*Neural Processing Letters* **4:** 17–27, 1996.
© 1996 *Kluwer Academic Publishers. Printed in the Netherlands.*

# An Energy Function for the Random Neural Network

JOSE AGUILAR C.
*Dpto. de Computación, Facultad de Ingeniería, Universidad de los Andes, Av Tulio Febres, 5101 Mérida, Venezuela*
*E-mail: aguilar@ing.ula.ve*

**Key words:** energy function, optimization problems, random neural network

**Abstract.** Since Hopfield's seminal work on energy functions for neural networks and their consequence for the approximate solution of optimization problems, much attention has been devoted to neural heuristics for combinatorial optimization. These heuristics are often very time-consuming because of the need for randomization or Monte Carlo simulation during the search for solutions. In this paper, we propose a general energy function for a new neural model, the random neural model of Gelenbe. This model proposes a scheme of interaction between the neurons and not a dynamic equation of the system. Then, we apply this general energy function to different optimization problems.

## 1. Introduction

Since the seminal papers of the early eighties [1, 2], the study of emergent collective properties of artificial neural networks has created an exciting area for research. For instance, it is well known that for the Hopfield network with symmetric weights, as well as for other models, each individual state change of the networks has the effect of reducing an appropriately defined energy function [1]. This elementary but subtle observation has spawned a large body of work on using neural networks to provide heuristic solutions to computationally intractable or very difficult optimization problems. This is usually achieved by designing a Hopfield (or other appropriate neural) network whose energy function mimics a cost function which embodies the optimization problem to be solved.

In 1989, Gelenbe modeled the neural network using an analogy with queuing theory [2–4]. This model does not use a dynamic equation, but rather a scheme of interaction among neurons. It calculates the probability of activation of the neurons in the network. Signals in this model take the form of impulses which mimic what is presently known of interneural signals in biophysical neural networks.

The random neural network (RNN) has been used in solution optimization [5, 6] and recognition problems [6]. In [7] a supervised learning procedure for the recurrent RNN model is proposed which is mainly based on the minimization of a quadratic error function. In [6, 8], we have explored the relationship between the RNN model applied to optimization and network learning. Recently, we have applied the evolutionary learning the RNN model [9].

In this paper, we propose a general energy function for the RNN. Then, this general energy function on different optimization problems. This work is organized as follows: in Section 2, the theoretical basis of the RNN is reviewed. Then, we present the general energy function. In section 4 we present the energy function for two NP-hard problems (Graph Partitioning and Minimum Node Covering). Remarks concerning future work and conclusion are provided in Section 5.

## 2. The Random Neural Network Model

The RNN model consists of a network of $n$ neurons in which positive and negative signals circulate. Each neuron accumulates signals as they arrive, and can fire if its total signal count at a given instant of time is positive. Firing then occurs at random according to an exponential distribution of constant rate, and signals are sent out to other neurons or to the outside of the network. Each neuron $i$ of the network is represented at any time $t$ by its input signal potential $k_i(t)$.

Positive and negative signals have different roles in the network. A negative signal reduces by 1 the potential of the neuron at which it arrives (inhibition) or has no effect on the signal potential if it is already zero; while an arriving positive signal adds 1 to the neuron potential.

Signals can either arrive at a neuron from the outside of the network or from other neurons. Each time a neuron fires, a signal leaves it, depleting the total input potential of the neuron. A signal which leaves neuron $i$ heads for neuron $j$ with probability $p^+(i,j)$ as a positive signal (excitation), or as negative signal with probability $p^-(i,j)$ (inhibition), or it departs from the network with probability $d(i)$. Clearly we shall have:

$$\sum_{j=1}^{n} [p^+(i,j) + p^-(i,j) + d(i) = 1 \quad \text{for } 1 \leq i \leq n.$$

External positive signals arrive to the $i^{\text{th}}$ neuron according to a Poisson process of rate $\Lambda(i)$. External negative signals arrive to the $i^{\text{th}}$ neuron according to a Poisson process of rate $\lambda(i)$. The rate at which neuron $i$ fires is $r(i)$. The main property of this model is the excitation probability of a neuron $i, q(i)$, which satisfies a non-linear equation:

$$q(i) = \lambda^+(i)/(r(i) + \lambda^-(i)) \tag{1}$$

where,

$$\lambda^+(i) = \sum_{j=1}^{n} q(j)r(j)p^+(j,i) + \Lambda(i)$$

$$\lambda^-(i) = \sum_{j=1}^{n} q(j)r(j)p^-(j,i) + \lambda(i)$$

## 3. A General Energy Function for the Random Neural Network

In the RNN, $q(i)$ depends on $\Lambda(i)$, $\lambda(i)$, $p^+(j, i)$, $p^-(j, i)$, $r(i)$ and the other $q(j)$'s. In the optimization, $p^+(j, i)$, $p^-(j, i)$ and $r(i)$ are fixed and depend on the nature of the combinatorial problem. Besides, in the optimization problem the relationship between two neurons is competitive or cooperative, that is either $p^+(j, i)$ or $p^-(j, i)$ is null. Of course, if there is no interaction between them, then both $p^+(j, i)$ and $p^-(j, i)$ are null. On the other hand, emission of external signals is not interesting for optimization, it is better to employ the signals to inhibit or to excite the neighbor neurons, that is, $d(i)$ is null. The firing rate $r(i)$ is obtained by the reciprocity of effect between neurons. When two neurons $i$ and $j$ are excited and $i$ emits signals to $j$, the excitation or inhibition that $i$ exerts over $j$ must be the same as the excitation or inhibition that $i$ receives.

If $p^+(j, i)$, $p^-(j, i)$ and $r(i)$ are fixed, the only way to lead the network from one stationary state to another one is to act over the inputs. This state of the RNN model is defined by $(q(i), ..., q(n))$. The use of two external flows to every neuron permits a complex scaling of an external positive flow to an external negative flow [6, 8]. In optimization, the use of two flows is not interesting. We consider $\lambda(i)$ as null so that the neurons only receive external positive signals, representing the preference that the neuron belongs to the solution. In this way, $q(i)$ and $\Lambda(i)$ become the variables of the RNN model. The general form of the energy function proposed is:

$$E = \textstyle\sum_{i<j} a_{ij}q(i)q(j) + \sum_i a_{ii}q(i)^2 + \sum_i b_i q(i) + c \qquad \forall\, i, j = 1, ..., n \tag{2}$$

where $a_{ij}$, $b_i$ and $c$ are the parameters of the optimization problem. It is interesting to see how this energy function definition differs from the classical approach of Hopfield. Note the additional terms which are squared in one state variable and linear in the other. Therefore, the above energy function can correspond to a quadratic cost function. Our reference to a quadratic energy function is motivated by the 'usual' formulation of optimization problems with neural networks.

Now, we search to define a dynamic of external positive signals in the RNN model, in order to find the state that gives the minimal energy in the network. Using the technique of gradient descent, the dynamic of the external excitation signal is defined as:

$$\Lambda(u)^{m+1} = \Lambda(u)^m - \mu \;\; [\partial E/\partial \Lambda(u)]^m \quad \text{in the } m^{\text{th}} \text{ iteration} \tag{3}$$

Equation (3) describes the control that must be applied to the system to minimize the energy function. The general procedure that we propose for the RNN is [6]:

- Initialize $\Lambda(i)$ in some appropriate manner
- Repeat

- Solve Equation (1)
- Using (3) and the previous results, update $\Lambda(i)$.
- If $\Lambda(i)$ is outside of $[0, r(i)]$, replace for the nearest bounds until the change in the new value of $q(i)$ is smaller than some predetermined value.

Thus,

$$\begin{aligned}\partial E/\partial\Lambda(u) &= \sum\nolimits_{i,j}\{(2a_{ii}q(i)+b_i)1[i=j]+(a_{ij}1[j>i]\\ &+ a_{ji}1[j<i])\,q(j)\}\partial q(i)/\partial\Lambda(u)\end{aligned}$$

Given,

$$X_{ij} = (2a_{ii}q(i)+b_i)1[i=j]+(a_{ij}1[j>i]+a_{ji}1[j<i])\,q(j)$$

then,

$$\partial E/\partial\Lambda(u) = \sum\nolimits_{i\neq j} X_{ij}\partial q(i)/\partial\Lambda(u)$$

Now, we must explain $\partial(i)/\partial\Lambda(u)$ using the stationary solution of the network. Given:

$$\begin{aligned}N(i) &= \sum\nolimits_j w^+_{ji}q(j)+\Lambda(i)\\ D(i) &= \sum\nolimits_j w^-_{ji}q(j)+\sum\nolimits_j(w^+_{ij}+w^-_j)\end{aligned}$$

where,

$$\begin{aligned}&w^+_{ji} = r(j)p^+(j,i),\\ &w^-_{ji} = r(j)p^-(j,i),\\ &r(i) = \textstyle\sum_j(w^+_{ij}+w^-_{ij})\end{aligned}$$

then,

$$\begin{aligned}\partial q(i)/\partial\Lambda(u) &= \left[\sum\nolimits_j w^+_{ji}\partial q(j)/\partial\Lambda(u)+1[i=u]\right]/D(i)\\ &- \left[\sum\nolimits_j w^-_{ji}\partial q(j)/\partial\Lambda(u)\right] N(i)/D(i)^2\end{aligned}$$

Given:

$$Y_{ij} = w^{+}_{ji}/D(i) - w^{-}_{ji}N(i)/D(i)^2$$
$$\partial q(i)/\partial\Lambda(u) = \textstyle\sum_j Y_{ij}\partial q(j)/\partial\Lambda(u) + 1[i = u]/D(u)$$

That is,

$$\partial q/\partial\Lambda(u) = \partial q/\partial\Lambda(u) \cdot C + e_{u*}1/D(u)$$
$$\text{with } e_{ui} = 1\,[i = u] \text{ and } \mathrm{C} = \left(\textstyle\sum_j Y_{ij}\right)$$

finally,

$$\partial q/\partial\Lambda(u) = 1/D(u)_{*}e_{u}.[I - C]^{-1}$$

## 4. Our Energy Function on Different Optimization Problems

### 4.1. The Graph Partitioning Problem

The problem consists in dividing a graph into several subgraphs, so as to minimize a given cost function [1]. In a very general way, to place the problem on a mathematical formulation, the following definition is necessary: $\prod = (N, A)$ where $\prod$ is a directed graph, $N$ is a set of $n$ nodes, $A = ad_{ij}$ are node pairs that define the arcs (it is known as the adjacency matrix).

The problem consists in dividing the graph into $K$ different subgraphs $\prod = \{\prod_1, ..., \prod_K\}$, according to a certain cost function. The cost function associates a real value to every subgraph configuration. We propose the cost function:

$$F = \sum_{i,j\in D} ad_{ij} + b\left(\sum_{k=1}^{K}(N_{\Pi_k} - {}^{n}/_{K})^2\right)/K \tag{4}$$

where $D = \{i \in \prod_k \,\&j \in \prod_l \&l \neq k\}$

The first term minimizes the edges which belong to the cut. The second summation term will have a minimum value only when the number of nodes in the partitions are the same. The balance factor (b) defines the importance of the interconnection cost with respect to the imbalance cost, and $N_{\Pi}k$ is the number of nodes in $\Pi_k\ \forall\, k = 1, ..., K$. The graph partitioning problem is reduced to finding a subgraph configuration with the minimum value for the cost function.

#### 4.1.1. *Our RNN for This Problem*

In this approach, we will construct an RNN composed of $nK + K$ neurons. For each (node, subgraph) pair $(i, u)$ we will have a neuron $\mu(i, u)$ whose role is to 'decide' whether node i should be assigned to subgraph u. We will denote by $q(\mu(i, u))$ the probability that $\mu(i, u)$ is excited: thus, if it is close to 1, we will be encouraged to assign $i$ to $u$. In order to reduce connections between subgraphs in the selected partition, $\mu(i, u)$ will tend to *excite* any neuron $\mu(j, u)$ if $j$ is connected to $i$, and will tend to *inhibit* $\mu(j, v)$ if $j$ is connected to $i$ and $u \neq v$. Similarly, $\mu(i, u)$ will *inhibit* $\mu(j, v)$, $\forall_{v=1,\ldots,K}$, if $j$ is not connected to $i$. On the other hand, neurons $\mu(i, u)$ and $\mu(i, v)$, $u \neq v$, will *inhibit* each other so as to indicate that the same node should not be assigned to different subgraphs.

For each subgraph $u$ we will have a neuron $\pi(u)$ whose role is to let us know whether $u$ is heavily loaded with node or not. If $u$ is very heavily loaded, it will attempt to reduce the load on subgraph $u$ by *inhibiting* neurons $\mu(i, u)$, and it will attempt to increase the load on subgraphs $v \neq u$ by *exciting* neurons $\pi(v)$. In the same way, $\mu(i, u)$ will *excite* neuron $\pi(u)$ to increase the load on subgraph $u$. The parameters of the RNN expressing these intuitive criteria are chosen as follows:

- $\Lambda(\mu(i, u))$ = random, - $\lambda(\mu(i, u)) = 0$,
- $\Lambda(\pi(u)) = n/K$, to express the desirable equal load sharing property,
- $\lambda(\pi(u)) = 0$, - r$(\mu(i, u)) = nK$
- r$(\pi(u)) = n + K - 1$
- r$(\mu(i, u))p^+(\mu(i, u), \mu(j, v)) = 1$ if $(ad_{ij}= 1$ or $ad_{ji} = 1)$ and $u = v$, 0 otherwise.
- r$(\mu(i, u))p^-(\mu(i, u), \mu(j, v)) = 1$ if $(u \neq v$ and $(ad_{ij} = 1$ or $ad_{ji} = 1$ or $i = j))$, or $(ad_{ij} = 0$ and $ad_{ji} = 0)$, 0 otherwise.
- r$(\mu(i, u))p^+(\mu(i, u), \pi(v)) = 1$ if $u = v$, 0 otherwise.
- r$(\pi(u))p^-(\pi(u), \mu(i, u)) = 1$ if $q(\pi(u)) \sim 1$, 0 otherwise.
- r$(\pi(u))p^+(\pi(u), \pi(v)) = 1$ if $q(\pi(u)) \sim 1$, 0 otherwise.

Equation (1) for this case is:

$$\begin{aligned}
&q(\mu(i,u)) = \\
&\left\{\sum\nolimits_{(ad_{ij}=1 \text{ or } ad_{ji}=1)} q(\mu(j,u))r(\mu(j,u))p^+(\mu(j,u),\mu(i,u))\right\} / \\
&\left\{r(\mu(i,u)) + \sum\nolimits_{v\neq u} \sum\nolimits_{(ad_{ij}=1 \text{ or } ad_{ji}=1 \text{ or } i=j)}\right. \\
&q(\mu(j,v))r(\mu(j,v))p^-(\mu(j,v),\mu(i,u)) + \\
&\sum\nolimits_{v} \sum\nolimits_{(ad_{ij}=0 \,\&\, ad_{ji}=0)} q(\mu(j,v))r(\mu(j,v))p^-(\mu(j,v),\mu(i,u)) + \\
&\left. q(\pi(u))r(\pi(u))p^-(\pi(u),\mu(i,u))\right\} \qquad (5)
\end{aligned}$$

$$q(\pi(u)) =$$
$$\{\Lambda(\pi(u)) + \sum_{j=1}^{n} q(\mu(j,u))r(\mu(j,u))p^{+}(\mu(j,u),\pi(u)) +$$
$$\sum_{v=1}^{K} q(\pi(v))r(\pi(v))p^{+}(\pi(v),\pi(u))\}/r(\pi(u))$$

For this problem, using the general energy function (2) and the cost function (4), we propose the following energy function:

$$E = \sum_{i,j \in D} ad_{ij}q(\mu(i,k))q(\mu(j,l)) +$$
$$b\left(\sum_{k=1}^{K}\left(\sum_{i=1}^{n} q(\mu(i,k)) - {}^{n}/_{K}\right)^{2}\right)/K + \sum_{i=1}^{n}\left(\sum_{k=1}^{K} q(\mu(i,k)) - 1\right)^{2}$$

If we develop this function, we obtain the following values for $a_{ij}$, $b_i$ and $c$:

$$\begin{aligned} a_{ij} &= ad_{ij} && \text{if} i < j \\ & b/k + 1 && \text{if} i = j \\ & 0 && \text{otherwise} \\ b_i &= -2(nb/K^2 + 1) \\ c &= bn^2/K^3 + 1 \end{aligned}$$

### 4.1.2. *Performance Evaluation*

We compare the RNN with the approximate heuristics proposed in [6]: genetic algorithms (GA), simulated annealing (SA) and Kernighan's heuristic (Kern). The random graphs used are defined for the average number of nodes ($n$) and the average degree of the successor nodes of a node ($d$). For each graph, the successors of a node are chosen randomly from a uniform distribution in the interval $[1, d]$. The execution time is in seconds.

The parameters of the simulations are the following: the total number of subgraphs ($K$), the mean number of nodes per graph ($n$), the mean number of successors per node ($d$) and the balance factor ($b$). We generate 50 random graphs for the set of parameters where $n = 10, 20, 50$, $K = 2$ and $d = 2$.

We obtain the optimum solutions using an enumerative search algorithm. We study the following performance criteria:

- $T_1$, the mean value of the computation time on a workstation for each heuristic.
- Sop$_1$, the mean value of the solutions for a given set of parameters.
- $\delta_1$, the percentage of cases where a heuristic obtains the optimum solution.

Table I. Performance criteria for $l = 10$, $n = 20$, $d = 2$, $b = 1$ and $K = 2$.

| Method | $E$ | $\delta$ | $T$ | Sop |
|---|---|---|---|---|
| SA | 0.06 | 0.9 | 28 | 2.6 |
| GA | 0.12 | 0.8 | 19 | 2.8 |
| RNN | 0.12 | 0.8 | 9 | 2.8 |
| Kern | 1.06 | 0.15 | 3 | 4.1 |

– $E_l$, the mean relative error of the solutions of a heuristic compared to the optimum solution,

$$E_l = \sum_{i,l} (S_{il} - S_i^{opt})/S_i^{opt}$$

where $l$ ($l = 1, ..., 10$) is the number of times that we execute the heuristic to obtain these values on the graph $i$ ($i = 1, ..., 50$), $S_i^{opt}$ is the optimum solution of the graph $i$ and $S_{il}$ is the solution of the heuristic for the graph $i$. Due to space limitations, the results presented in this section were chosen because they are representative of the phenomena studied.

Sop and $\delta$ are approximately the same for every heuristic, but Kernighan's heuristic gives the worst results. SA is the heuristic with the least mean relative error, but we obtain interesting results with RNN with a short execution time.

## 4.2. Minimum Node Covering Problem

The formal problem can be stated as follows. Let $G$ be a graph with $n$ nodes $\mathbf{N} = \{V_1, ..., V_n\}$ and edges denoted by $(V_i, V_j)$. A cover of $G$ is a subset $\mathbf{S}$ of $\mathbf{N}$ such that for each edge $(V_i, V_j)$ in $G$, either $V_i$ or $V_j$ are in $\mathbf{S}$. A minimum cover of $G$ is a set $\mathbf{S^*}$ such that the number of nodes in $\mathbf{S^*}$ is no larger than the number of nodes in any cover $S$ of $G$: $| \mathbf{S^*} | \leq | \mathbf{S} |$. The cost function for this problem is:

$$F = \sum_{i=1}^{n} \left\{ 2n\beta_i^* - [D(i)\beta_i^* - D(i)\beta_i] - \sum_{j=1}^{n} ad_{ji}\beta_i^*\beta_j + \beta_i\beta_i^* \right\} \quad (6)$$

where, $D(i)$ is the degree of node $i$.

$$\beta_i = 1 \quad \text{if node} V_i \in S \text{in this solution}$$
$$0 \quad \text{otherwise.}$$
$$\beta_i^* = 1 \quad \text{if node} V_i \in S \text{in this solution}$$
$$0 \quad \text{otherwise.}$$

The first term states that there should be as few nodes as possible in the minimum cover, while the second and third terms state that we should favour nodes which have a large degree. The fourth term states that we would like to have only one of each end node of an edge in the cover. In this way, this term eliminates illegal solutions. The last term states that we cannot have the same node both in and out of the cover. Though this cost function is quite elaborate, it does provide a more detailed representation of the problem's constraints. Let us now present a RNN for the minimum graph covering problem.

### 4.2.1. *Our RNN for This Problem*

Each node $i$ in the graph is represented by two neurons, $\mu(i)$ and $\pi(i)$. The first of these will be *'on'* if the network recommends that $i$ be included in the cover, while the second will have the opposite role. The parameters of the RNN expressing these intuitive criteria are chosen as follows:

- $\Lambda(\mu(i))$ = degree of node $i$ $(D(i))$,
- $\lambda(\mu(i)) = 0$, – $\Lambda(\pi(i))$ = random,
- $\lambda(\pi(i)) = 0$, – $r(\mu(i)) = 2n$
- $r(\pi(i))$ = degree of node $i$ $(D(i))$,
- $r(\mu(i))\, p^-(\mu(i), \pi(j)) = 2n$ if $(i = j)$, 0 otherwise.
- $r(\pi(i))\, p^+(\pi(i), \mu(j)) = 1$ if $(ad_{ij} = 1)$ or $(ad_{ji} = 1)$, 0 otherwise.
- All other weights are set to zero.

Equation (1) for this case is:

$$\begin{aligned} q(\mu(i)) &= \left\{\Lambda(\mu(i)) + \sum\nolimits_{j=1}^{n} q(\pi(j))r(\pi(j))p^+(\pi(j), \mu(i))\right\} / r(\mu(i)) \\ q(\pi(i)) &= \Lambda(\pi(i)) / \{r(\pi(i)) + q(\mu(i))r(\mu(i))p^-(\mu(i), \pi(i))\} \end{aligned} \tag{7}$$

For this problem, using the general energy function (2) and the cost function (6), we propose the following energy function:

$$\begin{aligned} E = \textstyle\sum_{i=1}^{n} \{&2nq(\mu(i)) - [D(i)q(\mu(i)) - D(i)q(\pi(i))] - \\ &\textstyle\sum_{j=1}^{n} ad_{ji}\, q(\mu(i))q(\pi(j)) + q(\mu(i))q(\pi(i))\} \end{aligned}$$

but, if we suppose that,

$$q(\mu(i)) = 1 - q(\pi(j)) \qquad \text{(ideal case)}$$

the energy function is:

$$\begin{aligned} E = \textstyle\sum_{i=1}^{n} \sum_{j=1}^{n} ad_{ij} q(\pi(i))q(\pi(j)) + \sum_{i=1}^{n} [2D(i) + 1 - \\ 2n - \textstyle\sum_{j=1}^{n} ad_{ji}] q(\pi(i)) - \sum_i q^2(\pi(i)) + 2n - \sum_i D(i) \end{aligned}$$

Table II. Performance criteria.

| Method | $\delta$ | Exc | $T$ |
|---|---|---|---|
| n = 20 | $\sigma + 0.5$ | | |
| Heur | 72 | 0.3 | 2 |
| RNN | 100 | 0 | 6 |
| n = 50 | $\sigma = 0.125$ | | |
| Heur | 12 | 1.5 | 1 |
| RNN | 80 | 0.2 | 12 |

If we develop this function, we obtain the following values for $a_{ij}$, $b_i$ and $c$:

$$\begin{aligned} a_{ij} &= ad_{ij} && \text{if} i \neq j \\ &-1 && \text{otherwise} \\ b_i &= 2D(i) + 1 - 2n - \textstyle\sum_{j=1}^{n} ad_{ji} \\ c &= 2n + \textstyle\sum_i D(i) \end{aligned}$$

4.2.2. *Performance Evaluation*

We compare the RNN with an approximate heuristic (Heur) proposed in [5] and with the exact solution. The number of nodes in the graphs has been varied with the following values: $n = 20, 50$. Once the size of the graphs is fixed at a value, the random generation is carried out as follows. A probability value $s$ is fixed; this is the probability that for any pair of nodes $V_i$, $V_j$ there is an edge ($V_i$, $V_j$). We have taken different values of $s = 0.125$, 0.5 and of $n$. These results are present in Table II. For each value ($n$, $\sigma$) we generated 25 graphs at random in this manner, and the results which are tabulated below are average values over this set. These heuristics are compared with respect to the following criteria:

– $\delta$ is the percentage of cases where a heuristic obtains the minimum cover.
– Exc is the average number of nodes in excess of the minimum for the set of 25 graphs, for a given ($n$, $\sigma$) pair.
– $T$ is the mean value of the computation time on a workstation for each heuristic.

The results clearly show that the RNN heuristic easily outperforms the greedy heuristic in all cases, and that this performance improvement is particularly substantial when the graph is sparse (i.e., small $\sigma$).

## 5. Conclusion

The purpose of this paper has been to consider the formulation of a general energy function to solve combinatorial optimization problems using the RNN. The major advantage of this model is that it has a purely numerical and computationally fast solution, which removes the need for complex search techniques and other Monte Carlo simulation-based optimization methods. The definition of a general energy function for the RNN permits us to define a dynamic to search for an optimum solution of a combinatorial optimization problem.

Then, we have illustrated the utilization of our general energy function on two optimization problems: the Graph Partitioning and the Minimum Node Covering. Further work will examine the results proved for this model using this energy function in other optimization problems.

## References

1. J. Hopfield and D. Tank. "Neural computation of decisions in optimization problem", Biological Cybernetics, Vol. 52, pp. 141–152, 1985.
2. E. Gelenbe. "Random neural networks with positive and negative signals and product form solution", Neural Computation, Vol. 1, No. 4, pp. 502–510, 1989.
3. E. Gelenbe. "Stable random neural networks", Neural Computation, Vol. 2, No. 2, pp. 239–247, 1990.
4. E. Gelenbe. "Theory of the random neural network model", in E. Gelenbe (ed) Neural Networks: Advances and Applications, North-Holland, pp. 1–20, 1991.
5. E. Gelenbe and F. Batty. "Application of the random neural network model to minimum graph covering", in E. Gelenbe (ed) Neural Networks: Advances and Applications 2, North Holland: pp. 36–45, 1992.
6. J. Aguilar. "L'allocation de tâches, l'équilibrage de la charge et l'optimisation combinatoire", PhD thesis, UFR Mathematique et Informatique, Rene Descartes University, France, 1995.
7. E. Gelenbe. "Learning in the recurrent Random Neural Network", Neural Computation, Vol. 5, No. 5, pp. 154–164, 1993.
8. J. Aguilar. "An approach for combinatorial optimization problem based on learning in the recurrent random neural network", Proceedings of the World Congress on Neural Networks, pp. 420–424, International Neural Network Society: San Diego, CA, USA, July 1994.
9. J. Aguilar. "Evolutionary Learning on Recurrent Random Neural Network", Proceedings of the World Congress on Neural Networks, pp. 232–236, International Neural Network Society: Washington, DC, USA, July 1995.