) using equations (1)–(3) and (5);</td>
</tr>
<tr>
<td>while ( k &gt; \text{iterations} ) or stability criteria are satisfied do</td>
</tr>
<tr>
<td>for ( l := 1 ) to ( L ) do</td>
</tr>
<tr>
<td>perturbation ( V_i ) in the neighborhood using (7)–(8);</td>
</tr>
<tr>
<td>calculate ( E ) using (1)–(3) and (5);</td>
</tr>
<tr>
<td>update ( V_{k+1} ) using (9)–(11);</td>
</tr>
<tr>
<td>end-for</td>
</tr>
<tr>
<td>set ( k = k + 1 );</td>
</tr>
<tr>
<td>end-while</td>
</tr>
<tr>
<td>end-procedure</td>
</tr>
</tbody>
</table>

It was observed [22] that the LS described above can make significant advances in the earlier stage of learning procedures, decreasing the error function rapidly and dramatically. However, with the implementation goes, the improvements of solutions found by LS become slower and slower, and after several iterations, a local minimum is encountered, suggesting that the error function no longer decreases. The local minimum problem is caused by the essential gradient descent dynamics of the update rule of the LS.

3. PMLS for Learning MVL Networks

Aiming at solving the inherent local minimum problem of LS and improving its searching performance, a perturbation operator is applied to the local minimum to generate a new starting point for the LS when it is trapped into a local minimum. It is desirable that the generated starting point should be in a promising area in the search space to guide LS for further search. Based on this consideration, in this section, we propose a PM mutation operator as the perturbation operator in the LS. Further explanations are given in the followings.

3.1 A Brief Introduction to PM

PM is a new area of evolutionary computation, signaling a paradigm shift in genetic and evolutionary computation research [25],[26]. Incorporating (automated) linkage learning techniques into a graphical probabilistic model, PM exploits a feasible probabilistic model built around superior solutions found thus far while efficiently traversing the search space [25]. Thus, it has a theoretical foundation in probability theory and serves as a population-based search tool. An algorithmic framework of most PMs can be described as:
Framework of PM

\[ Pop = \text{InitializePopulation();} \quad /*\text{Initialization}*/ \]

\[ \text{while} \text{ Stopping criteria are not satisfied} \quad \text{do} \]
\[ Pop_{sel} = \text{Select}(Pop); \quad /*\text{Selection}*/ \]
\[ \text{Prob} = \text{Estimate}(Pop_{sel}); \quad /*\text{Estimation}*/ \]
\[ Pop = \text{Sample(Prob);} \quad /*\text{Sampling}*/ \]
\[ \text{end-while} \]

At the beginning of the PM search, a solution population \( Pop \) and a solution distribution model \( Prob \) is initialized, and this is followed by a main search loop, consisting of three principal stages. The first stage is to select some best individuals from \( Pop \), according to the fitness criteria. These individuals are used in the second stage in which the solution distribution model \( Prob \) is updated or recreated. The third stage consists of sampling the updated solution distribution model to generate new solution offspring. These new solutions are evaluated and incorporated into the original population, replacing some or all of the old ones. This process is repeated until the termination criterion is met.

3.2 LS Combined with PM

The PM is originally introduced to evolutionary algorithms which are population-based search methods. In this paper, the PM is introduced to the LS which is a single-point search method. The LS based on the PM (PMLS) can be summarized by the follows.

Algorithm 2: PMLS for MVL Network

\text{begin-procedure}
\begin{align*}
\text{initialize } V_k; \quad &s \text{et } k = 0; \quad \text{initialize best-so-far solution } gb \text{ and probability model } P; \\
\text{calculate } E \text{ using equations (1)–(3) and (5);} \\
\text{for } r := 1 \text{ to } \text{descents} \text{ do} \\
\text{while}(k > \text{iterations} \text{ or stability criteria are satisfied}) \\
\text{for } l := 1 \text{ to } L \\
\text{perturbation } V_k \text{ in the neighborhood using (7)–(8);} \\
\text{calculate } E \text{ using (1)–(3) and (5);} \\
\text{update } V_{k+1} \text{ using (9)–(11);} \\
\text{end-for} \\
\text{set } k = k + 1; \\
\text{end-while} \\
\text{update and keep track for the best-so-far solution } gb; \\
\text{update } P \text{ using current local optima } V_k; \\
\text{reset } P \text{ to } 1 - P \text{ after a specified number of descents; perturb } gb \text{ using PM mutation to generate a new starting point } V_k^* \text{ for the LS;} \\
\text{end-for} \\
\text{end-procedure} 
\end{align*}

The proposed algorithm PMLS repeatedly invokes two procedures: LS and PM. The LS is the essential learning engine for MVL networks, adjusting the system parameters iteratively, meanwhile minimizing the error function along with the set of decent direction, and finally leading the network to a local minimum. On the other hand, the PM incorporated into LS performs a local-optimal-problem solving technique, constructing a good starting solution for further LS search.

In Algorithm 2, the parameter \( \text{descents} \) can be seen as an upper bound on the number of LS invocations and then can be seen as a stopping criterion. One LS search iteration is defined as one complete search for all \( L \) directions according to Eqs. (1)–(3), (5), and (7)–(11). The total iteration number of the proposed algorithm is maximally \( \text{descents} \times \text{iterations} \). In PMLS, the implementation of LS is the same as that described in Section 2, while the main components of PM are depicted in the followings.

3.2.1 Representation of PM

In this paper, the univariate marginal distribution (UMD) which is one of the famous probability models [27]–[29] is adopted to estimate the distribution of promising regions over the search based on the local optimal solutions, given by:

\[
P = \begin{pmatrix}
  p_{11} & p_{12} & \ldots & p_{1r} \\
  p_{21} & p_{22} & \ldots & p_{2r} \\
  \vdots & \vdots & \ddots & \vdots \\
  p_{L1} & p_{L2} & \ldots & p_{Lr}
\end{pmatrix}
\]

(12)

where \( P \) is the probability matrix, characterizing the distribution of promising solutions in the search space. \( p_{ij} \) represents the probability that the value of the \( i \)-th element in the selected solution \( V_k \) is \( j - 1 \). \( L \) and \( r \) are defined as in the former section.

3.2.2 Initialization of PM

Initially, \( P \) is set as:

\[
  p_{ij} = 1/r, \quad \text{for } i = 1, 2, \ldots, L; \quad j = 1, 2, \ldots, r
\]

(13)

Each element in \( P \) has the same value, revealing that the probability model has no guiding information for initial search. Nevertheless, if there is a prior knowledge of the distribution of promising solutions in the search space, \( P \) can be initialized by such a knowledge to bias towards promising areas.

3.2.3 Update of Probability Matrix \( P \)

Along with the implementation of LS, a number of local optimal solutions will be visited after the descents of LS. Statistical information of these optimal solutions can be extracted for updating the probability model. The updating of \( P \) occurs at each descent of the LS in the same way as in the population-based incremental learning algorithm [27].

Assuming the current local optima is \( V_k^* = \{ c_{1}^*, \ldots, c_{m}^*, a_{11}^*, \ldots, a_{nm}^*, b_{11}^*, \ldots, b_{nm}^* \}^T \) and the current probability matrix is \( P = (p_{ij})_{L \times r} \). The updating rule can thus be defined as:

\[
p_{ij} = (1 - \lambda)p_{ij} + \lambda I_{ij}(V_k^*)
\]

(14)
where $\lambda$ ($0 \leq \lambda \leq 1$) is a learning rate. The bigger the $\lambda$ is, the greater the contribution of the current local optimal solution $V_k^*$ to the probability matrix $P$. And,

$$I_{ij}(V_k^*) = \begin{cases} 1 & \text{if } V_k^*(i) = j - 1 \\ 0 & \text{Otherwise} \end{cases} \quad (15)$$

where $V_k^*(i)$ denotes the $i$-th element in $V_k^*$.

Then a normalization process of the probability matrix takes place:

$$p_{ij} := \frac{p_{ij}}{\sum_{j=1}^{r} p_{ij}} \quad i = 1, 2, \ldots, L; j = 1, 2, \ldots, r \quad (16)$$

Based on Eqs. (14)–(16), we can easily find that the current (updated) probability matrix is equal to the sum of the fraction of its previous matrix and the contribution of the current local optimal solution. Therefore, the updating procedure not only makes the current matrix track the accumulated search directions based on the previous matrices, but also brings it toward the current optimal solution.

3.2.4 PM Perturbation

Once the LS has trapped in a local minima, a PM mutation perturbs the best-so-far solution $gb$ based on the current probability matrix $P$, which characterizes distribution of promising solutions. It can be expected that the resultant offspring falls in or close to a promising area in the search space. Algorithm 3 shown in the following describes the PM perturbation, where the function `rand()` returns a random number distributed uniformly on $[0,1]$.

```
Algorithm 3– PM Perturbation in LS
begin-procedure
Set $U = \{1, 2, \ldots, r\}$;
Input: the best-so-far solution $gb = (gb(1), \ldots, gb(L))$, the current probability matrix $P = (p_{ij})_{r \times r}$, and a positive parameter $\beta \leq 1.0$;
Output: a resultant solution $V_k^r = (V_k^r(1), \ldots, V_k^r(L))$;
for $i := 1$ to $L$ do
    If $\text{rand()} < \beta$ /*Sample from probability matrix */
        randomly draw a $j \in U$ from $U$ with probability: $p_{ij}$;
        set $V_k^r(i) = j - 1$;
        Else
            set $V_k^r(i) = gb(i)$; /*Copy from $gb$ */
end-procedure
```

In the PM perturbation process, each bit in the resultant solution $V_k^r$ (as a new starting point for the further search by LS) is sampled from the probability matrix $P$ randomly or directly copied from the best solution found so far, which is controlled or balanced by the parameter $\beta$. The larger $\beta$ is, the more bits of the new starting point are sampled from $P$.

The justification of the use of PM perturbation in LS is manifold. First and foremost, since part of the best-so-far solution $gb$ may already be close to optimal, the new starting point $V_k^r$, part of which inherits from $gb$, can thus keep the characteristics of the local minimum. As a result, the LS implemented afterwards requires only a few steps to reach the next local optimum, revealing that new local optima can be identified very fast, typically much faster than when starting from a randomly generated solution. Besides, as some bits of $V_k^r$ are also sampled from the probability matrix $P$, it can be expected that the mutated solution $V_k^r$ can jump out of the current local minimum and be guided to a promising area. Last but not least, the random sampling mechanism in PM also provides diversity for the search afterwards.

Generally, the PM perturbation thus provides a mechanism for combining global statistical information about the search space (from $P$) and the information of the best solution found during the previous search (from $gb$). Therefore, it can perturb the current best-so-far solution to generate a good new starting point for further search.

3.2.5 Restarting Strategy

In PMLS, if the best-so-far solution $gb$ can not be improved for a successive number of descents (generally be set as five descents in this paper), a restarting strategy takes place. Since PMLS has intensively exploited the current area, we should generate a new starting point which is as far from the previous search areas as possible for further search.

Compared to the commonly-used random restart strategy, a better idea is to sample new solutions which are far from the current searching area by means of resetting $P$ to $1 - P$ in PM. By doing so, we are able to utilize some information from the previous search and thus able to avoid overlapping search in previously searched areas.

All in all, the PM incorporated in LS also consists of three main processes: selection, estimation, and sampling, like other PM based algorithms. The local optimal solution encountered by LS after each descent procedure is selected; then the selected solution is used in the estimation stage where the probability model is updated or reset; finally a PM perturbation samples a new solution for further LS searching.

However, there are several differences between the PMLS and the most usual population-based PM algorithms. Compared with other pure PM algorithms, it can be observed in the PMLS that: first, only one solution is selected and estimated to update the probability matrix; second, only one solution is sampled from the probability model; and third, the probability model is not sampled directly as in the pure PM but rather used as part of the guided mutation in [30].

4. Simulation Results and Discussions

In order to assess the validity and performance of the proposed algorithm, we presented in this section the experimental results by applying it to learn several MVL networks. The instances of MVL networks used in our simulation were summarized in Table 1, where the information data we recorded for each instance were the instance name,
the number of variable, the number of logic value, the number of non-zero minterms, and the sum of the values of non-zero minterms. Considering MVL2,4,11 which was taken from the literature [22] as an illustration, the corresponding truth-table (also used as the target values for learning in MVL networks) was shown in Table 2.

In this study, we first analyzed the effects of two important parameters used in the probability model: α and β. Next, by depicting the learning results after the specific iterations, for example the initial solution, and the first encountered local minima, we characterized the learning properties of the proposed algorithm. In addition to these, further considerations dealt with the performance comparison during the proposed PMLS algorithm, one of the variations by altering PM with random initializations in the PMLS, and two other improved LS-based algorithms. All simulations were conducted in Visual Studio 2005 environment running on an ordinary personal computer with Pentium(R) 4 CPU 2.80 GHz and 512-MB memory.

4.1 Effects of λ and β

Two important parameters, λ and β, should be carefully tuned in the PMLS. λ is the learning rate used in the update of the probability model, balancing the contributions between the old statistical information extracted from historical local minima and the information of the current local minimum to the new probability matrix. The bigger the λ is, the greater the contribution of the current local minimum is. As for the parameter β, it controls the strength of the PM perturbation, in effect, adjusting the tradeoff between the global information and location information of solutions found so far. As a result, the bigger the parameter β is, the further the guided solution is from the current best-so-far solution. That is to say, the PM perturbation will be adequately able to guide the search to jump out of the local minima.

In the experiment, apart from α and β, the user-defined parameters were set as in the follows. The number of neurons in the second layer (i.e. m) in MVL network was set as the total number of minterms in the truth-table. The maximal number of descents of PMLS was twenty and each descent was allowed to have 1000 iterations. Besides, the stability criteria for each descent were defined as that when the current solution could not be improved for sequential 50 iterations. The restart strategy occurred when the best-so-far solution had remained the same for successive five descents.

The preliminary simulation dealing with the parameter sensitivity analysis for λ and β was tested on the instance MVL2,4,11. The available values for each parameter were divided into ten discrete domains, from 0.0 to 1.0 with an interval of 0.1 between every two successive values. Then, for each combination of them, the PMLS was implemented thirty times. The average errors of the best-so-far solutions were summarized in Table 3. For an intuitive estimation, the resulting surface of these results was illustrated in Fig. 2.

As can be observed from Table 3, large values for β resulted in better average qualities of final solutions. In particular, β = 0.1 performed much better than β = 0.0, implying that sampling from the probability model did contribute positively to the performance of the algorithm. However, β = 1.0 was slightly worse than β = 0.9, thus suggesting that without the corporation of the global information obtained from the best-so-far solution, the algorithm could not achieve the best performance. The following analysis dealt with the sensitivity of λ. One point worth emphasizing was that when β = 0.0, according to the Algorithm 3, λ was never used in the algorithm, i.e., it was useless for adjusting the algorithm. As a result, when β = 0.0, different

![Fig. 2 Surface of the average errors of best-so-far solutions found in thirty runs on MVL2,4,11.](image-url)
Table 3 Parameter sensitivity analysis for $\beta$ and $\lambda$: the average errors of best-so-far solutions found in thirty runs on MVL$_{2,4,11}$, where “average1” denotes the average value of solutions for each $\beta$ when $\lambda \in [0.0, 1.0]$; and “average2” denotes the average value of solutions for each $\lambda$ when $\beta \in [0.1, 1.0]$.

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>0.0</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1.0</th>
<th>average2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>4.97</td>
<td>2.77</td>
<td>2.63</td>
<td>2.73</td>
<td>2.55</td>
<td>2.33</td>
<td>2.63</td>
<td>2.17</td>
<td>2.13</td>
<td>1.83</td>
<td>1.97</td>
<td>2.63</td>
</tr>
<tr>
<td>0.1</td>
<td>5.50</td>
<td>3.23</td>
<td>2.77</td>
<td>2.57</td>
<td>2.38</td>
<td>2.43</td>
<td>2.47</td>
<td>2.00</td>
<td>2.03</td>
<td>1.67</td>
<td>2.07</td>
<td>2.67</td>
</tr>
<tr>
<td>0.2</td>
<td>5.47</td>
<td>3.17</td>
<td>2.53</td>
<td>2.47</td>
<td>2.33</td>
<td>2.43</td>
<td>2.40</td>
<td>2.45</td>
<td>2.30</td>
<td>1.97</td>
<td>2.13</td>
<td>2.59</td>
</tr>
<tr>
<td>0.3</td>
<td>4.64</td>
<td>3.00</td>
<td>2.57</td>
<td>2.27</td>
<td>2.27</td>
<td>2.07</td>
<td>2.30</td>
<td>2.33</td>
<td>1.90</td>
<td>1.70</td>
<td>2.23</td>
<td>2.58</td>
</tr>
<tr>
<td>0.4</td>
<td>5.03</td>
<td>3.27</td>
<td>2.80</td>
<td>2.23</td>
<td>2.93</td>
<td>2.50</td>
<td>2.33</td>
<td>2.30</td>
<td>2.13</td>
<td>1.93</td>
<td>1.93</td>
<td>2.63</td>
</tr>
<tr>
<td>0.5</td>
<td>6.17</td>
<td>2.97</td>
<td>2.53</td>
<td>2.30</td>
<td>2.23</td>
<td>2.37</td>
<td>2.17</td>
<td>2.33</td>
<td>1.83</td>
<td>1.73</td>
<td>1.80</td>
<td>2.53</td>
</tr>
<tr>
<td>0.6</td>
<td>5.40</td>
<td>3.00</td>
<td>2.60</td>
<td>2.17</td>
<td>2.10</td>
<td>2.50</td>
<td>2.17</td>
<td>2.00</td>
<td>1.80</td>
<td>1.60</td>
<td>1.80</td>
<td>2.50</td>
</tr>
<tr>
<td>0.7</td>
<td>5.77</td>
<td>2.83</td>
<td>2.67</td>
<td>2.55</td>
<td>2.45</td>
<td>2.23</td>
<td>2.35</td>
<td>1.87</td>
<td>1.65</td>
<td>1.75</td>
<td>1.87</td>
<td>2.46</td>
</tr>
<tr>
<td>0.8</td>
<td>5.43</td>
<td>2.67</td>
<td>2.57</td>
<td>2.43</td>
<td>2.25</td>
<td>1.90</td>
<td>2.03</td>
<td>1.83</td>
<td>1.87</td>
<td>1.50</td>
<td>1.77</td>
<td>2.36</td>
</tr>
<tr>
<td>0.9</td>
<td>4.87</td>
<td>2.47</td>
<td>2.23</td>
<td>2.43</td>
<td>1.90</td>
<td>2.10</td>
<td>1.90</td>
<td>2.20</td>
<td>2.10</td>
<td>1.63</td>
<td>1.90</td>
<td>2.46</td>
</tr>
<tr>
<td>1.0</td>
<td>5.61</td>
<td>2.93</td>
<td>2.23</td>
<td>2.23</td>
<td>2.57</td>
<td>2.27</td>
<td>2.07</td>
<td>2.13</td>
<td>2.20</td>
<td>2.03</td>
<td>2.00</td>
<td>2.56</td>
</tr>
<tr>
<td>average1</td>
<td>5.35</td>
<td>2.94</td>
<td>2.56</td>
<td>2.40</td>
<td>2.35</td>
<td>2.28</td>
<td>2.25</td>
<td>2.15</td>
<td>1.99</td>
<td>1.76</td>
<td>1.95</td>
<td></td>
</tr>
<tr>
<td>average2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 3 Learning characteristics of the PMLS on MVL$_{2,4,11}$: Error versus Iteration.

values of lambda did not make sense, the values of solutions only depended on the initial values of the system parameter $V$. That is to say, those values of solutions obtained were somewhat random. Therefore, the “average2” was defined when $\beta$ took values from $[0.1, 1.0]$. From Table 3, we found that better solutions can be acquired when $\lambda \in [0.6, 0.7, 0.8, 0.9]$. This indicated that both the global statistical information collected from the past search and the information of the current local minimum were beneficial to the algorithm. Generally, an appropriate combination for the two parameters was set $\lambda = 0.8, \beta = 0.9$.

It was worth pointing out that similar conclusions can be drawn from other MVL instances, although those results were not given in this paper.

4.2 Learning Characteristics

To highlight the significant features of the probability model to the local search, a typical learning curve on the instance MVL$_{2,4,11}$ was outlined in Fig. 3, where the horizontal axis denoted the number of iterations of the search, while the vertical axis represented the error defined in Eq. (5). The bold line indicated the best-so-far solutions found by the PMLS. From Fig. 3, it was clear that once the local search trapped into a local minimum, the probability modeling can guide the search to a new starting point which was expected to be a promising solution for further search. Although it could not be guaranteed that the following search would always find a better solution than the current one, it was evident that the accumulated search information in the past ultimately took effect through the probability model.

Specifically, four typical MVL networks conducted after learning were depicted in Fig. 4. Initially, the system parameters $V$ in Eq. (4) were randomly generated in the uniform domain of $R$ as illustrated in Fig. 4(a). Here, a point worth emphasizing was that those neurons whose biasing parameters $c_j = 0$ or the window parameters $a_{ij} > b_{ij}$ did not contribute to the output and thus can be delete from the network, represented by the symbol $\times$ over the joint line. Considering that the solution remained unimproved from the 84-th iteration to the 134-th iteration, learning could be regarded as being trapped into a local minimum (Fig. 4(b)). Profiting from both the current best-so-far solution and the past search experience which was collected by the probability modeling, a new starting point was generated to help the search jump out of the local minima, as shown in Fig. 4(c). Without a doubt, the new generated starting point could be expected to be within a promising area [26], [30], especially to be better than the randomly generated starting points. Consequently, at the 12-th descent a global optimal solution shown in Fig. 4(d) was found. More details referring to the actual solutions in Fig. 4, see Appendix.

4.3 Performance Evaluation

In order to evaluate the performance the PMLS, we first compared the PMLS with its variation algorithm RLS. The difference between them was the new starting point generating strategy. In the RLS, a random mechanism rather than the PM was used to generate a new starting point for further LS search. By doing so, we can verify the effect that PM taken on LS. For both PMLS and LS, exactly the same tun-
able parameters were used. More specially, for 2-variable 4-valued and 4-variable 4-valued instances, the stability criterion was defined as that when the current solution could not be improved for sequential 50 iterations, while for 2-variable 16-valued instances, it was set as 500 iterations.

Table 4 showed the best (Best), averages (Av.) results, standard deviation (Std.), average total iterations (Ati.) used in each run, and average computation time in seconds (Time) over 50 runs on 15 benchmark problems produced by PMLS and RLS. From Table 4, we can find that the PMLS produced better results for all tested instances, especially for larger instances, in terms of the best and average solutions. Further, the PMLS always required less total iterations to terminate the algorithm, which indicated a fast convergence speed to identify a new local minimum after jumped out the former local minimum. This result also suggested that LS can benefit from the PM by utilizing the statistical information of the past search and the information of global solution. The computational time between the PMLS and LS were almost the same, revealing that the additional computation cost for the calculation of PM can be compensated by less number of iterations that needed.

Further considerations dealt with the statistically significant between LS and PMLS. The two-sample assuming unequal variances t-test results at the 0.05 critical P level (also called the alpha level) were also presented in Table 4 for the alternative hypothesis that the average quality of solutions obtained by PMLS was better than that obtained by RLS. From the two-tailed p values derived from the t-tests, we can find that the difference of the average solution values between the two algorithms for all instances were significant by rejecting the null hypothesis (p < 0.05) except for the instance MVL2-d6.

Then, we compared the PMLS with two other LS-based algorithms, SDLS [23] and CDLS [24] for five typical MVL instances. The merit of the SDLS was the usage of a stochastic dynamic mechanism in the LS learning process. It permitted the error to be temporarily increased in the earlier search phase and then performed the original LS in the later search phase. Similarly, by using the chaotic properties, such as ergodicity, a chaotic dynamic mechanism was incorporated in the earlier search stage of CDLS, and then an annealing strategy made the chaotic dynamic disappear to stabilize the search. Both SDLS and CDLS were tried to prevent the search from being trapped into a local minimum, and to give more chances (only in the earlier search phase) to
guide the search to a global minimum. Obviously, this kind of guidance was somewhat “blind.” A direct but better idea was that when the LS trapped, we tried to flip it and gave it an additional “energy” to jump out the current local minima. Based on this consideration, the PM was performed as the “energy.”

Table 5 showed the comparison results between the original LS and its improved algorithms involving the SDLS, CDLS and PMLS. All algorithms were implemented fifty runs independently. The results that we recorded for each instance were the ratio in the reduction of the average error after performing a given number of iterations. The ratio was calculated as:

$$RATIO = \frac{E - E_{LS}}{E_{LS}}$$  

where the $E_{LS}$ was the final solutions found by the original LS, while $E$ indicated the current compared algorithm which was selected from the SDLS, CDLS and PMLS. From Table 5, we can give some remarks as shown in the following. Within small number of iterations (such as $1 \times 10^2$), the SDLS always performed worse than LS, which indicated that the SDLS was strongly influenced by the temporary error increasing mechanism in the stochastic dynamic. Contrarily the CDLS can generate the best solutions in earlier phase due to its pseudo-random and ergodic property. Although the PMLS can find better solutions than the LS in the earlier search phase, the differences were obscure. Different from the SDLS and CDLS, the PMLS made no effect on the local search process before it trapped into a local minimum. Nevertheless, when more iterations were implemented, the errors produced by the PMLS reduced drastically, while the errors produced by the LS, SDLS and CDLS reduced slowly. The PMLS can find the best solutions than the other algorithms by carrying out a large number of iterations (such as $1 \times 10^3$ or $3 \times 10^3$), which indicated that the PM indeed drastically improve the performance of the local search learning algorithm. In brief, the proposed PMLS can find better results for learning MVL networks within comparable computation time than previous local search based approaches.

5. Conclusion

This paper presented a novel local search algorithm (LS) based on the probabilistic modeling (PM), for learning multiple-valued logic (MVL) networks. The proposed PMLS incorporated the global statistical information collected from local best solution found by the past search into the best-so-far found solution and therefore the best-so-far solution can be flipped to jump out the current local minimum and keep on searching. Simulation results on several benchmark MVL networks showed that the PMLS was better than previous LS-based methods.

Since the PMLS is a general optimization algorithm framework, it therefore can be applied to other real-world problems. In the future, we will apply the PMLS to solve other combinatorial optimization problems, such as the polygonal approximation, the job-shop scheduling problem, the quadratic assignment problem, and so on. Further, we also plan to incorporate the PM into other methodologies, for example, the ant colony optimization and the memetic algorithm.

References


Appendix: Solutions in Fig. 4.

Solution of Fig. 4(a):
\[
F_1(x_1, x_2) = 2 \land x_1(0, 3) \land x_2(2, 2) \lor 1 \land x_3(0, 3) \land x_3(2, 3) \lor 2 \land x_4(0, 3) \land x_3(1, 3) \lor 1 \land x_1(1, 2) \land x_2(0, 3) \lor 1 \land x_1(0, 2) \land x_2(0, 1)
\]
(A-1)

Solution of Fig. 4(b):
\[
F_1(x_1, x_2) = 3 \land x_1(0, 2) \land x_2(2, 2) \lor 1 \land x_3(1, 3) \land x_3(3, 3) \lor 1 \land x_1(0, 3) \land x_3(0, 2)
\]
(A-2)

Solution of Fig. 4(c):
\[
F_1(x_1, x_2) = 2 \land x_1(1, 2) \land x_2(0, 3)
\]
(A-3)

Solution of Fig. 4(d):
\[
F_1(x_1, x_2) = 1 \land x_1(0, 0) \land x_2(0, 1) \lor 1 \land x_1(3, 3) \land x_2(0, 3)
\]
GAO et al.: LOCAL SEARCH WITH PROBABILISTIC MODELING FOR LEARNING MULTIPLE-VALUED LOGIC NETWORKS

\[ \vee 2 \land x_1(3, 3) \land x_2(3, 3) \vee 1 \land x_1(1, 1) \land x_2(1, 3) \]
\[ \vee 3 \land x_1(0, 2) \land x_2(2, 2) \] (A·4)

Shangce Gao received a B.S. degree from Southeast University, Nanjing, China in 2005 and an M.S. degree from University of Toyama, Toyama, Japan in 2008. Now, he is working toward the D.E. degree at University of Toyama, Toyama, Japan. His main research interests are multiple-valued logic, artificial immune system and artificial neural networks.

Qiping Cao received the B.S. degree from Zhejiang University, Zhejiang, China, an M.S. degree from Beijing University of Posts and Telecommunications, Beijing, China and a D.E. degree from Kanazawa University, Kanazawa, Japan in 1983, 1986 and 2005, respectively. From 1987 to 1989, she was an assistant professor in the Institute of Microelectronics at Shanghai Jiaotong University, Shanghai, China. From 1989 to 1990, she was a research student in Nagoya University, Nagoya, Japan. From 1990 to 2000, she was a senior engineer in Sanwa Newtech Inc., Japan. In 2000, she joined Tateyama Systems Institute, Japan. Her current research interests include multiple-valued logic, neural networks and optimizations.

Masahiro Ishii obtained a Diploma in Electronic Engineering at Akita University in 1990 and a Doctorate from Tokyo Institute of Technology in 1995. Between 1995 and 1997 he was a postdoctoral fellow at the Center for Vision Research in York University of Canada. He was a research associate at Tokyo Institute of Technology between 1997 and 2000. In 2000 he moved to University of Toyama, where he is currently as associate professor of Computer Engineering. His research interests include human computer interaction, human perception and artificial neural networks.

Zheng Tang received the B.S. degree from Zhejiang University, Zhejiang, China in 1982 and an M.S. degree and a D.E. degree from Tsinghua University, Beijing, China in 1984 and 1988, respectively. From 1988 to 1989, he was an Instructor in the Institute of Microelectronics at Tsinghua University. From 1990 to 1999, he was an associate professor in the Department of Electrical and Electronic Engineering, Miyazaki University, Miyazaki, Japan. In 2000, he joined Toyama University, Toyama, Japan, where he is currently a professor in the Department of Intellectual Information Systems. His current research interests include intellectual information technology, neural networks, and optimizations.