Chapter 8

SCATTER SEARCH

Manuel Laguna[1] and Rafael Martí[2]
[1]*Leeds School of Business, University of Colorado at Boulder,* laguna@colorado.edu;
[2]*Dpto. de Estadística e Investigación Operativa, Universidad de Valencia,* rafael.marti@uv.es

Abstract: This chapter discusses the principles and foundations behind scatter search and its application to the problem of training neural networks. Scatter search is an evolutionary method that has been successfully applied to a wide array of hard optimization problems. Scatter search constructs new trial solutions by combining so-called reference solutions and employing strategic designs that exploit context knowledge. In contrast to other evolutionary methods like genetic algorithms, scatter search is founded on the premise that systematic designs and methods for creating new solutions afford significant benefits beyond those derived from recourse to randomization. Our implementation goal is to create a combination of the 5 elements in the scatter search methodology that proves effective when searching for optimal weight values in a multilayer neural network. Through experimentation, we show that our instantiation of scatter search can compete with the best-known training algorithms in terms of training quality while keeping the computational effort at a reasonable level.

Key words: Metaheuristics, neural networks, optimization

1. INTRODUCTION

Scatter search (SS) was first introduced by Glover (1977) as a heuristic for integer programming. In the original proposal, solutions are purposely (i.e., non-randomly) generated to take account of characteristics in various parts of the solution space. The orientation of SS is to explore systematically relative to a set of reference points that typically consist of good solutions obtained by prior problem solving efforts, where the criteria

for “good” are not restricted to objective function values. In this way, SS shares with other evolutionary methods the philosophy of operating on a set of solutions rather than on a single solution at a time. It also shares the conceptual framework of embedding procedures for combining these solutions to create new ones. However, the meaning of “combining” and the way is carried out have a rather special origin and character in the SS setting. A distinguishing feature of scatter search is its intimate association with the Tabu Search (TS) metaheuristic (Glover and Laguna 1997), which explains its adoption of the principle that search can benefit by incorporating special forms of adaptive memory, along with procedures particularly designed for exploiting that memory. In fact, scatter search and tabu search share common origins, and initially SS was simply considered one of the component processes available within the TS framework. However, most of the TS literature and the preponderance of TS implementations have disregarded this component, with the result that the merits of scatter search did not come to be recognized until quite recently. Nowadays it is a well established methodology within the metaheuristic community, although general awareness of the procedure still lags behind that of other population-based methods such as genetic algorithms and evolutionary strategies.

Artificial neural networks (ANNs) offer a general framework for representing non-linear mappings from several input variables to several output variables. They are built by tuning a set of parameters known as weights, and can be considered as an extension of the multitude of conventional mapping techniques discussed in the literature. While in classification or recognition problems the outputs of a neural network are categories, in prediction or approximation problems they are continuous variables. Although this paper is focused on the prediction problem, most of the key issues in the functionality of a neural network are common to both.

In this chapter we have considered the most commonly employed architecture for prediction and classification: *a multilayer feed-forward network with a single hidden layer*. A schematic representation of such a network appears in Figure 1.

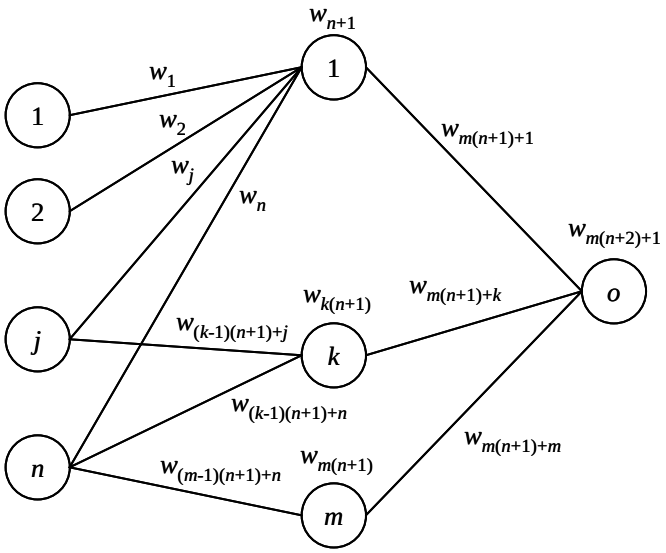

Figure 8-1. Neural network with one hidden layer and one output

Note that the weights in the network depicted in Figure 1 are numbered sequentially starting with the first input to the first hidden neuron. Therefore, the weights for all the inputs to the first hidden neuron are w_1 to w_n. The bias term for the first hidden neuron is w_{n+1}.

In the process of training the neural network (*supervised learning*), the problem is to find the values of the weights w that minimize the error across a set of input/output pairs (patterns). This set is referred to as the training set E. For a single output and input vector x, the error measure is typically the root mean squared difference (RMSE) between the predicted output $p(x,w)$ and the actual output value $f(x)$ for all the elements x in E; therefore, the training is an unconstrained nonlinear optimization problem, where the decision variables are the weights and the objective is to reduce the training error:

$$\min_{w} \; error(E,w) = \sqrt{\frac{\sum_{x \in E} (f(x) - p(x,w))^2}{|E|}}$$

In the remainder of the chapter, we first introduce in Section 2 the scatter search methodology, and then describe in Section 3 the SS method by Laguna and Martí (2000) for the neural network training. Section 4 is devoted to the computational experiments and the chapter finishes with relevant conclusions.

2. SCATTER SEARCH METHODOLOGY

Scatter Search derives its foundations from earlier strategies for combining decision rules and constraints. Historically, the antecedent strategies for combining decision rules were introduced in the context of scheduling methods to obtain improved local decision rules for job shop scheduling problems (Glover, 1963). New rules were generated by creating numerically weighted combinations of existing rules, suitably restructured so that their evaluations embodied a common metric.

The scatter search methodology is very flexible, since each of its elements can be implemented in a variety of ways and degrees of sophistication. In this section we give a basic design to implement scatter search based on the "five methods" given in the well-known *template* (Glover 1998). The advanced features of scatter search are related to the way these five methods are implemented. That is, the sophistication comes from the implementation of the SS methods instead of the decision to include or exclude some elements. An extensive description of advanced designs can be found in Laguna and Martí (2003) and the integration of additional elements with tabu search memory is treated in Glover (2004).

The fact that the mechanisms within scatter search are not restricted to a single uniform design allows the exploration of strategic possibilities that may prove effective in a particular implementation. These observations and principles lead to the following "five-method template" for implementing scatter search.

1. A *Diversification Generation Method* to generate a collection of diverse trial solutions, using an arbitrary trial solution (or seed solution) as an input.
2. An *Improvement Method* to transform a trial solution into one or more enhanced trial solutions. (Neither the input nor the output solutions are required to be feasible, though the output solutions will more usually be expected to be so. If no improvement of the input trial solution results, the "enhanced" solution is considered to be the same as the input solution.)
3. A *Reference Set Update Method* to build and maintain a reference set consisting of the b "best" solutions found (where the value of b is typically small, e.g., no more than 20), organized to provide efficient accessing by other parts of the method. Solutions gain membership to the reference set according to their quality or their diversity.
4. A *Subset Generation Method* to operate on the reference set, to produce a subset of its solutions as a basis for creating combined solutions.

5. A *Solution Combination Method* to transform a given subset of solutions produced by the Subset Generation Method into one or more combined solution vectors.

The Diversification Generation Method is used to build a large set P of diverse solutions. The size of P (*PSize*) is typically at least 10 times the size of *RefSet*. The initial reference set is built according to the Reference Set Update Method. It could consist of selecting b distinct and maximally diverse solutions from P.

The reference set, *RefSet*, is a collection of both high quality solutions and diverse solutions that are used to generate new solutions by way of applying the Combination Method. We can use a simple mechanism to construct an initial reference set and then update it during the search. The size of the reference set is denoted by $b = b_1 + b_2 = |RefSet|$. The construction of the initial reference set starts with the selection of the best b_1 solutions from P. These solutions are added to *RefSet* and deleted from P. For each solution in *P-RefSet*, the minimum of the distances to the solutions in *RefSet* is computed. Then, the solution with the maximum of these minimum distances is selected. This solution is added to *RefSet* and deleted from P and the minimum distances are updated. The process is repeated b_2 times, where $b_2 = b - b_1$. The resulting reference set has b_1 high quality solutions and b_2 diverse solutions.

Solutions in *RefSet* are ordered according to quality, where the best solution is the first one in the list. The simplest form of the Subset Generation Method consists of generating all pairs of reference solutions. That is, the method would focus on subsets of size 2 resulting in $(b^2-b)/2$ of them when all solutions in the reference set are new (that is, they have not been combined). After the first iteration, the number of subsets generated depends on the number of new solutions admitted to the *RefSet*. The subsets are put in a list and then are selected one at a time in lexicographical order to apply the Solution Combination Method that generates one or more trial solutions. These trial solutions are subjected to the Improvement Method.

The Reference Set Update Method is applied once again. A typical and quite simple form of the application of this method is the so-called *static update*. Trial solutions that are constructed as combination of reference solutions are placed in a solution pool, denoted by *Pool*. After the application of both the Combination Method and the Improvement Method, the *Pool* is full and the reference set is updated. The new reference set consists of the best b solutions from the solutions in the current reference set and the solutions in the pool, i.e., the update reference set contains the best b solutions in $RefSet \cup Pool$.

If *RefSet* changes after the application of the Reference Set Update Method a flag indicates that at least one new solution has been inserted in the reference set. The procedure terminates after all subsets generated within the current iteration are subjected to the Combination Method and none of the improved trial solutions are admitted to *RefSet* under the rules of the Reference Set Update Method.

2.1 Advanced Designs

In the basic design, the new solutions that become members of *RefSet* are not combined until all eligible pairs are subjected to the Combination Method. The new reference set is built with the best solutions in the union of *Pool* and the solutions currently in *RefSet*. The alternative to this static update is the **Dynamic Update** strategy, which applies the Combination Method to new solutions in a manner that combines new solutions faster than in the basic design. That is, if a new solution is admitted to the reference set, the goal is to allow this new solution to be subjected to the Combination Method as quickly as possible. Instead of waiting until all the combinations have been performed to update the reference set, if a new trial solution warrants admission in the reference set, the set is immediately updated before the next combination is performed. Therefore, there is no need for an intermediate pool in this design, since solutions are either discarded or become part of the *RefSet* as soon as they are generated.

RefSet Rebuilding is another advanced design that is triggered when no new trial solutions are admitted to the reference set. This update adds a mechanism to partially rebuild the reference set when the Combination and Improvement Methods do not provide solutions of sufficient quality to displace current reference solutions. The *RefSet* is partially rebuilt with a diversification update that works as follows and assumes that the size of the reference set is $b = b_1 + b_2$. Solutions $x^{b_1+1},\ldots,x^{b}$ are deleted from *RefSet*. The Diversification Generation Method is reinitialized considering that the goal is to generate solutions that are diverse with respect to the reference solutions $x^{1},\ldots,x^{b_1}$. Then, the Diversification Generation Method is used to construct a set P of new solutions. The b_2 solutions $x^{b_1+1},\ldots,x^{b}$ in *RefSet* are sequentially selected from P with the criterion of maximizing the diversity. It is usually implemented with a distance measure defined in the context of the problem being solved. Then, maximum diversity is achieved by maximizing the minimum distance. The max-min criterion, which is part of the Reference Set Update Method, is applied with respect to solutions $x^{1},\ldots,x^{b_1}$ when selecting solution x^{b_1+1}, then it is applied with respect to solutions $x^{1},\ldots,x^{b_1+1}$ when selecting solution x^{b_1+2}, and so on.

Solution Combination Methods in scatter search typically are not limited to combining just two solutions and therefore the **Subset Generation Method** in its more general form consists of creating subsets of different sizes. The scatter search methodology assures that the set of combined solutions may be produced in its entirety at the point where the subsets of reference solutions are created. Therefore, once a given subset is created, there is no merit in creating it again. This creates a situation that differs noticeably from those considered in the context of genetic algorithms, where the combinations are typically determined by the spin of a roulette wheel.

The procedure for generating subsets of reference solutions uses a strategy to expand pairs into subsets of larger size while controlling the total number of subsets to be generated. In other words, the mechanism avoids the extreme type of process that creates all the subsets of size 2, then all the subsets of size 3, and so on until reaching the subsets of size *b*-1 and finally the entire *RefSet*. This approach clearly would not be practical, considering that there are 1013 subsets in a reference set of a typical size $b = 10$. Even for a smaller reference set, combining all possible subsets is not effective because many subsets will be almost identical. The following approach selects representative subsets of different sizes by creating subset types:

- Subset Type 1: all 2-element subsets.
- Subset Type 2: 3-element subsets derived from the 2-element subsets by augmenting each 2-element subset to include the best solution not in this subset.
- Subset Type 3: 4-element subsets derived from the 3-element subsets by augmenting each 3-element subset to include the best solutions not in this subset.
- Subset Type 4: the subsets consisting of the best i elements, for i = 5 to b.

We point out that an earlier proposal for generating subsets of different sizes using tabu search mechanisms, which apparently remains unexplored, appears in Glover (1994).

3. THE TRAINING METHOD

In this section we describe Laguna and Martí (2000) adaptation of scatter search to the neural network training problem. A pseudo-code of the algorithm is outlined in Figure 2 in which we refer to *w* as a solution to the training problem.

1. Normalize input and output data.
2. Start with $P = \emptyset$. Use the diversification method to construct a solution w between *wlow* and *whigh*. If $w \notin P$ then add w to P (i.e., $P = P \cup w$), otherwise, discard w. Repeat this step until $|P| = PSize$. Apply the improvement method to the best $b/2$ solutions in P to generate $w^{(1)}, \ldots, w^{(b/2)}$. Generate $b/2$ more solutions, where $w^{(b/2+i)} = w^{(i)}(1 + U[-0.3,0.3])$ for $i = 1, \ldots, b/2$. Build $RefSet = \{ w^{(1)}, \ldots, w^{(b)} \}$.
3. Order *RefSet* according to their objective function value such that $w^{(1)}$ is the best solution and $w^{(b)}$ the worst.

while (*NumEval* < *TotalEval*) **do**

3. Generate *NewPairs*, which consists of all pairs of solutions in *RefSet* that include at least one new solution. Make *NewSolutions* = ∅.

for (all *NewPairs*) **do**

4. Select the next pair ($w^{(i)}, w^{(j)}$) in *NewPairs*.
5. Obtain new solutions w as linear combinations of ($w^{(i)}, w^{(j)}$) and add them 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 $g(w) < g(w^{(b)})$) **then** make $w^{(b)} = w$ and reorder *RefSet*.

end for

while (*IntCount* < *IntLimit*)

7. Make *IntCount* = *IntCount* + 1 and $w = w^{(1)}$.
8. Make $w = w\,(1 + U[-0.05,0.05])$ and apply improvement method.

if ($g(w) < g(w^{(1)})$) **then** Make $w^{(1)} = w$ and *IntCount* = 0.

end while

9. Apply the improvement method to $w^{(i)}$ for $i = 1, \ldots, b/2$ in *RefSet*.
10. Make $w^{(b/2+i)} = w^{(i)}(1+U[-0.01,0.01])$ for $i = 1, \ldots, b/2$ in *RefSet*.

end while

Figure 8-2. Outline of training procedure

The procedure starts with the input and output data normalization. After this normalization, an initial reference set (*RefSet*) of b solutions is created. A set P of *PSize* solutions w (bounded between *wlow* and *whigh*) is built with the following diversification method, based on a controlled randomization scheme (Glover, Laguna and Martí, 2000). The range (*wlow*, *whigh*) is subdivided into 4 sub-ranges of equal size. Then, a solution w is constructed in two steps. First a sub-range is randomly selected. The probability of selecting a sub-range is inversely proportional to its frequency count. Then a value is randomly generated within the selected sub-range. The number of times sub-range j has been chosen to generate a value for w_i is accumulated in the frequency counter $freq(i, j)$. The main goal of the diversification generator is to create solutions that are diverse with respect to those solutions that have been already generated in the past. That is, the frequency counts work as "seed solutions" from which the diversification attempts to move away.

The reference set *RefSet* is filled with the best $b/2$ solutions in P to which an improvement method is applied (see below). The *RefSet* is completed with $b/2$ more solutions generated as perturbations of the first $b/2$. The perturbation consists of multiplying each weight by $1 + U[a,b]$, where $U[a,b]$

is the uniform distribution with parameters *a* and *b*. In step 2, the solutions in *RefSet* are ordered according to quality, where the best solution is the first one in the list. In step 3, the *NewPairs* set is constructed. *NewPairs* consists of all the new pairs of solutions that can be obtained from *RefSet*, where a "new pair" contains at least one new solution. Since all the solutions are new in the initial *RefSet*, the initial *NewPairs* consists of (b^2-*b*)/2 pairs. The pairs in *NewPairs* are selected one at a time in lexicographical order to create linear combinations in step 5. We consider the following three types of linear combinations, where we assume that the reference solutions are $w^{(i)}$ and $w^{(j)}$, and *r* is a random number in the range (0, 1):

C1: $w = w^{(i)} - d$

C2: $w = w^{(i)} + d$ $\qquad d = r\,\dfrac{w^{(i)} - w^{(j)}}{2}$

C3: $w = w^{(j)} + d$

The following rules are used to generate solutions with these three types of linear combinations:

- If $i \le b/2$ and $j \le b/2$ then 4 solutions are generated by applying C1 and C3 once and C2 twice.
- If $i \le b/2$ and $j > b/2$ then 3 solutions are generated by applying C1, C2 and C3 once.
- If $i > b/2$ and $j > b/2$ then 2 solutions are generated by applying C2 once and randomly choosing between applying C1 or C3.

The solutions created as linear combinations of solutions in the reference set are added to the *NewSolutions* set. Once all combinations have been made, the best *b* solutions in *NewSolutions* are subjected to the improvement method in step 6. Each improved solution *w* is then tested for admission into *RefSet*. If a newly created solution improves upon the worst solution currently in *RefSet*, the new solution replaces the worst and *RefSet* is reordered.

The procedure now intensifies the search around the best-known solution. In step 7, the counter *IntCount* is increased and the best solution is copied to a temporary memory location *w*. The solution is perturbed and the improvement method is applied in step 8. The best solution is updated if the perturbation plus improvement generates a better solution. When the best solution is improved, the intensification count *IntCount* is reset. If *IntLimit* intensification iterations are performed without improving the best solution, the procedure abandons the intensification phase.

Finally, steps 9 and 10 operate on the entire *RefSet*. Step 9 applies the improvement method to the best $b/2$ solutions in *RefSet* and step 9 replaces the worst $b/2$ solutions with perturbations of the best $b/2$. The training procedure stops when the number of objective function evaluations (*NumEval*) reaches the total allowed (*TotalEval*). Note that the evaluation of the objective function $g(w)$ consists of the calculation of the mean squared error.

The SS procedure employs the well known Nelder and Mead (1965) optimizer as an improvement method. Given a set of weights w, the Nelder and Mead method starts by perturbing each weight to create an initial simplex from which to begin the local search. We use the implementation of the Nelder-Mead method in Press, et al. (1992) with the default parameters for 500 iterations. Note that this improvement method is used in three different situations during the search: (1) to improve upon the best $b/2$ solution in the initial *RefSet*, (2) to improve upon the b best solution that result from the linear combinations, (3) to improve upon the perturbed solutions generated during the intensification, and (3) to improve upon the $b/2$ best solutions when rebuilding *RefSet* in steps 9 and 10.

4. COMPUTATIONAL EXPERIMENTS

For our computational testing, we implemented, in C, the classical Back-Propagation method (BP), the extended tabu search method, ETS, of Sexton et al. (1998) and the SS method described in the previous section (SS). We run these three methods to approximate the 15 test functions introduced in El-Fallahi and Martí (2003). This test set includes well known multimodal functions such as the Six-Hump-Camelback, the Goldstein or the Schwefel instances. After preliminary experimentation we set the number of hidden neurons equal to 9 (and we keep this parameter fixed for all the functions).

The main goal in the design of an ANN is to obtain a model which makes good predictions for new inputs (i.e. to provide a good generalization). The standard way to measure how well this goal is accomplished consists of introducing an additional set of points in the domain of f called the validation set V. We assume that no point in V belongs to E (the training set) and $f(x)$ is known for all $x \in V$. Once the optimization has been performed and the weights w^* that minimize E have been set, the validation set V is presented to the resulting network and the corresponding errors are computed ($error(T,w^*)$). The network must exhibit a good fit between the target f-values and the output (prediction) in the training set and also in the validation set. In our experiments, 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 additional observations drawn from the same uniform distributions.

Tables 1 and 2 report, respectively, the training and validation errors obtained with the three methods when applied to the 15 problems considered. In order to obtain statistically valid results, we ran 20 times each method on each function and report the average and standard deviation of the 20 runs (each run is limited to ten minutes). In all the cases, we have considered the same training and validation sets.

Table 8-1. Training errors for three different training methods

Problem	BP	ETS	SS
1	1.60 ± 0.26	0.04 ± 0.02	0.00 ± 0.00
2	8.32 ± 4.30	1.79 ± 0.78	0.05 ± 0.03
3	1.63 ± 0.21	0.34 ±0.03	0.4 ± 0.19
4	45.52 ± 7.82	17.66 ±6	0.2 ± 0.08
5	12.62 ± 3.87	18.98 ±5.26	0.27 ± 0.26
6	13.98 ± 1.58	53.28 ±3.94	1.98 ± 0.29
7	16.09 ± 5.80	63.26 ±1.18	0.55 ± 0.01
8	0.20 ± 0.06	0.01 ±0.00	0.07 ± 0.03
9	7.35E+09±1.07E+09	3.30E+09±8.44E+07	4.95E+04±5.01E+03
10	21.40 ± 1.49	22.22± 4.12	4.6 ± 0.22
11	5.28E+06±1.34E+06	4.17E+06±1.28E+05	1.03E+03±1.07E+02
12	107.95 ± 3.01	156.12±5.57	0.1 ± 0.05
13	3.93 ± 1.97	10.13 ± 3.25	0.02 ± 0.1
14	5.58E+0 ± 6.76E+03	4.44E+04±2.48E+03	3.95E+04±2.01E+03
15	2.88 ± 0.5	527.14±3.07	0.39± 0.02

Table 8-2. Validation errors for three different training methods

Problem	BP	ETS	SS
1	1.77 ± 1.6	0.05 ± 0.02	0.00 ± 0.14
2	8.59 ± 3.94	2.13 ± 0.95	0.00 ± 0.12
3	1.68 ± 0.22	0.48 ± 0.05	0.44 ± 0.43
4	43.89 ± 9.86	21.71 ± 7.49	8.76 ± 26.13
5	14.30 ± 5.50	19.85 ± 6.19	1.06 ± 2.42
6	15.32 ± 0.87	51.54 ± 6.07	106.11 ± 221.11
7	21.56 ± 11.97	60.62 ± 5.31	52.04 ± 137.21
8	0.19 ± 0.06	0.04 ± 0.0	0.01 ± 0.02
9	1.22E+10±1.24E+04	7.03E+09±5.38E+08	3.3049E+10±5.04E+10
10	13.67 ± 0.02	16.59 ± 0.57	107.24 ± 192.39
11	4.13E+06±4.02E+03	6.51E+06±6.02E+05	5.70E+07±1.33E+08
12	111.17 ± 6.93	149.31 ± 7.22	0.03 ± 0.04
13	5.25 ± 2.56	10.32 ± 0.60	0.10 ± 0.70
14	5.64E+04±2.62E+03	4.33E+04±1.94E+03	1.30E+07± 1.30E+07
15	2.93 ± 0.53	554.68 ± 22.76	275.41±173.12

This experiment shows that none of the methods can effectively handle problems 9, 11 and 14 within the run time considered. (We also ran the

training methods in these three instances for about half an hour of CPU time with no significant improvement). Therefore, we can say that in practice the neural network configuration that we tested is not able to approximate these functions.

Considering the average values over the 20 runs, Table 1 shows that the SS method is able to obtain the best solutions with respect to the training error in 13 instances, while the ETS method obtains 2 best solutions. Table 2 shows similar results, with SS obtaining 10 best solutions with respect to the testing error and ETS obtaining 5 (including instances 9, 11 and 14). The results of the BP method are quite acceptable considering its simplicity.

5. CONCLUSIONS

In this chapter we have described the implementation of scatter search for training a single-layer feed-forward neural network. The focus of scatter search leads to a number of implications associated with the design of improved optimization procedures. These research 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 often is fashionable in evolutionary methods. Our goal was to develop a training procedure to obtain neural networks capable of providing high-quality predictions. Our experiments show that the scatter search implementation reaches a prediction accuracy that in most cases is acceptable. The experimentation also shows that the SS implementation improves upon the results found with a recently developed tabu search method.

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).

6. REFERENCES

El-Fallahi, A. and Martí, R., 2003. Tabu and Scatter Search for Training Neural Networks, Computational Modeling and Problem Solving in the Networked World, Hemant K. Bhargava and Non Ye (Eds.), Interfaces in Computer Science and Operations Research**,** 79-96. Kluwer Academic Publishers.

Glover F., 1963. Parametric Combinations of Local Job Shop Rules. ONR Research Memorandum no. 117, GSIA, Carnegie Mellon University, Pittsburgh, PA.

Glover, F., 1977. Heuristics for Integer Programming Using Surrogate Constraints, Decision Sciences, vol. 8, pp. 156-166.

Glover, F., 1994. Tabu Search for Nonlinear and Parametric Optimization (with Links to Genetic Algorithms), Discrete Applied Mathematics, vol. 49, pp. 231-255.

Glover, F., 1998. A Template for Scatter Search and Path Relinking. In: Hao, J.-K., Lutton, E., Ronald, E., Schoenauer, M., Snyers, D. (Eds.), Artificial Evolution, Lecture Notes in Computer Science 1363, Springer, pp. 13-54.

Glover, F., 2004. Parametric Tabu Search for Mixed Integer Programming, Technical Report, University of Colorado at Boulder (downloadable from http://spot.colorado.edu/~glover/Recentpapers.html).

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

Glover, F., Laguna, M. and Martí, R., 2000. Fundamentals of Scatter Search and Path Relinking, Control and Cybernetics, vol. 29, no. 3, pp. 653-684.

Laguna, M. and Martí R., 2000. Neural Network Prediction in a System for Optimizing Simulations, IIE Transaction, vol. 34, no. 3, pp. 273-282.

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

Sexton, R. S., B. Alidaee, R. E. Dorsey and J. D. Johnson, 1998. Global Optimization for Artificial Neural Networks: A Tabu search Application, European Journal of Operational Research, vol. 106, pp. 570-584.