Chapter 3

TABU SEARCH

Fred Glover[1] and Rafael Martí[2]
[1] *Leeds School of Business, University of Colorado, Campus Box 419, Boulder, CO 80309;*
[2] *Dpto. de Estadística e Investigación Operativa, Universidad de Valencia, Dr. Moliner 50, 46100 Burjassot (Valencia) Spain*

Abstract: Tabu Search is a metaheuristic that guides a local heuristic search procedure to explore the solution space beyond local optimality. One of the main components of tabu search is its use of adaptive memory, which creates a more flexible search behavior. Memory based strategies are therefore the hallmark of tabu search approaches, founded on a quest for "integrating principles," by which alternative forms of memory are appropriately combined with effective strategies for exploiting them. In this chapter we address the problem of training multilayer feed-forward neural networks. These networks have been widely used for both prediction and classification in many different areas. Although the most popular method for training these networks is backpropagation, other optimization methods such as tabu search have been applied to solve this problem. This chapter describes two training algorithms based on the tabu search. The experimentation shows that the procedures provide high quality solutions to the training problem, and in addition consume a reasonable computational effort.

Key words: Intelligent problem solving; memory structures; adaptive memory programming.

1. INTRODUCTION

The basic form of Tabu Search (TS) is founded on ideas proposed by Fred Glover (1986). The method is based on procedures designed to cross boundaries of feasibility or local optimality, instead of treating them as barriers. Early examples of these procedures, derived from surrogate constraint methods and cutting plane approaches, systematically impose and released constraints to permit exploration of otherwise forbidden regions.

TS is based on the premise that problem solving, in order to qualify as intelligent, must incorporate adaptive memory and responsive exploration. The adaptive memory feature of TS allows the implementation of procedures that are capable of searching the solution space economically and effectively. Since local choices are guided by information collected during the search, TS contrasts with memory-less designs that heavily rely on semi-random processes that implement a form of sampling. The emphasis on responsive exploration (and hence purpose) in tabu search, whether in a deterministic or probabilistic implementation, derives from the supposition that a bad strategic choice can often yield more information than a good random choice, and therefore provides a basis for progressively improved strategies that take advantage of search history. TS can be directly applied to virtually any kind of optimization problem. We focus here on the non-linear problem with continuous variables that arises in the context of neural network training.

In the remainder of the chapter, we first introduce in Section 2 the basic tabu search methodology, and then in Section 3 describe the neural network training application we are interested in examining. Section 4 describes two TS methods that have been developed for solving this problem, one by Sexton et al. (1998) and the other by El_Fallahi et al. (2005). Section 5 is devoted to computational experiments that compare outcomes from these two methods, and shows the advantages these methods yield over a state-of-the-art variant of the classical backpropagation approach. The chapter finishes with relevant conclusions.

2. TABU SEARCH METHODOLOGY

In its best known form, tabu search can be viewed as beginning in the same way as ordinary local or neighborhood search, proceeding iteratively from one point (solution) to another until a chosen termination criterion is satisfied. Each solution x has an associated neighborhood $N(x) \subset X$, and each solution $x' \in N(x)$ is reached from x by an operation called a *move.*

We may contrast TS with a simple descent method where the goal is to minimize $f(x)$. Such a method only permits moves to neighbor solutions that improve the current objective function value and ends when no improving solutions can be found. The final x obtained by a descent method is called a local optimum, since it is at least as good as or better than all solutions in its neighborhood. The evident shortcoming of a descent method is that such a local optimum in most cases will not be a global optimum, i.e., it usually will not minimize $f(x)$ over all $x \in \mathbf{X}$.

Tabu search permits moves that deteriorate the current objective function value and selects the moves from a modified neighborhood $N^*(x)$. Short and long term memory structures are responsible for the specific composition of $N^*(x)$. In other words, the modified neighborhood is the result of maintaining a selective history of the states encountered during the search. In TS strategies based on short term considerations, $N^*(x)$ characteristically is a subset of $N(x)$, and the tabu classification serves to identify elements of $N(x)$ excluded from $N^*(x)$. In TS strategies that include longer term considerations, $N^*(x)$ may also be expanded to include solutions not ordinarily found in $N(x)$, such as solutions found and evaluated in past search, or identified as high quality neighbors of these past solutions. Characterized in this way, TS may be viewed as a dynamic neighborhood method. This means that the neighborhood of x is not a static set, but rather a set that can change according to the history of the search.

The structure of a neighborhood in tabu search differs from that used in local search in an additional manner, by embracing the types of moves used in constructive and destructive processes (where the foundations for such moves are accordingly called *constructive neighborhoods* and *destructive neighborhoods*). Such expanded uses of the neighborhood concept reinforce a fundamental perspective of TS, which is to define neighborhoods in dynamic ways that can include serial or simultaneous consideration of multiple types of moves.

TS uses attributive memory for guiding purposes (i.e., to compute $N^*(x)$). Instead of recording full solutions, attributive memory structures are based on recording attributes. This type of memory records information about solution properties (attributes) that change in moving from one solution to another. The most common attributive memory approaches are recency-based memory and frequency-based memory.

Recency-based memory is the most common memory structure used in TS implementations. As its name suggests, this memory structure keeps track of solutions attributes that have changed during the recent past. To exploit this memory, selected attributes that occur in solutions recently visited are labeled *tabu-active*, and solutions that contain tabu-active elements, or particular combinations of these attributes, are those that become tabu. This prevents certain solutions from the recent past from belonging to $N^*(x)$ and hence from being revisited. Other solutions that share such tabu-active attributes are also similarly prevented from being visited. Although the tabu classification strictly refers to solutions that are forbidden to be visited, by virtue of containing tabu-active attributes (or more generally by violating certain restriction based on these attributes), moves that lead to such solutions are also often referred to as being tabu.

Frequency-based memory provides a type of information that complements the information provided by recency-based memory, broadening the foundation for selecting preferred moves. Like recency, frequency often is weighted or decomposed into subclasses that can refer to particular subregions of the search space. Also, frequency can be integrated with recency to provide a composite structure for creating penalties and inducements that modify move evaluations.

A key element of the adaptive memory framework of tabu search is to create a balance between search *intensification* and *diversification.* Intensification strategies are based on modifying choice rules to encourage move combinations and solution features historically found good. They may also initiate a return to attractive regions to search them more thoroughly. Diversification strategies, on the other hand, seek to incorporate new attributes and attribute combinations that were not included within solutions previously generated. In one form, these strategies undertake to drive the search into regions dissimilar to those already examined. It is important to keep in mind that intensification and diversification are not mutually opposing, but are rather mutually reinforcing.

Most types of intensification strategies require a means for identifying a set of elite solutions as basis for incorporating good attributes into newly created solutions. Membership in the elite set is often determined by setting a threshold that is connected to the objective function value of the best solution found during the search. Two simple variants for elite solution selection have proved quite successful. One introduces a diversification measure to assure the solutions recorded differ from each other by a desired degree, and then erases all short term memory before resuming from the best of the recorded solutions. The other keeps a bounded length sequential list that adds a new solution at the end only if it is better than any previously seen, and the short term memory that accompanied this solution is also saved.

A degree of diversification is automatically created in TS by short term memory functions, but effective diversification is particularly supported by certain forms of longer term memory. TS diversification strategies are often based on modifying choice rules to bring attributes into the solutions that are infrequently used. Alternatively, they may introduce such attributes by periodically applying methods that assemble subsets of these attributes into candidate solutions for continuing the search, or by partially or fully restarting the solution process. Diversification strategies are particularly helpful when better solutions can be reached only by crossing barriers or "humps" in the solution space topology.

A TS process based strictly on short term strategies may allow a solution x to be visited more than once, but it is likely that the corresponding reduced

neighborhood $N^*(x)$ will be different each time. The inclusion of longer term considerations effectively removes the risk of duplicating a previous neighborhood upon revisiting a solution and more generally of making choices that repeatedly visit only a limited subset of X.

An extensive description of the TS methodology can be found in Glover and Laguna (1997) and the integration of evolutionary methods with tabu search memory is treated in Laguna and Martí (2003).

2.1 Advanced Designs

There are many forms in which a simple tabu search implementation can be improved by adding long term elements. We restrict our attention to two of the most commonly used methods, namely strategic oscillation and path relinking, which constitute the core of many adaptive memory programming algorithms.

Strategic oscillation operates by orienting moves in relation to a critical level, as identified by a stage of construction or a chosen interval of functional values. Such a critical level or oscillation boundary often represents a point where the method would normally stop. Instead of stopping when this boundary is reached, however, the rules for selecting moves are modified, to permit the region defined by the critical level to be crossed. The approach then proceeds for a specified depth beyond the oscillation boundary, and turns around. The oscillation boundary again is approached and crossed, this time from the opposite direction, and the method proceeds to a new turning point (see Figure 3-1).

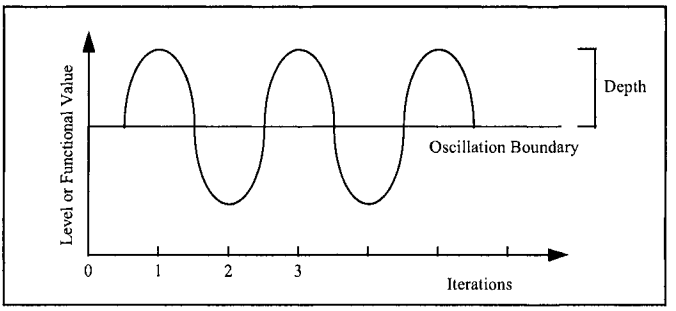

Figure 3-1. Strategic Oscillation.

The process of repeatedly approaching and crossing the critical level from different directions creates an oscillatory behavior, which gives the method its name. Control over this behavior is established by generating modified evaluations and rules of movement, depending on the region

navigated and the direction of search. The possibility of retracing a prior trajectory is avoided by standard tabu search mechanisms, like those established by the recency-based and frequency-based memory functions.

When the level or functional values in Figure 1 refers to degrees of feasibility and infeasibility, a vector-valued function associated with a set of problem constraints can be used to control the oscillation. In this case, controlling the search by bounding this function can be viewed as manipulating a parameterization of the selected constraint set. A preferred alternative is often to make the function a Lagrangean or surrogate constraint penalty function, avoiding vector-valued functions and allowing tradeoffs between degrees of violation of different component constraints.

Path Relinking, as a strategy of creating trajectories of moves passing through high quality solutions was first proposed in connection with tabu search in Glover (1989). The approach was then elaborated in greater detail as a means of integrating TS intensification and diversification strategies, and given the name path relinking (PR), in Glover and Laguna (1993). PR generally operates by starting from an initiating solution, selected from a subset of high quality solutions, and generating a path in the neighborhood space that leads toward the other solutions in the subset, which are called guiding solutions. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions.

Path relinking can be considered an extension of the Combination Method of Scatter Search (Glover and Laguna, 1993; Laguna and Martí, 2003). Instead of directly producing a new solution when combining two or more original solutions, PR generates paths between and beyond the selected solutions in the neighborhood space. The character of such paths is easily specified by reference to solution attributes that are added, dropped or otherwise modified by the moves executed. Examples of such attributes include edges and nodes of a graph, sequence positions in a schedule, vectors contained in linear programming basic solutions, and values of variables and functions of variables.

The approach may be viewed as an extreme (highly focused) instance of a strategy that seeks to incorporate attributes of high quality solutions, by creating inducements to favor these attributes in the moves selected. However, instead of using an inducement that merely encourages the inclusion of such attributes, the path relinking approach subordinates other considerations to the goal of choosing moves that introduce the attributes of the guiding solutions, in order to create a "good attribute composition" in the current solution. The composition at each step is determined by choosing the best move, using customary choice criteria, from a restricted set — the set of those moves currently available that incorporate a maximum number (or a maximum weighted value) of the attributes of the guiding solutions.

(Exceptions are provided by aspiration criteria, as subsequently noted.) The approach is called path relinking either by virtue of generating a new path between solutions previously linked by a series of moves executed during a search, or by generating a path between solutions previously linked to other solutions but not to each other.

To generate the desired paths, it is only necessary to select moves that perform the following role: upon starting from an initiating solution, the moves must progressively introduce attributes contributed by a guiding solution (or reduce the distance between attributes of the initiating and guiding solutions). The roles of the initiating and guiding solutions are interchangeable; each solution can also be induced to move simultaneously toward the other as a way of generating combinations. First consider the creation of paths that join two selected solutions x' and x'', restricting attention to the part of the path that lies 'between' the solutions, producing a solution sequence $x' = x(1), x(2), \ldots, x(r) = x''$. To reduce the number of options to be considered, the solution $x(i + 1)$ may be created from $x(i)$ at each step by choosing a move that minimizes the number of moves remaining to reach x''. The relinked path may encounter solutions that may not be better than the initiating or guiding solution, but that provide fertile "points of access" for reaching other, somewhat better, solutions. For this reason it is valuable to examine neighboring solutions along a re-linked path, and keep track of those of high quality which may provide a starting point for launching additional searches.

3. THE NEURAL NETWORK MODEL EXAMINED

Artificial neural networks (ANNs) have been widely used for both classification and prediction. The classification or recognition problem consists of the identification of the class to which a given object belongs. The input of the net is a description of the object to be recognized, while the output is a discrete value identifying its class. The prediction problem consists in approximating unknown functions. In this chapter we restrict our attention to this latter problem; in particular we focus on the approximation of real mappings ($f: \Re^n \rightarrow \Re$).

We consider the most widely-used architecture for prediction and classification: a multilayer feed-forward network with a single hidden layer. In particular, we target a two layer feed-forward network, with sigmoid activation function in the hidden nodes and linear activation in the output node. Let $NN=(N, A)$ be an ANN where N is the set of nodes (neurons) and A is the set of arcs. N is partitioned into three subsets: N_I, input nodes, N_H, hidden nodes and N_O, output nodes. We assume that there are n variables in

the function that we want to predict or approximate, therefore $|N_I| = n$. The neural network has m hidden neurons ($|N_H| = m$) with a bias term in each hidden neuron and a single output neuron. There is an arc, with an associated weight, from each node in N_I to each node in N_H, and from each node in N_H to the output node.

The net's input is given by the values of the function variables and the output is the estimation of the function image. We focus on the prediction-approximation problem, therefore, the output of the net for a given input, should be as close as possible to the value of a given function for this input. In mathematical terms, given a real function $f: \Re^n \rightarrow \Re$ and a neural net NN, the objective is to find appropriate values for the arc weights w of the net, such as its output $NN(x,w)$ from an input vector x, approximates the value $f(x)$. We refer the reader to the excellent book by Bishop (1995) for a comprehensive review of ANNs.

The most common error measure used to report the quality of the network performance is the Root Mean Squared Error (RMSE). Let $E=\{x^1, x^2,...x^t\}$ be a random sample of points in the domain of f (usually called the training set), and suppose that the value of $f(x)$ is known for all x in E. Given the weights w, for each x in E the error can be computed as:

$$error(x,w) = [\, f(x) - NN(x,w) \,]^2,$$

and the RMSE across all the elements in the training set E is given by:

$$Error(E,w) = \sqrt{\frac{\sum_{i=1}^{t} error(x^i, w)}{t}}.$$

Therefore, training the neural network can be formulated as the following non-linear unconstrained optimization problem:

$$\min_{w} \; Error(E, w).$$

Backpropagation (BP) is the most widely used optimization method for accomplishing this training. It is an effective algorithm based on the steepest descent direction. Several modifications and improvements to the original method have been proposed, as it is the case of the "momentum" term where each new search direction is computed as a weighted sum of the current gradient and the previous search direction. Recently, metaheuristics such as simulated annealing (SA), genetic algorithms (GA) and tabu search (TS) have been proposed to solve this optimization problem. In this chapter we focus on the tabu search methodology and its applications to solve the neural network training problem.

4. TABU SEARCH TRAINING METHODS

In this section we describe the two tabu search implementations we will be examining for the purpose of minimizing error when training a neural network: the extended tabu search method by Sexton et al. (1998) and the tabu search approach with path relinking by El-Fallahi et al. (2005).

4.1 The Extended Tabu Search Method

The Extended Tabu Search method by Sexton et al. (1998) is mainly based on a random sampling around the best solution found. The authors propose two methods, the first one, called "preliminary" TS, is used as a baseline for comparison with the second one, Extended Tabu Search (ETS), which is the main contribution of the paper. A description of the extended method follows. Since the training set E is fixed, from now on we simplify the notation $Error(E,w)$ and use plainly $E(w)$.

An initial solution w_0 is randomly drawn from a uniform distribution in the range [-10,10] and the current best solution w_{best} is initialised to w_0. Solutions are randomly generated in this range for a given number of iterations. When generating a new point w_{new}, aspiration level and tabu conditions are checked. If $E(w_{new})<E(w_{best})$, the point is automatically accepted and both w_{best} and $E(w_{best})$ are updated; otherwise the tabu conditions are tested. If there is one solution w_i in the tabu list (TL) such as $E(w_{new}) \in [E(w_i)-0.01*E(w_i), E(w_i)+0.01*E(w_i)]$, then the complete test is applied to w_{new} and w_i; otherwise the point is accepted. The test checks if all the weights in w_{new} are within ± 0.01 from w_i, in this case the point is rejected, otherwise the point is accepted and w_{new} and $E(w_{new})$ are entered into TL. This process continues for 1000 iterations of accepted solutions. Then, another cycle of 1000 iterations of random sampling begins. These cycles will continuously repeat while $E(w_{best})$ improves.

When the random sampling ends, the process of intensification starts by performing a search from the best solution found w_{best}. The new points are drawn by modifying the w_{best} by a small *step* value, where:

$$step=((0.1*w_{best})-(0.2*w_{best})*random)/change. \quad (1)$$

Each cycle of the intensification phase generates 1000 new points. This phase makes a maximum of 20 cycles as long as there is at least one reduction in the $E(w_{best})$. Once this phase finishes, the diversification process begins in order to expand the search area. The step value is now computed as:

$$step=((0.1*w_{best})-(0.2*w_{best})*random)*change \quad (2)$$

This diversification phase generates new points by modifying w_{best} with step value (2). As in the intensification phase, cycles of 1000 iterations are performed up to a maximum of 20. Both phases, intensification and diversification, are alternated for a maximum of 5 consecutive iterations. The *random* variable is a random number drawn from a uniform distribution in the range [0, 1], the *change* variable is initialised to one, and is increased in one after each intensification phase. The whole process consists of 10 of these global iterations.

It is important to point out that in both this method and the one described in the next subsection, the search takes place only over the weights from the input to the hidden layer and the bias factor of the hidden neurons. Weights from the hidden layer to the output neuron, $w_{n+j,s}$ as well as the bias factor of node s, w_s, are obtained with linear regression to minimize the sum of squares associated with *Error(E,w)*. The advantage of this search scheme is that the number of weights that the training procedure needs to adjust is reduced by $m+1$. The disadvantage, on the other hand, is that the regression model needs to be solved every time any of the first $m(n+1)$ weights is changed in order to calculate the mean squared error.

4.2 The Tabu Search Method with Path Relinking

The tabu search algorithm by El-Fallahi et al. (2005) operates in a somewhat different manner. In this approach the short term memory is implemented in a beginning phase called TSProb. An iteration of TSProb begins by randomly selecting a weight from the current solution w. The probability of selecting weight w_i^t at iteration t, is proportional to the absolute value of the partial derivative of the RMSE on E with respect to w_i^t. These derivative values can be efficiently computed with the first phase of the BP method. The neighborhood consists of solutions that are reached from w^t by modifying the value of the selected weight w_i^t. Specifically, three solutions are considered with the following expression:

$$w_i^{t+1} = w_i^t + \alpha\beta w_i^t \;;\; w_j^{t+1} = w_j^t \;,\; \forall j \neq i$$

The method selects the best solution from among the three considered (given appropriate α values), and labels it as w^{t+1}. Note that the move is executed even when the error of w^{t+1} is greater than the error of w^t, thus resulting in a deterioration of the current value of the objective function. The moved weight becomes tabu-active for *TabuTenure* iterations, and therefore it cannot be selected during this time. The factor β scales the change in the selected weight according to the status of the search (reducing

its value from 1 as long as the current solution is close to a local optimum). Starting from a random initial solution, the TSProb method finishes after a number of k consecutive iterations with no improvement. The search parameters have been set to the values recommended by the authors: *TabuTenure*= $n(m+1)/3$, $\alpha=(0.3, 0.5, 0.8)$, $\beta\in[0,1]$ and k=500.

The foregoing method is coupled with a Path Relinking phase, which is a form of TS strategy that is finding increasing use in applications. It starts with the creation of the Reference Set (*RefSet*), which contains the b elite solutions found during the application of the TSProb method. These b solutions must be different and they must be far enough apart to ensure that the BFGS improvement method (Smith and Lasdon, 1992) will converge to different final solutions. Therefore, a solution is admitted to *RefSet* if its Euclidean distance from each solution already in the set is larger than a pre-specified threshold *th_d*. The improvement method is applied to the $b/2$ best solutions in *RefSet* and the improved solutions are ordered according to quality (i.e., to their *error(E,w)* value).

At each iteration of the path relinking algorithm, the set *NewPairs* is constructed with all pairs of solutions in *RefSet* that include at least one new solution. (In the first iteration it contains $(b^2-b)/2$ pairs, but in successive iterations this number is usually significantly smaller.) For each pair (w', w'') in *NewPairs* a path is initiated from w' to w'', and the best solution found in the path is added to the set *PRSol*. Once all the pairs in NewPairs have been subjected to the path relinking method, the BFGS algorithm is applied to the best b solutions in *PRSol*. Each newly created solution is tested to determine whether it improves upon the worst solution currently in *RefSet*, in which case the new solution replaces the worst and *RefSet* is reordered. Then, if *RefSet* contains a new solution we perform another iteration of the path relinking algorithm, starting with the creation of the set NewPairs; otherwise, the algorithm terminates.

The path relinking method constructs a path to join two solutions u and v generated by the process described above. Considering the m neurons in the hidden layer in a given order, a path containing m solutions is constructed from solution u to solution v by performing moves that transform u into v. The first step creates the first solution in the path, w^1, by replacing in u the values of the weights in the arcs from the n input neurons to the first hidden neuron with their values in v. Similarly, the second step creates the solution w^2 by replacing in w^1 the values of the weights in the arcs from the n input neurons to the second hidden neuron with their values in v. The method proceeds in this way until we obtain solution w^m, which only differs from solution v in the values associated with the weights from the hidden layer to the output neuron.

```
0. Select the training set E and the testing set T. Normalize input
   and output data.
1. Build RefSet = { w(1), …, w(b) } with the best solutions found with
   the TSProb method. Apply the improvement method (with stopping
   condition modified as explained below) to the best b/2 solutions in
   RefSet.
2. Order RefSet according to their objective function value such that
   w(1) is the best solution and w(b) the worst. Compute
   E_best=error(E,w(1)) and T_best as the minimum of error(T,w(i)) for
   i=1,…,b. Set T_Improve = 0.
  while ( T_Improve < T_Limit ) do
   3. Generate NewPairs, which consists of all pairs of solutions in
      RefSet that include at least one new solution. Make
      NewSolutions = Ø.
   for ( all NewPairs ) do
      4. Select the next pair ( w(i), w(j) ) in NewPairs.
      5. Obtain new solutions in the path from w(i) to w(j) and add
         the best one to NewSolutions.
   end for
   6. Select the best b solutions in NewSolutions and apply the
   improvement method.
   for ( each improved w ) do
      if ( w is not in RefSet and error(E,w)<error(E,w(b)) ) then
         7. Make w(b) = w and reorder RefSet.
   end for
   8. Make T_current = the minimum of error(T,w(i)) for i=1,…,b.
   if ( T_current < T_best ) then
      9. Make T_best=T_current and T_improve=0.
   else
      10. Make T_improve= T_improve +1.
  end while
```

Figure 3-2. Path relinking pseudocode.

The effectiveness of adding a local search exploration from some of the generated solutions within the relinking path has been well documented (Laguna and Martí, 2003). In the context of neural network training, the application of the BFGS procedure as the improvement method is a time-consuming operation, so we have limited it to the best solution found in the path, as described above.

5. COMPUTATIONAL EXPERIMENTS

For our computational testing, we have created C implementations of: (1) an effective (state-of-the-art) variant of the classical Backpropagation method (BP), (2) the extended tabu search method, ETS, of Sexton et al. (1998) and (3) the TS procedure with Path Relinking, TSPR, of El-Fallahi et al. (2005). Figure 3-3 shows the expression of the 15 functions used to compare the performance of the 3 methods under consideration.

Backpropagation is one of the first methods for neural network training, and is the most widely used algorithm in practical applications. It is a gradient descent procedure that computes the derivatives' values in a very efficient way (from the output layer back towards the input layer), and modifies the weights according to a parameter known as 'learning rate'. The original algorithm has been modified in many ways; the most popular consists in adding a 'momentum' term (Rumelhart and McClelland, 1986) when the weights are updated. The inclusion of this term leads to significant improvements, although it introduces a second parameter in the algorithm. Jacobs (1988) suggested a different modification called the 'delta-bar-delta rule' which introduces a separate learning rate for each weight. It has been shown (Bishop, 1995) that this rule increases the convergence of the method in some cases, but does not work well in practice across different instances due to some stability problems. Several methods have been proposed to compute the learning rate. Examples are the *quickprop* method (Fahlman, 1988) and the *Rprop* method (Riedmiller and Heinrich, 1993). However, in general, these variants share the limitations associated with first derivative based methods. Our adaptation of the backpropagation algorithm (BP) has the learning rate set to 0.3 and the momentum term set to 1, and compares favorably with commercial implementations of this method, as documented by (El-Fallahi, 2002).

The training set consists of 200 observations with data randomly drawn from [-100, 100] for x_1 and [-10,10] for x_2. The validation set consists of 100 observations drawn from the same uniform distributions that were not used in the search process at all. We use one hidden layer with 9 nodes in all the experiments as it is done in previous works (El-Fallahi et al., 2005).

1.Sexton 1: $f(x) = x_1 + x_2$

2. Sexton 2: $f(x) = x_1 * x_2$

3. Sexton 3: $f(x) = \frac{x_1}{|x_2| + 1}$

4. Sexton 4: $f(x) = x_1^{\,2} - x_2^{\,3}$

5. Sexton 5: $f(x) = x_1^{\,3} - x_1^{\,2}$

6. Branin: $f(x) = \left(x_2 - \left(\frac{5}{4\pi^2}\right)x_1^2 + \left(\frac{5}{\pi}\right)x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_i) + 10$

7. B2: $f(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$

8. Easom: $f(x) = -\cos(x_1)\cos(x_2)\exp\left(-\left((x_1 - \pi)^2 + (x_2 - \pi)^2\right)\right)$

9. Goldstein:

$$f(x) = \left(1 + (x_1 + x_2 + 1)^2\left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right)$$
$$\left(30 + (2x_1 - 3x_2)^2\left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right)$$

10. Shubert: $f(x) = \left(\sum_{j=1}^{5} j\cos((j+1)x_1 + j)\right)\left(\sum_{j=1}^{5} j\cos((j+1)x_2 + j)\right)$

11. Beal:

$$f(x) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_1x_2^3)^2$$

12. Booth: $f(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$

13. Matyas: $f(x) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2$

14. SixHumpCamelB: $f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$

15. Schwefel: $f(x) = 418.9829n + \sum_{i=1}^{n}\left(-x_i \sin\sqrt{|x_i|}\right)$

Figure 3-3. Test Functions.

Tables 3-1 and 3-2 report, respectively, the training and validation errors obtained with the three methods in the 15 problems considered. In order to obtain statistically significant solutions, we run each method 20 times on each function and report the average and standard deviation of the 20 runs (limiting each run to ten minutes). In all the cases, we have employed the same training and validation sets. All the experiments have been performed on a Pentium IV 2.8 Ghz personal computer.

Table 3-1. Training Error across different methods

NP	BP	ETS	TSPR
1	1.60 ± 0.26	0.04 ± 0.02	0.00 ± 0.00
2	8.32 ± 4.30	1.79 ± 0.78	0.00 ± 0.00
3	1.63 ± 0.21	0.34 ±0.03	0.00 ± 0.00
4	45.52 ± 7.82	17.66 ±6	0.00 ± 0.00
5	12.62 ± 3.87	18.98 ±5.26	0.00 ± 0.00
6	13.98 ± 1.58	53.28 ±3.94	0.09 ± 0.04
7	16.09 ± 5.80	63.26 ±1.18	0.25 ± 0.00
8	0.20 ± 0.06	0.01 ±0.00	0.00 ± 0.00
9	7.35E+09±1.07E+09	3.30E+09±8.44E+07	1.37E+09±1.61E+08
10	21.40 ± 1.49	22.22± 4.12	16.14 ± 1.67
11	5.28E+06±1.34E+06	4.17E+06±1.28E+05	1.80E+06±1.36E+05
12	107.95 ± 3.01	156.12±5.57	0.01 ± 0.00
13	3.93 ± 1.97	10.13 ± 3.25	0.00 ± 0.00
14	5.58E+0 ± 6.76E+03	4.44E+04±2.48E+03	1.34E+04±8.34E+03
15	2.88 ± 0.5	527.14±3.07	0.02 ± 0.00

Table 3-2. Validation Error across different methods

NP	BP	ETS	TSPR
1	1.50 ± 0.22	0.05 ± 0.05	0.00 ± 0.00
2	7.91± 3.10	2.06 ± 0.85	0.00 ± 0.00
3	1.72 ± 0.21	0.67 ± 0.05	0.00 ± 0.00
4	48.03 ± 8.98	20.91 ± 7.15	0.00 ± 0.00
5	11.60 ± 2.68	21.43 ± 6.55	0.00 ± 0.00
6	15.09 ± 1.36	53.18 ± 4.80	0.00 ± 0.00
7	17.63 ± 5.87	61.2 ± 1.85	0.00± 0.00
8	0.20 ± 0.06	0.00 ± 0.00	0.00 ± 0.00
9	1.01E+10±1.75E+09	7.41E+09±3.59E+08	2.15E+09±4.58E+01
10	17.22 ± 2.9	25.59 ± 0.3	20.7 ± 0.64
11	3.83E+06±2.42E+05	5.89E+06±2.57E+05	3.29E+06±5.01E+05
12	112.09 ± 5.13	162.47 ± 7.34	0.00 ± 0.00
13	4.72 ± 2.74	10.52 ± 0.41	0.01 ± 0.00
14	5.20E+04±5.54E+03	4.38E+04±2.96E+03	1.44E+04±7.78E+03
15	2.78 ± 0.46	528.26±18.48	0.03 ± 0.00

Tables 3-1 and 3-2 show that the best solution quality is obtained with the TSPR method in all cases. Note that this method presents a lower average error as well as a lower deviation than the other methods. This experiment also shows that none of the methods can effectively handle problems 9, 11 and 14 within the run time considered, suggesting either that the ANN model itself needs to be modified in these cases or that the structure of these problems poses an unusual level of difficulty. Considering the average values over the 20 runs, Table 1 shows that the TSPR method is able to obtain the best solutions with respect to the training error in each of the other 12 instances. Table 2 shows similar results since TSPR obtains the best solutions in the 12 cases with reasonable error values. It should be

mentioned that our BP algorithm variant obtains deviation values that are reasonably good, on average, considering its simplicity.

Figure 3-4 shows the average validation error on the 12 instances in which the methods provide relative good approximation. We have replaced the 528.26 achieved with the ETS method on instance 15 with a 180 for scaling purposes. This figure clearly shows the superiority of the TSPR method compared with the other approaches under consideration.

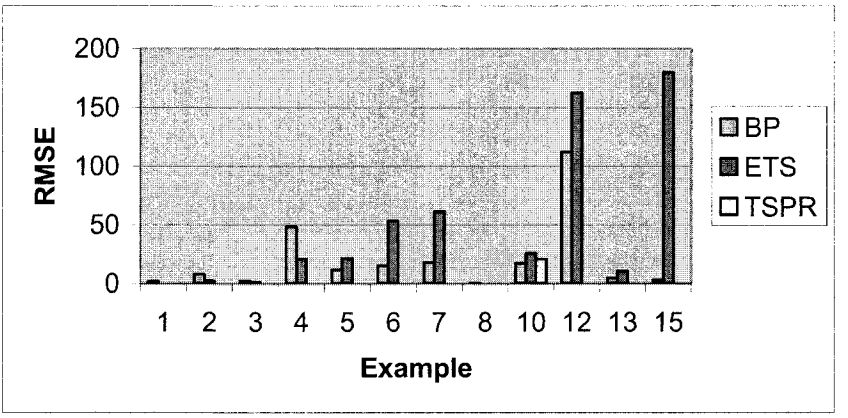

Figure 3-4. Average Validation Error.

6. CONCLUSIONS

The focus and emphasis of tabu search have a number of implications for the goal of designing improved optimization procedures. These opportunities carry with them an emphasis on producing systematic and strategically designed rules, rather than following the policy of relegating decisions to random choices as is often fashionable in evolutionary methods. The adaptive memory structures underlying tabu search and the excellent results that they provide, invites the use of TS in other metaheuristics.

We have described different implementations of tabu search for training a single-layer feed-forward neural network. Two TS methods were compared with the well known Backpropagation algorithm. The best results are obtained by the Tabu Search Path Relinking method coupled with an improvement phase based on the BFGS optimizer.

ACKNOWLEDGMENTS

Research by Rafael Martí is partially supported by the *Ministerio de Educación y Ciencia* (refs. TIN2004-20061-E and TIC2003-C05-01) and by the *Agencia Valenciana de Ciència i Tecnologia* (ref. GRUPOS03 /189).

REFERENCES

Bishop, C. M., 1995, *Neural Networks for Pattern Recognition*, Oxford University Press.

El-Fallahi A., 2002, *Entrenamiento de Redes Neuronales*, Trabajo de Investigación, Dpto Estadística e I.O. University of Valencia.

El-Fallahi, A., Martí, R., and Lasdon, L., 2005, Path relinking and GRG for artificial neural networks, *European journal of operational research* **169**:508-519.

Fahlman, S. E., 1988, An empirical study of learning speed in back-propagation networks, in: *Connectionist Models Summer School*, T. J. Sejnowski, G. E. Hinton and D. S. Touretzky, eds., San Mateo, CA, Morgan Kaufmann, pp. 38-51.

Glover, F., 1986, Future paths for integer programming and links to artificial intelligence, *Computers and Operations Research* **13**:533-549.

Glover, F., 1989, Tabu search part I, *ORSA Journal on Computing* **1**:190-206.

Glover, F., and Laguna, M., 1993, Tabu search, in: *Modern Heuristic Techniques for Combinatorial Problems,* C. Reeves, ed., Blackwell Scientific Publishing, Oxford, pp. 70-150.

Glover, F., and Laguna, M., 1997, *Tabu Search,* Kluwer Academic Publishers, Boston.

Jacobs, R. A., 1988, Increased rates of convergence through learning rate adaptation, *Neural Networks* **1**:295-307.

Laguna, M., Martí, R., 2003, *Scatter Search—Methodology and Implementations in C,* Kluwer Academic Publishers, Boston.

Riedmiller, M., and Heinrich, B., 1993, A direct adaptive method for faster back-propagation learning: the RPROP algorithm, *IEEE Intl. Conf. on Neural Networks*, pp. 586-591.

Rumelhart, D. E., and McClelland, J. L., 1986, *Parallel distributed processing: explorations in the microstructure of cognition*, Cambridge, MA: The MIT Press.

Sexton, R. S., Alidaee, B., Dorsey, R. E., and Johnson, J. D., 1998, Global optimization for artificial neural networks: a tabu search application, *European Journal of Operational Research* **106**:570-584.

Smith, S., and Lasdon, L., 1992, Solving large nonlinear programs using GRG, *ORSA Journal on Computing* **4**(1): 2-15.