Shortest Common Superstring Problem with Discrete Neural Networks

D. López-Rodríguez and E. Mérida-Casermeiro

Department of Applied Mathematics, University of Málaga, Málaga, Spain
{dlopez,merida}@ctima.uma.es

Abstract. In this paper, we investigate the use of artificial neural networks in order to solve the Shortest Common Superstring Problem. Concretely, the neural network used in this work is based on a multivalued model, MREM, very suitable for solving combinatorial optimization problems. We describe the foundations of this neural model, and how it can be implemented in the context of this problem, by taking advantage of a better representation than in other models, which, in turn, contributes to ease the computational dynamics of the model. Experimental results prove that our model outperforms other heuristic approaches known from the specialized literature.

1 Introduction

Many problems in computational biology, such as DNA sequencing [1,2,3,4], and in data compression [5,6], can be formulated as instances of the Shortest Common Superstring Problem (SCSS).

For example, DNA sequencing consists in determining the correct sequence of nucleotides in a DNA molecule. Nucleotides (adenine, cytosine, guanine and thymine) are represented by the alphabet {a,c,g,t}. Currently, the nucleotides of a DNA fragment can be directly determined in laboratories. Once the nucleotides of all fragments have been determined, the sequence assembly problem aims at reconstructing the original molecule from overlapping fragments. SCSS can be viewed as an abstract representation of this particular task.

This problem is defined as follows: Given a set of strings $P = \{s_1, \ldots, s_N\}$, the objective is to find the string S^* of minimum length, such that, for all $i \in \{1, \ldots, N\}$, $s_i \in P$ is a substring of S^*.

Finding such a superstring is known to be a NP-hard problem [7,8]. Furthermore, it is MAX-SNP-hard [9]. Arora, in [10] proved that problems in MAX-SNP-hard do not admit polynomial time approximation schemes unless P=NP.

Among the approximation algorithms used to compute the shortest superstring, we can find a greedy algorithm [9], which consists in merging pairs of strings with maximum overlap, until a unique string is obtained, which is the approximated solution.

This greedy approach is conjectured to be a 2-approximation algorithm [11], meaning that, in the worst case, the solution provided by this method is twice as long as the optimal solution. However, Blum et al. [9] proved a factor of 4 for this problem.

This greedy algorithm was further improved by Jiang [12] (within a factor of $2\frac{2}{3}$) and later by Sweedyk [13] (obtaining a constant factor of 2.5), the best result up-to-date. However, these algorithms are not easily implementable and, in practice, the original greedy algorithm (reliable and fast), proposed in [9], is preferred, as well as some of its variants.

In this work, we propose the use of a discrete neural network to solve SCSS problem. In the recent years, a Hopfield-like discrete neural network [14] has been presented to solve this problem. However, its use implies the correct fine-tuning of some parameters. Another drawback of that model is that it needed N^2 neurons to represent the solution of the problem when the number of strings to be merged is $|P| = N$.

The neural model proposed in this work is a generalization of Hopfield's discrete model [15], allowing the neurons to take any value in a discrete set. With the help of a simple computational dynamics, this model is able to represent solutions to this problem better than the previous neural approach, just by using N neurons.

The multivalued MREM model has obtained very good results when applied to other combinatorial optimization problems [16,17,18,19], guaranteeing the convergence to local minima of the energy function.

The rest of this paper is structured as follows: in Sec. 2, we present a detailed formulation of the SCSS problem. Later, in Sec. 3, the neural model MREM is described, as well as the implementation of the computational dynamics to solve the problem at hand. In Sec. 4, experimental results of applying our neural model are shown, whereas in Sec. 5 some conclusions and remarks to this work are presented.

2 Description of the Problem

Given an alphabet A, and a set of strings over the alphabet, $P = \{s_1, \ldots, s_n\}$, the Shortest Common Superstring Problem consists in finding a string S^* containing all strings in P as substrings and with minimum length.

Let us define the overlap $s_{i,j}$ between strings s_i and s_j (in this order), as the string of maximum length (denoted by $|s_{i,j}|$) such that it is a suffix for s_i and a prefix for s_j.

The solution to SCSS can be represented as a permutation Π of numbers $\{1, \ldots, n\}$, meaning the order in which strings in S must be arranged to get the solution string $S^* = S_\Pi$.

Thus, the objective function to be minimized is:

$$|S_\Pi| = F(\Pi) = \sum_{i=1}^{n} |s_i| - \sum_{i=1}^{n-1} |s_{\Pi(i),\Pi(i+1)}| \quad (1)$$

where $|s|$ denotes the length of string s. Note that $s_{\Pi(i),\Pi(i+1)}$ is the overlap between 2 consecutive strings in S_Π, corresponding to strings at positions $\Pi(i)$ and $\Pi(i+1)$.

The minimization of the total length of S_Π is here achieved by maximizing the sum of the lengths of the respective overlaps in the corresponding order given by permutation Π.

Note that the solution may not be unique:

Example. Let us consider the set of strings $P = \{\text{agcct}, \text{acgcgt}, \text{cgtacg}, \text{tgatc}, \text{gtgag}\}$ over the alphabet $A = \{\text{a}, \text{c}, \text{g}, \text{t}\}$. Then, $S_1 =$ cgtacgcgtgagcctgatc and $S_2 =$ tgatcgtacgcgtgagcct are superstrings containing all strings in P, of equal length.

3 The MREM Model

In this section, the fundamentals of the Multivalued REcurrent Model (MREM) [20] are described. This discrete neural network is a generalization of Hopfield's model [21,15] and other binary and multivalued models, such as SOAR [22] and MAREN [23].

3.1 Description of the Neural Network

Let us consider a recurrent neural network formed by N neurons, where the state of each neuron $i \in \mathcal{I} = \{1, \ldots, N\}$ is defined by its output v_i taking values in any finite set $\mathcal{M} = \{m_1, m_2, \ldots, m_L\}$. This set does not need to be numerical. For example, $\mathcal{M} = \{\text{red}, \text{green}, \text{blue}\}$ or $\mathcal{M} = \{\text{Sunday}, \text{Monday}, \ldots, \text{Saturday}\}$.

The vector V whose components are the corresponding neuron outputs, $V = (v_1, v_2, \ldots, v_N)$, is called state vector. Associated to each state vector, an energy function, similar to Hopfield's, can be defined:

$$E(V) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{i,j} f(v_i, v_j) + \sum_{i=1}^{N} \theta_i(v_i) \tag{2}$$

where

- $W = (w_{i,j})$ is the synaptic weight matrix, expressing the connection strength between neurons.
- $f\colon \mathcal{M} \times \mathcal{M} \to \mathbb{R}$ is the so-called similarity function, since $f(v_i, v_j)$ measures the similarity between the outputs of neurons i and j.
- $\theta_i\colon \mathcal{M} \to \mathbb{R}$ is the generalization of the biases $\theta_i \in \mathbb{R}$, present in Hopfield's model.

The aim of the network is to minimize the energy function given by Eq. (2), i.e., to achieve a stable state corresponding to a local (global, when possible) minimum of the energy function, which is usually identified with the objective function of the problem to solve.

The introduction of the similarity function f makes the network very versatile and usually causes a better representation of the problem at hand, see, for example, [24,18,25,26]. It leads to a better representation of problems than other multivalued models, as SOAR and MAREN [23,22], since in those models most of the information enclosed in the multivalued representation is lost by the use of the signum function that only produces values in $\{-1, 0, 1\}$.

Many computational dynamics can be defined for this model, that is, several neuron updating schemes are available provided the versatility of the network.

Usually, neuron updates are made by taking into consideration the input to the network, called synaptic potential. This potential is computed as $U = -\Delta E$, that is, the opposite of the energy increase produced by the studied neuron update. Thus, if E is the current energy value, and E' is the energy value associated to the proposed update, then $U = E - E'$.

If several possible updates $\{V_1, \ldots, V_K\}$ are studied, consider $U_j = E - E'_j$, where E'_j is the energy value associated to the possible new state V_j. In this case, the update is given by the new state achieving the maximum potential $u = U_j = \max\{U_1, \ldots, U_K\}$.

If $u > 0$, then the update reduces the value of the energy function. Otherwise, since no improvement is obtained by that update, the network does not perform the action. In this situation, the network is said to have converged to a stable state.

Stable states correspond to local minima of the energy function, in the sense that, by using the given dynamics, it is not possible to achieve a further improvement of the solution.

3.2 MREM Applied to SCSS

Note that a solution to SCSS problem can be represented as a permutation of the strings, meaning the order in which strings have to be merged to obtain that solution.

Then, we define feasible state vectors as those representing permutations of $\{1, \ldots, n\}$. Thus, any feasible state vector V will represent an ordering of the strings in S. $v_i = k$ means that s_k is placed in the i-th place in the solution string s_V.

It can be observed that the objective function in Eq. (1) for SCSS consists of two terms. The first one, $\sum_i |s_i|$ is fixed, and therefore it is not important at the optimization stage.

The other term, $-\sum_{i=1}^{n-1} |s_{\Pi(i),\Pi(i+1)}|$, can be expressed as the energy function of the MREM model.

By comparing the objective function in Eq. (1) and the energy function of the neural model, in Eq. (2), we can define:

$$\begin{aligned} w_{i,j} &= \begin{cases} 2, \text{ if } j = i+1, i = 1, \ldots, n-1 \\ 0, \qquad\qquad \text{otherwise} \end{cases} \\ f(x,y) &= |s_{x,y}| \\ \theta_i(x) &= 0 \end{aligned}$$

to obtain the desired identification between both functions.

The computational dynamics of this model is based on that the network must remain in a feasible state along iterations. This is the reason for not needing the fine-tuning of parameters, usually present in Hopfield's energy function, as penalty terms for unsatisfied constraints. Furthermore, it is an easily implementable dynamics, and it can be described as follows:

1. Select a random initial feasible state for the network.
2. The net sequentially selects 2 neurons m and p such that $1 \leq m < p \leq N$. Then, the current solution V can be expressed as the concatenation of 3 subsequences, represented by 3 vectors $a = (v_1, \dots, v_m)$, $b = (v_{m+1}, \dots, v_p)$ and $c = (v_{p+1}, \dots, v_N)$.
3. The network studies the updates to different configurations: acb, bac, bca, cab, cba, where

$$acb = (v_1, \dots, v_m, v_{p+1}, \dots, v_N, v_{m+1}, \dots, v_p)$$
$$bac = (v_{m+1}, \dots, v_p, v_1, \dots, v_m, v_{p+1}, \dots, v_N)$$
$$bca = (v_{m+1}, \dots, v_p, v_{p+1}, \dots, v_N, v_1, \dots, v_m)$$
$$cab = (v_{p+1}, \dots, v_N, v_1, \dots, v_m, v_{m+1}, \dots, v_p)$$
$$cba = (v_{p+1}, \dots, v_N, v_{m+1}, \dots, v_p, v_1, \dots, v_m)$$

   by computing the corresponding synaptic potentials:

$$U_{abc} = 0 \quad \text{(since there is no change in state vector)}$$
$$U_{acb} = |s_{v_m,v_{p+1}}| + |s_{v_N,v_{m+1}}| - |s_{v_m,v_{m+1}}| - |s_{v_p,v_{p+1}}|$$
$$U_{bac} = |s_{v_p,v_1}| + |s_{v_m,v_{p+1}}| - |s_{v_m,v_{m+1}}| - |s_{v_p,v_{p+1}}|$$
$$U_{bca} = |s_{v_p,v_{p+1}}| + |s_{v_N,v_1}| - |s_{v_m,v_{m+1}}| - |s_{v_p,v_{p+1}}| = |s_{v_N,v_1}| - |s_{v_m,v_{m+1}}|$$
$$U_{cab} = |s_{v_N,v_1}| + |s_{v_m,v_{m+1}}| - |s_{v_m,v_{m+1}}| - |s_{v_p,v_{p+1}}| = |s_{v_N,v_1}| - |s_{v_p,v_{p+1}}|$$
$$U_{cba} = |s_{v_N,v_{m+1}}| + |s_{v_p,v_1}| - |s_{v_m,v_{m+1}}| - |s_{v_p,v_{p+1}}|$$

   These expressions are derived from $U = E - E'$, being E the energy associated to the current network state, and E' the energy associated to the corresponding update.
4. The next network configuration is the one decreasing most the energy function value (equivalently, achieving the greatest potential): if U_{ijk} is the maximum in $\{U_{abc}, U_{acb}, U_{bac}, U_{bca}, U_{cab}, U_{cba}\}$, then the next state is $ijk \in \{abc, acb, bac, bca, cab, cba\}$. If $ijk = abc$, there is no change in the state vector.
5. Repeat steps 2 - 4 until convergence is detected, that is, all pairs of neurons have been studied, and no change is done in the configuration of the network (state vector).

Once the network converges, the stable state represents a minimum of the energy function which, in our case, is equivalent to a maximum of the aggregate overlap length in the resulting string, given by S_V.

Algorithm 1. Greedy Heuristic

```
Data: Set $P = \{s_1, \ldots, s_N\}$ of strings.
Result: A string $s$ such that every $s_i$ is a substring of $s$ (intended to have
        minimal length).
begin
    while $|P| > 1$ do
        Select two strings $a, b \in P$ with maximal overlap
        Merge $a$ and $b$ into a new string $c$
        $P \longleftarrow (P \setminus \{a, b\}) \cup \{c\}$
    return the unique string $s \in P$.
end
```

4 Experimental Results

In this section, we compare the efficiency of our neural model MREM to the greedy heuristic presented in [9], which was conjectured to be a 2-approximation algorithm. This greedy heuristic is shown in Algorithm 1.

Two experiments have been performed with these algorithms. The first one consisted on find the SCSS of a set of strings of fixed length, whereas the second allowed to use strings of variable length.

Fixed length string datasets were randomly built according to 3 parameters: string length ({6,8,10}), number of words in P ($|P| \in \{25, 50, 100\}$) and number of symbols in the alphabet ($|A| \in \{2, 4, 6, 8\}$). For each combination of these parameters, 10 instances were built (that is, 10 sets P), and the algorithms were independently run 100 times to obtain the superstring length results given in Table 1. Note that the greedy algorithm always selected the same solution, whereas MREM achieves different results depending on its random initial state, what helps avoiding local optima of the energy function.

In the last two columns of the tables, we present the improvement made by MREM with respect to the greedy algorithm (in %). Positive values indicate that MREM performed better than the greedy. Note that our neural approach outperformed the greedy algorithm in most cases on average, and always on best result.

For variable length string datasets, the definition of $|P|$ and $|A|$ remain the same, but string length was randomly selected in $\{2, \ldots, 10\}$. Thus, for each value of $|A|$, $|P|$ strings of length between 2 and 10, formed the set P. As before, for each combination of the parameters, 10 sets P were built and 100 independent executions were performed with each one. Superstring length results are given in Table 2. Note that, in all cases, MREM outperformed the greedy algorithm, obtaining shorter superstrings, not only on minimal length, but also on average length.

There are 2 behaviors that can be seen on these tables:

– As the number $|A|$ of symbols in the alphabet increases (for a fixed number of strings, $|P|$), MREM and the greedy algorithm tend to obtain more similar results, reducing the improvement made by MREM over the latter.

Table 1. Average and minimum superstring length comparison between our neural proposal and the greedy algorithm, for fixed length strings

			MREM			Greedy		Improvement	
Length	$\|P\|$	$\|A\|$	Average	Best	Time (sec.)	Best	Time (sec.)	Average	Best
6	25	2	67.169	63.1	0.0216	91.5	0.0256	26.59	31.04
		4	106.065	103.1	0.0211	113	0.0234	6.14	8.76
		6	121.516	118.3	0.0217	120.1	0.0239	-1.18	1.5
		8	126.033	123.7	0.0209	124.1	0.0237	-1.56	0.32
	50	2	100.276	92.9	0.1442	168.7	0.091	40.56	44.93
		4	195.324	189	0.1342	209.5	0.0917	6.77	9.79
		6	225.096	219.8	0.1336	236	0.0912	4.62	6.86
		8	238.884	234	0.1447	249.2	0.092	4.14	6.1
	100	2	146.332	136.3	0.9982	326.7	0.4456	55.21	58.28
		4	362.473	352.2	0.7874	383.5	0.4063	5.48	8.16
		6	418.712	409.1	0.8418	424.1	0.4063	1.27	3.54
		8	451.846	443.8	0.8881	458	0.4118	1.34	3.1
8	25	2	112.69	108.1	0.0251	127.9	0.0272	11.89	15.48
		4	158.908	155.7	0.0204	158.9	0.257	-0.01	2.01
		6	170.554	167.8	0.0221	174.3	0.025	2.15	3.73
		8	178.423	176.1	0.0206	179.6	0.0258	0.66	1.95
	50	2	180.117	171.6	0.149	237.6	0.0954	24.19	27.78
		4	291.753	284.8	0.1399	305.1	0.0961	4.37	6.65
		6	323.997	318.6	0.1391	332.1	0.0965	2.44	4.07
		8	339.505	334.9	0.1338	346.1	0.0944	1.91	3.24
	100	2	287.94	275	0.9774	439.7	0.4076	34.51	37.46
		4	547.072	536.8	0.8547	567.1	0.414	3.53	5.34
		6	617.981	609.9	0.8469	633.1	0.4253	2.39	3.66
		8	650.429	642.8	0.9	656.7	0.4273	0.95	2.12
10	25	2	155.939	151	0.0234	176.7	0.0258	11.75	14.54
		4	207.935	204.6	0.0219	215.3	0.0259	3.42	4.97
		6	220.99	218.2	0.0207	220	0.0261	-0.45	0.82
		8	226.615	224.4	0.0206	229.6	0.0262	1.3	2.26
	50	2	284.153	274	0.1614	318.6	0.0984	10.81	14
		4	396.893	390.5	0.1396	399	0.0986	0.53	2.13
		6	425.848	420.9	0.134	432.4	0.0995	1.52	2.66
		8	439.814	434.8	0.1449	445.6	0.0999	1.3	2.42
	100	2	465.955	450.5	0.955	581.9	0.4389	19.93	22.58
		4	740.486	729.7	0.8943	770.4	0.428	3.88	5.28
		6	818.364	809.9	0.8752	834.6	0.4318	1.95	2.96
		8	853.481	845.4	0.8965	867	0.4371	1.56	2.49

– As the number of strings in P increases (for fixed number $|A|$), MREM improves its relative performance with respect to the greedy algorithm. Thus, in real-world problems, MREM may achieve better results than the greedy algorithm.

Table 2. Average and minimum superstring length comparison between our neural proposal and the greedy algorithm, for variable length strings

		MREM			Greedy		Improvement	
$\|P\|$	$\|A\|$	Average	Best	Time (sec.)	Best	Time (sec.)	Average	Best
25	2	66.004	62.2	0.0189	77.3	0.0293	14.61	19.53
25	4	101.112	98.5	0.0186	105.7	0.024	4.34	6.81
25	6	109.799	107.3	0.0204	110.4	0.0238	0.54	2.81
25	8	109.829	107.5	0.0216	111.5	0.0238	1.5	3.59
50	2	117.959	110.4	0.1303	142.9	0.0929	17.45	22.74
50	4	181.495	176.1	0.1302	189.2	0.0935	4.07	6.92
50	6	198.456	192.9	0.1382	210.1	0.0972	5.54	8.19
50	8	225.345	220.8	0.1448	230.7	0.0954	2.32	4.29
100	2	201.824	190.8	0.9567	253.8	0.436	20.48	24.82
100	4	351.522	342	0.8342	382	0.4214	7.98	10.47
100	6	388.724	380.6	0.8386	405.2	0.4205	4.07	6.07
100	8	421.852	414.4	0.8033	435.6	0.4248	3.16	4.87

5 Conclusions and Future Work

In this work, a neural model, MREM, is presented to solve the Shortest Common Superstring problem. This problem arises in real-world applications coming from molecular genetics (DNA sequencing) and data compression.

The neural model MREM is a generalization of Hopfield's model. Its main feature is that neuron states can be selected from a discrete set $\mathcal{M} = \{m_1, \ldots, m_L\}$, instead of taking value in {-1,1} or {0,1}. This fact makes the network represent combinatorial optimization problems more easily.

A neural dynamics has been developed and implemented to solve the problem at hand, taking advantage of the representation of a solution as a permutation of the indices of the strings to be merged.

We have tested our approach by comparing it to a greedy algorithm, well-known from the specialized literature. In our results, MREM proved to outperform the greedy algorithm in most cases. It may be of great help in tackling real-world SCSS instances.

As a future work, we plan to:

– Develop a parallel version of the computational dynamics presented in this paper, in order to reduce the computational time used to achieve the solution.
– Introduce some mechanism to avoid local optima of the objective function. The hybridization of MREM with other stochastic techniques (Genetic Algorithms, Simulated Annealing) may be helpful.
– Make a theoretical study on the behavior of this new neural algorithm, in order to confirm the improvement over the greedy algorithm.

Acknowledgements

This work is partially supported by Junta de Andalucía (Spain) under contract TIC-01615, project name Intelligent Remote Sensing Systems.

Authors also wish to thank Prof. Gabriela Andrejková, from University of Kosice, Slovakia, for introducing the Shortest Common Superstring problem to them.

References

1. Ilie, L., Popescu, C.: The shortest common superstring problem and viral genome compression. Fundamenta Informaticae 73(1,2), 153–164 (2006)
2. Lesk, A.: Computational Molecular Biology, Sources and Methods for Sequence Analysis. Oxford University Press, Oxford (1988)
3. Li, M.: Towards a dna sequencing theory (learning a string). In: Proc. 31st Annual Symposium on Foundations of Computer Science, pp. 125–134 (1990)
4. Peltola, H., Soderlund, H., Tarhio, J., Ukkonen, E.: Algorithms for some string matching problems arising in molecular genetics. In: Proc. IFIP Congress, pp. 53–64 (1983)
5. Daley, M., McQuillan, I.: Viral gene compression: complexity and verification. In: Domaratzki, M., Okhotin, A., Salomaa, K., Yu, S. (eds.) CIAA 2004, vol. 3317, pp. 102–112. Springer, Heidelberg (2005)
6. Storer, J.: Data Compression: Methods and Theory. Computer Science Press, Rockville (1988)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability. In: Garey, M.R., Johnson, D.S. (eds.) A guide to the theory of NP-Completeness, W. H. Freeman and Company, New York (1979)
8. Maier, D., Storer, J.: A note on the complexity of the superstring problem. In: Proceedings of the 12th Annual Conference on Information Science and Systems, pp. 52–56 (1978)
9. Blum, A., Jiang, T., Li, M., Tromp, J., Yannakakis, M.: Linear approximation of shortest superstring. Journal of the ACM 41(4), 630–647 (1994)
10. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and hardness of approximation problems. In: 33rd Annual Symposium on Foundations of Computer Science, pp. 14–23 (1992)
11. Turner, J.: Approximation algorithms for the sortest common superstring problem. Information and Computation 83(1), 1–20 (1989)
12. Jiang, T., Jiang, Z., Breslauer, D.: Rotation of periodic strings and short superstrings. In: Proc. 3rd South American Conference on String Processing (1996)
13. Sweedyk, Z.: A $2\frac{1}{2}$-approximation algorithm for shortest superstring. SIAM Journal of Computing 29, 954–986 (1999)
14. Andrejkov, G., Levick, M., Oravec, J.: Approximation of shortest common superstring using neural networks. In: Proc. of 7th International Conference on Electronic Computers and Informatics, pp. 90–95 (2006)
15. Hopfield, J., Tank, D.: Neural computation of decisions in optimization problems. Biological Cybernetics 52, 141–152 (1985)
16. Mérida-Casermeiro, E., Galán-Marín, G., Muñoz-Pérez, J.: An efficient multivalued hopfield network for the travelling salesman problem. Neural Processing Letters 14, 203–216 (2001)

17. Mérida-Casermeiro, E., Muñoz-Pérez, J., Domínguez-Merino, E.: An n-parallel multivalued network: Applications to the travelling salesman problem. In: Mira, J., Álvarez, J.R. (eds.) IWANN 2003. LNCS, vol. 2686, pp. 406–413. Springer, Heidelberg (2003)
18. Mérida-Casermeiro, E., López-Rodríguez, D.: Graph partitioning via recurrent multivalued neural networks. In: Cabestany, J., Prieto, A.G., Sandoval, F. (eds.) IWANN 2005. LNCS, vol. 3512, pp. 1149–1156. Springer, Heidelberg (2005)
19. López-Rodríguez, D., Mérida-Casermeiro, E., Ortiz-de-Lazcano-Lobato, J.M., López-Rubio, E.: Image compression by vector quantization with recurrent discrete networks. In: Kollias, S.D., Stafylopatis, A., Duch, W., Oja, E. (eds.) ICANN 2006. LNCS, vol. 4132, pp. 595–605. Springer, Heidelberg (2006)
20. Mérida-Casermeiro, E.: Red Neuronal recurrente multivaluada para el reconocimiento de patrones y la optimización combinatoria. Ph. D thesis, Universidad de Málaga (2000)
21. Hopfield, J.: Neural networks and physical systems with emergent collective computational abilities, vol. 79, pp. 2254–2558 (1982)
22. Ozturk, Y., Abut, H.: System of associative relationships (soar) (1997)
23. Erdem, M.H., Ozturk, Y.: A new family of multivalued networks. Neural Networks 9(6), 979–989 (1996)
24. Mérida, E., Muñoz, J., Benítez, R.: A recurrent multivalued neural network for the N-queens problem. In: Mira, J., Prieto, A.G. (eds.) IWANN 2001, vol. 2084, pp. 522–529. Springer, Heidelberg (2001)
25. López-Rodríguez, D., Mérida-Casermeiro, E., Ortiz-de-Lazcano-Lobato, J.M., Galán-Marín, G.: *k*-pages graph drawing with multivalued neural networks. In: de Sá, J.M., Alexandre, L.A., Duch, W., Mandic, D.P. (eds.) ICANN 2007. LNCS, vol. 4669, pp. 816–825. Springer, Heidelberg (2007)
26. Galán-Marín, G., Mérida-Casermeiro, E., López-Rodríguez, D.: Improving neural networks for mechanism kinematic chain isomorphism identification. Neural Processing Letters 26, 133–143 (2007)