A Hybrid Neural Optimization Scheme Based on Parallel Updates

G. Papageorgiou, A. Likas and A. Stafylopatis

17-96

Preprint no. 17-96/1996

Department of Computer Science
University of Ioannina
45 110 Ioannina, Greece

A Hybrid Neural Optimization Scheme Based on Parallel Updates

G. Papageorgiou[1], A. Likas[2] and A. Stafylopatis[1]

[1] Department of Electrical and Computer Engineering, National Technical University of Athens, 157 73 Zographou, Athens, Greece.

[2] Department of Computer Science, University of Ioannina, 451 10 Ioannina, Greece.

Abstract

A synchronous Hopfield-type neural network model containing units with analog input and binary output, which is suitable for parallel implementation, is examined in the context of solving discrete optimization problems. A hybrid parallel update scheme concerning the stochastic input-output behaviour of each unit is presented. This parallel update scheme maintains the solution quality of the Boltzmann Machine optimizer, which is inherently sequential. Experimental results on the Maximum Independent Set problem demonstrate the benefit of using the proposed optimizer in terms of computation time. Excellent speedup has been obtained through parallel implementation on both shared memory and distributed memory architecures.

Keywords: Optimization, parallel computing, Boltzmann Machine, Cauchy Machine.

1 Introduction

The usual approach for solving a discrete optimization problem using neural network techniques, is to formulate the cost function and the constraints of the problem in terms of the minimization of a quadratic *energy function*, which is defined over the binary space $\{0,1\}^n$ and has the following form:

$$E(\vec{v}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} v_i v_j w_{ij} - \sum_{i=1}^{n} v_i \theta_i \tag{1}$$

It is assumed that $w_{ij} = w_{ji}$ for every $i = 1,\ldots,n$, $j = 1,\ldots,n$ and $w_{ii} = 0$ for every i.

The most widely studied neural network approaches that can be used for the minimization of the above energy function are based on the Boltzmann Machine [1, 2] (which extends the discrete Hopfield model [9]) and the analog Hopfield network [10].

The operation of the Boltzmann Machine is based on an integration of the dynamics of the discrete Hopfield model with the Simulated Annealing methodology [1]. At each step,

a unit i is randomly selected and the energy difference ΔE_i that would be caused by a change in the state of this unit is computed as follows

$$\Delta E_i = (2v_i - 1)(\sum_{j=1}^{n} w_{ij}v_j + \theta_i) \tag{2}$$

If $\Delta E_i < 0$ the change is accepted, otherwise it is accepted with probability that depends on the quantity $\exp(\Delta E_i/T)$, where the temperature parameter T decreases according to a specified annealing schedule.

The Boltzmann Machine optimizer is very effective when an appropriate annealing schedule is used. The proper mapping of problems onto the network, as well as efficient operation schemes can be very crucial for the performance of the model [11]. However, as is the general case with Simulated Annealing, it has the disadvantage of being strictly sequential: each unit is examined after the update of the previous one has been completed. Although the Simulated Annealing and Boltzmann Machine approaches are of inherently sequential nature, several attempts have been made to reduce the required computation time through the use of parallel machines [1, 4, 8, 13]. Parallel implementations can be obtained either on general purpose parallel machines (for example on Transputer-based machines [5, 6]) or using special purpose VLSI [3, 14, 15] or optical neuroprocessors [21]. The parallel schemes considered vary depending on the type of computation performed or the synchronization policy adopted, and are characterized by some sort of trade-off between solution quality and speedup. In general, attempts to parallelize the operation of the Boltzmann Machine are not very successful, since simultaneous update of many network units can lead to oscillations and low quality of solutions [1].

On the other hand, in the analog Hopfield neural network model all units are updated simultaneously at each time instant, but due to the existence of a sigmoid output function, the output v_i of each unit i can take any continuous value in the unit interval. This has the disadvantage that in many cases the network is trapped in local minimum states inside the unit hypercube and does not manage to reach one of the corners.

By applying a step threshold function to the analog Hopfield network in place of the typically used sigmoid function a type of Hopfield neural network is obtained which is capable of operating in a synchronous way. Each unit of this model has analog input and discrete output. Such a model has been considered in [18]. We shall refer to this model as the *synchronous discrete Hopfield* neural network. This characterization distinguishes the model from both the discrete Hopfield model [9], which is discrete but sequential, and the analog Hopfield model which is synchronous but not discrete [10]. The employment of

units with discrete outputs offers a more effective exploration of the binary problem state space in comparison to the case of analog outputs, since the search is confined only at the corners of the hypercube that has to be explored [7, 17, 18, 19, 20].

Moreover, integration of the synchronous discrete Hopfield network with a fast Simulated Annealing methodology [16] results in the Distributed Cauchy Machine [18], which incorporates Cauchy colour noise in the update process of each neuron, in order to provide stochastic hillclimbing capabilities and avoid convergence to false local minima. As a general purpose optimizer, the Cauchy Machine has the advantage of exhibiting full parallelism, but generally provides solutions of lower quality than the Boltzmann Machine optimizer. In this paper we propose a hybrid stochastic update scheme for the synchronous discrete Hopfield network that results from a combination of features of both the Cauchy and the Boltzmann Machine. The proposed scheme exhibits exploration performance analogous with that of the Boltzmann Machine optimizer. In terms of execution time it is clearly superior to the Cauchy Machine and outperforms the Boltzmann Machine if implemented in parallel.

The organization of the paper has as follows. In Section 2 results concerning the synchronous discrete Hopfield neural network and the Distributed Cauchy Machine are presented along with their basic features. In Section 3 the hybrid scheme is introduced, whereas Section 4 contains comparative experimental results from the solution of the Maximum Independent Set Problem. The parallel implementation of the proposed model is described in Section 5, and the main conclusions are summarized in Section 6.

2 The Cauchy Machine

The synchronous discrete Hopfield network is a fully parallel neural network model. Each network unit i is characterized by analog input u_i (taking any real value) and binary output v_i (taking values in $\{0,1\}$). The input-output behaviour of each unit at time t is based on the McCulloch-Pitts model of operation:

$$v_i(t) = \begin{cases} 1 & \text{if } u_i(t) > 0 \\ 0 & \text{if } u_i(t) \leq 0 \end{cases} \tag{3}$$

During operation, the value of u_i is updated at each time step using the following motion equation:

$$\frac{\Delta u_i}{\Delta t} = -\frac{\Delta E_i}{\Delta v_i} \tag{4}$$

where ΔE_i is the difference in the energy of the network that would be caused by a change in the output of unit i. Simulation of the dynamics is based on first-order discrete approximation: $u_i(t + \Delta t) = u_i(t) + \Delta u_i$.

Equation (4) means that each unit follows gradient descent dynamics which in general leads the synchronous discrete Hopfield network to an equilibrium state. In the case where the energy function is given by equation (1), the motion equation of each unit i is given by

$$\frac{\Delta u_i}{\Delta t} = \sum_{j=1}^{n} w_{ij} v_j + \theta_i \tag{5}$$

A binary vector $(v_1, \ldots, v_n)$ constitutes an equilibrium state of the synchronous discrete Hopfield network if for each $i = 1, \ldots, n$,

$$v_i = 1 \text{ and } \Delta u_i \geq 0$$
$$\text{or}$$
$$v_i = 0 \text{ and } \Delta u_i \leq 0.$$

The above conditions state in essence that once the network has reached an equilibrium state it will remain there forever, since no change is possible in the output of any unit. For a unit with $v_i = 1$ the value of u_i will keep on increasing and v_i will always be 1. For a unit with $v_i = 0$ the requirement $\Delta u_i \leq 0$ ensures that the value of u_i will always remain negative and hence v_i will always be 0.

Moreover it is easy to show that there exists a one-to-one correspondence between the stable states of a discrete Hopfield network (or Boltzmann Machine) and the equilibrium states of a synchronous discrete Hopfield network having the same set of weights. Indeed, let $v = (v_1, \ldots, v_n)$ be a stable state of a discrete (binary) Hopfield network. Then for each $i = 1, \ldots, n$ it holds that $\Delta E_i \geq 0$, where ΔE_i denotes the difference in the energy that will be caused in case the state of unit i is changed. According to equation (2) this implies that either $v_i = 1$ and $\sum_{j=1}^{n} w_{ij} v_j + \theta_i \geq 0$ or $v_i = 0$ and $\sum_{j=1}^{n} w_{ij} v_j + \theta_i \leq 0$. Taking into account equation (5), it is straightforward to establish the one-to-one correspondence between the stable states of the discrete Hopfield network and the equilibrium states of a synchronous discrete Hopfield network characterized by the same energy function.

The most important feature of the synchronous discrete Hopfield network is that the output of each unit i depends solely on the value of u_i and not on the difference in the network's energy ΔE_i as is the case with the discrete Hopfield network (or the Boltzmann Machine). The quantities u_i can be considered as a kind of 'memory' representing the cumulative activation of each unit during network operation. In order for two units to

change their state simultaneously, not only must the corresponding Δu_i have the appropriate sign, but, in addition, the signs of the values u_i should also change at the same time instant. Thus, although the network operates synchronously, the probability that many output updates be performed simultaneously is significantly reduced.

In order to provide the synchronous discrete Hopfield network with stochastic hill-climbing capabilities, the *Distributed Cauchy Machine* [18] has been developed, in which the output of each unit i is stochastically updated at each time step t using the Cauchy distribution:

$$s_i(t) = Pr\{v_i(t) = 1\} = \frac{1}{2} + \frac{1}{\pi} \arctan\left(\frac{u_i(t)}{T_C(t)}\right) \tag{6}$$

In the above equation, T_C is an artificial temperature parameter that is adjusted according to a fast annealing schedule [18] as a function of time t:

$$T_C(t) = \frac{T_0}{1 + \beta t} \tag{7}$$

where T_0 is the initial temperature value and β is a value in the range $(0, 1)$ that controls the speed of the schedule. If the value of T_C is equal to 0, we have the case of a deterministic synchronous discrete Hopfield network.

3 The Hybrid Update Scheme

In terms of optimality of the obtained solutions, the Boltzmann Machine is superior to the Cauchy Machine and exhibits greater flexibility during operation, in the sense that it is able to perform a wider search of the problem state space. The reason is twofold. First, since each unit of the Boltzmann Machine is updated sequentially, the decision about changing its state is made based on the correct energy difference. Second, in the case where a unit i of the Boltzmann Machine is found with $\Delta E_i \leq 0$, its state is updated immediately. On the contrary, in a similar situation, a unit i of the Cauchy Machine may not change its state due to the current value of its input u_i, which can change sign only after many time steps.

The above frequently encountered update delay can be both advantageous and disadvantageous. The benefit is that it does not allow many units to simultaneously change state (especially at low temperatures), thus allowing the Cauchy Machine to converge although it operates synchronously. The drawback is the lack of flexibility that makes difficult the acceptance of state changes suggested by the values Δu. The latter constitutes a serious problem especially in cases where a Cauchy Machine has operated for a long time and the

quantities u_i have attained high absolute values. Therefore, there is a trade-off between the necessity to accept a suggested state transition as soon as possible and the necessity to maintain the convergence property. To make such state transitions more probable and enhance the exploration capability of the network without sacrificing convergence, we have developed a hybrid operation scheme of the synchronous discrete Hopfield network, in which the stochastic update rule is based on a convex combination of the Cauchy Machine update rule involving u_i (equation (6)) and a criterion involving Δu_i or equivalently ΔE_i. As such we have considered the Metropolis acceptance criterion frequently employed in the Boltzmann Machine case.

According to the Metropolis criterion [12], if $\Delta E_i(t)$ is the change in the energy of the network caused by the change in the state of a unit i at time t, then the probability of accepting this state transition at a temperature $T_B(t)$ is given by

$$p_i^B(t) = \begin{cases} 1 & \text{if } \Delta E_i(t) < 0 \\ \dfrac{1}{1+\exp(\frac{\Delta E_i(t)}{T_B(t)})} & \text{if } \Delta E_i(t) \geq 0 \end{cases} \tag{8}$$

It is interesting to note that the energy difference is given by $\Delta E_i(t) = (2v_i(t)-1)\Delta u_i(t)/\Delta t$, thus, the computation of $\Delta E_i(t)$ does not impose any additional burden to the usual computations required for the operation of the Cauchy Machine.

On the other hand, according to the Cauchy acceptance criterion the probability that a state transition be accepted at temperature $T_C(t)$ is

$$p_i^C(t) = \begin{cases} s_i(t) & \text{if } v_i(t) = 0 \\ 1 - s_i(t) & \text{if } v_i(t) = 1 \end{cases} \tag{9}$$

where $s_i(t)$ is computed using equation (6).

The proposed update scheme is based on a combination of the above probabilities. More specifically, at each time step, a state transition concerning unit i is accepted with probability:

$$p_i^H(t) = \alpha p_i^C(t) + (1-\alpha)p_i^B(t) \tag{10}$$

where the control parameter α takes values in the interval $(0,1)$. A small value of this parameter can destroy the property of convergence since the network will behave in essence as a synchronous Boltzmann Machine. On the other hand, a large value of the parameter α results in fast convergence but the state space will not be adequately explored, since the network will behave as a typical Cauchy Machine. Thus, an appropriate adjustment of the value of parameter α is needed in order to combine the desirable characteristics of the two approaches without causing any negative effects.

The above stochastic update scheme increases the probability that the output of a unit i will be modified in case such a change is suggested by the value of ΔE_i, although the corresponding input u_i may have large absolute value and the corresponding probability p_i^C may be small. In order for this flexibility to be exploited, it is necessary that a change in the output be followed by a change in the sign of the input. Otherwise, the output would probably change back to its previous value at the next step, since the value of u_i would continue to suggest that value. For this reason, in order to enable the network to quickly follow transitions that lead to lower energy states, we have incorporated the following rule in the adjustment of the quantities u_i, which is applied only in the case where a change occurs in the output v_i:

$$\text{if } v_i(t+\Delta t) \neq v_i(t) \text{ and } p_i^C(t) < 0.25 \text{ then } u_i(t+\Delta t) = -u_i(t) \tag{11}$$

Another remark is that the temperature values T_B and T_C used in the computation of the probabilities p_i^B and p_i^C may not be the same and may be adjusted according to different annealing schedules. This is reasonable, especially in what concerns the initial annealing temperatures which generally depend on the average values of ΔE and u respectively, since, obviously, the latter quantities are not of the same order of magnitude. The proposed algorithm is summarized below.

- *Map problem on Hopfield Network (Boltzmann Machine).*
- *Initialize u_i and v_i for all units.*
- *Repeat steps 1-2 below until termination condition is reached (t denotes current time step).*
  1. *Update temperatures $T_C(t)$ and $T_B(t)$.*
  2. *For all units do:*
     - $\Delta u_i = \left(\sum_{j=1}^{n} w_{ij} v_j(t-1) + \theta_i\right) \Delta t$.
     - $u_i(t) = u_i(t-1) + \Delta u_i$.
     - *Compute $p^B(t)$ using equation (8).*
     - *Compute $p^C(t)$ using equation (9).*
     - *Compute $p^H(t)$ using equation (10).*
     - *Generate state $v_i(t)$ of unit i according to $p^H(t)$.*
     - *Change $u_i(t)$ if necesassary, according to the rule (11).*

4 Experimental Results

We tested the effectiveness of the proposed approach on instances of the Weighted Maximum Independent Set (MIS) problem and compared its performance with the performance of both the Boltzmann Machine and the pure Cauchy Machine. The Weighted Maximum Independent Set constitutes an important discrete optimization problem and many other problems (for example Set Partitioning, Set Packing, Set Covering etc. [22]) can be solved through the solution of the weighted MIS.

The formulation of the MIS problem (weighted case) is the following: Consider an undirected graph $G = (V, E)$ where V (with $|V| = n$) is the set of vertices and E denotes the set of edges. Let also A denote the adjacency matrix of graph G, i.e., $a_{ij} = 1$ if $(i, j) \in E$, otherwise $a_{ij} = 0$. An *independent set* V' of this graph is a subset of V that contains vertices not connected to each other. If $c : V \rightarrow \Re^+$ is a cost function assigning a cost to each vertex of the graph, the Maximum Independent Set problem is to find the independent set V' of maximum cost, where the cost $f_c(V')$ is defined as $f_c(V') = \sum_{k \in V'} c_k$.

A Hopfield-type neural architecture suitable for the MIS consists of n units with the following specification of weights w_{ij} and thresholds θ_i [1, 22]:

$$w_{ij} = \begin{cases} -\{\max\{\theta_i, \theta_j\} + \epsilon\} a_{ij} & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases} \tag{12}$$

$$\theta_i = c_i \tag{13}$$

where ϵ is a very small positive value (which is set equal to 0.5 in our experiments).

The above specification of the weights and thresholds ensures that, if a discrete Hopfield network (or Boltzmann Machine) is employed to solve the problem, the network will finally settle into an equilibrium state corresponding to an independent set of the graph. In addition, this set is maximal in the sense that no other vertex can be added to it without violating the disjointness constraint. Moreover, the resulting energy function is *order preserving* in the sense that the lower the final energy value, the better the cost of the final solution [1]. Due to the correspondence discussed in Section 2 it is obvious that the local minimum states of the Boltzmann Machine will also be equilibrium states of the corresponding synchronous discrete Hopfield network.

Experiments have been conducted on randomly generated graphs of sizes 200, 500, 1000, 1500 and 2000. For each graph size, 15 instances were created and tested on a Silicon Graphics Power Challenge machine with R8000 processors. A graph instance was constructed by determining with probability 0.1 whether an edge exists between each pair of

graph vertices. In addition, a cost value was assigned to each vertex which was a randomly selected integer in the range from 5 to 15. Since the generated graphs are sparse, each of them is characterized by many independent sets and finding the best one is generally difficult. For each instance 5 runs were carried out using different seed values for the random number generator.

For each generated graph instance experiments were performed using: a) the Boltzmann Machine optimizer, b) the pure Cauchy Machine and c) the hybrid model. In the last case three experiments were conducted with different parameter settings.

The application of the Boltzmann Machine to the solution of the Maximum Independent Set problem has been extensively examined in [1, 22]. It has been verified that high quality near-optimal solutions can be obtained, provided that an appropriate (slow) annealing schedule is employed. In our experiments with the pure Boltzmann Machine, we have used the logarithmic schedule suggested in [22], which prescribes the value of the temperature at the kth step of the annealing process as follows:

$$T_k = \frac{T_{k-1}}{1 + \log f(k)} \tag{14}$$

The function f has the form $f(k) = f(k-1)(1+r)$ with $f(0) = 1$, where r is the parameter that adjusts the speed of the schedule. All experiments were conducted starting from temperature $T_0 = 5.0$ and using a temperature reduction rate $r = 10^{-6}$. At each temperature, $2n$ units were randomly selected and examined, and the annealing was terminated if for $2n$ consecutive time steps no update was performed on the state of any of the examined units.

In what concerns the experiments with the Cauchy Machine, we have used the annealing schedule of equation (7), with $\beta = 1$ and starting temperature $T_0 = 2.0$. The time step Δt was very small (equal to 0.001) to ensure slow annealing. The annealing was terminated if for 2 consecutive time steps no change had occured in the output of any unit and, additionally, the equilibrium condition of Section 2 was valid. It must be noted here, that the number of 2 consecutive time steps in the case of the Cauchy Machine, where n units operate in parallel, corresponds to the number of $2n$ consecutive time steps in the case of the Boltzmann Machine, where only one unit is examined at a time.

As far as the hybrid model is concerned, the values of β, Δt and T_0, as well as the update of parameter T_C and the stopping criterion were similar to those used in the case of the pure Cauchy Machine. For the update of the Boltzmann temperature T_B, we have considered that at each step $T_B(t) = \lambda T_C(t)$, where λ is an adjustable parameter. In order

	Boltzmann Machine		*Cauchy Machine*	
n	Avg. Time	Avg. Cost	Avg. Time	Avg. Cost
200	52sec	417	1min 15sec	365
500	5min 30sec	571	15min	475
1000	22min	678	1h 22min	563
1500	51min	741	4h 11min	617
2000	1h 30min	787	8h 25min	649

Table 1: Comparative results for the Boltzmann and the Cauchy Machine

	$\alpha = 0.25$		$\alpha = 0.50$		$\alpha = 0.75$	
n	Avg. Time	Avg. Cost	Avg. Time	Avg. Cost	Avg. Time	Avg. Cost
200	85sec	416	1min	416	1min	391
500	9min	574	8min	558	9min	528
1000	45min	689	39min	671	45min	616
1500	2h	740	1h 45min	729	1h 50min	699
2000	4h	775	4h 10min	771	5h	734

Table 2: Comparative results for three configurations of the hybrid model

to determine the best setting of the parameter λ, we performed experiments with various combinations of the parameters λ and α, for many problem instances. In all cases, the best results were obtained for values of λ close to 5.0, therefore we have adopted this value in all experiments.

Tables 1 and 2 show performance results as a function of the size n of the problem. The displayed values are averages obtained from the experiments conducted for each value of the size. The results concern both the effectiveness (quality of the obtained solutions) and the efficiency (execution time).

The results shown in Table 1 concern the Cauchy and the Boltzmann Machine considered individually. It can be observed that the Boltzmann Machine is clearly superior in both the quality of the obtained solutions and the execution time in sequential implementation. This superiority becomes more apparent as the size of the problem (and hence the difficulty) increases. The difference in the quality of solutions was not reduced even when we used a slower cooling schedule and a higher initial temperature for the Cauchy Machine. Indeed, the Cauchy Machine is characterized by premature convergence that limits the possibility of improving the quality of the solutions, since it hardly accepts updates at

low temperatures.

The results of Table 2 make apparent the benefit of using the hybrid model, which yields solution quality equivalent to that of the Boltzmann Machine and much better than that of the Cauchy Machine. The best behaviour is obtained in the case $\alpha = 0.25$, that is characterized by 25% contribution of p_i^C and 75% of p_i^B in the computation of p_i^H.

The effectiveness of the hybrid model is analogous with that of the Boltzmann Machine, in spite of the fact that the network operates synchronously, which could be unfavourable for the convergence of the Boltzmann Machine part. It is worth noting, however, that if the value of α is further decreased oscillation phenomena are produced, since the synchronous operation of the Boltzmann Machine becomes dominant.

5 Parallel Implementation

Both the Cauchy Machine and the proposed hybrid scheme are characterized by a high degree of parallelism. As at each step all the units of the network are updated simultaneously, parallelism refers to the individual unit level. Consequently, we should need ideally one processor to simulate the operation of each unit (fine grain parallelism). Moreover, each processor should have immediate access to the data of any other processor. Since the computations performed by each unit are very simple, the model can be directly implemented in VLSI so as to take the greatest benefit. If general purpose hardware is used, then a shared memory architecture seems to be the most appropriate. Indeed, we observe that at each step all units read in parallel the state vector $\vec{v}$ and, after the synchronous computation of the new state, each unit independently updates its respective component v_i. Hence, there is no possibility of collision during writing, since no two processors attempt to write simultaneously on the same shared memory location.

We have implemented the hybrid model on three different machines. For small problem instances (graph size 40–200) we used a Parsytec Multicluster 2 with 16 transputers. Due to memory limitations, larger instances could not be solved on the transputer system. For large problem instances (graph size 200–2000) we used a shared memory Silicon Graphics Power Challenge with 14 R8000 processors and a distributed memory Intel Paragon with 48 i860XP processors. Programming was performed using the C language with additional parallel calls for the coordination of the parallel processes. In all cases, the number of experiments, as well as the parameter settings and stopping criteria, were the same as in the corresponding sequential implementation. Also, the best case found during the sequential implementation ($\alpha = 0.25$) was considered in the parallel implementation.

	Avg. Speedup			
n	4 proc.	9 proc.	13 proc.	16 proc.
40	1.5	1.6	1.4	1.1
80	1.9	2.8	3.1	3.5
120	2.1	3.4	4.5	5.8
160	2.5	3.9	4.6	6.1
200	2.7	4.1	5.1	6.5

Table 3: Speedup for transputer implementation: small problem instances

	Avg. Speedup	
n	4 proc.	8 proc.
200	3.1	3.8
500	3.5	6.8
1000	3.7	7.1
1500	3.9	7.5
2000	3.9	7.9

Table 4: Speedup for shared memory implementation (Power Challenge): large problem instances

	Avg. Speedup		
n	5 proc.	11 proc.	21 proc.
200	3.4	6.5	6.1
500	3.8	9.0	15.5
1000	3.9	9.8	18.6
1500	3.9	9.9	19.7
2000	4.0	9.9	19.9

Table 5: Speedup for distributed memory implementation (Intel Paragon): large problem instances

In the case of shared memory machine the implementation was straightforward. The machine automaticaly partitions data, i.e. the units of the hybrid model, and assigns the partitions to each available processor. In the case of the two distributed memory machines apart of manually partitioning the data, we also had to simulate the shared memory operation. That is, at each repetition of the algorithm, we had to update the state vector $\vec{v}$ in each process. As a result we used an additional process to collect the state subvectors corresponding to each one of the p parallel processes, compose the vector $\vec{v}$, decide whether the terminating criterion has been reached and, if not, broadcast $\vec{v}$ to all processors. We should make clear at this point, that each parallel process manipulates n/p units, where p is the number of the available processors, and that each available processor runs only one such process.

The results of the three implementations are shown in Tables 3–5. We can easily observe that the parallel implementation of the proposed method fully exploits the available hardware, especially for large problem instances, where there is almost linear speedup. Note that speedup is computed in reference to the sequential execution time of the algorithm on the same machine. If we combine the results shown in Tables 2 and 4, referring to execution of the sequential and the parallel implementation of the proposed method on the same machine (shared memory), and compare them with the results shown in Table 1, it is interesting to see that the hybrid optimization method clearly outperforms the Boltzmann Machine in terms of execution time when implemented in parallel, for any instance of the problem. Even better execution times are expected if more processors are used.

6 Conclusions

The Boltzmann Machine Optimizer can provide near-optimal solutions of high quality to many hard optimization problems. The main disadvantage of this approach is that the network must operate sequentially (or with limited parallelism) in order to operate correctly and avoid oscillations. This sequential execution results in large solution times especially in the case of big problem instances.

The hybrid optimization scheme proposed in this paper has the same equilibrium states with the corresponding Boltzmann Machine and exhibits similar exploration performance with it. At the same time, it overcomes the sequential update bottleneck by incorporating the main features of a synchronous neural network model, such as the Cauchy Machine, thus allowing for an efficient parallel implementation. This combination results in an optimization tool that has the ability to perform effective exploration of the state space

while operating in a fully parallel and synchronous way.

References

[1] Aarts, E. and Korst, J., *Simulated Annealing and Boltzmann Machines, A Stochastic Approach to Combinatorial Optimization and Neural Computing*, J. Wiley & Sons, 1989.

[2] Ackley, D.H. Hinton, G. E. and Sejnowski, T. J., "A Learning Algorithm for Boltzmann Machines", Cognitive Science, vol. 9, pp. 147-165, 1985.

[3] Alspector, J. and Allen, R. B., "A Neuromorphic VLSI Learning System", in Advanced Research in VLSI, P. Losleben (ed.), pp. 313–346, MIT Press, 1987.

[4] Azencott, R., (ed.), *Simulated Annealing: Parallelization Techniques*, J. Wiley & Sons, Chichester, 1992.

[5] Barbosa, V.C. and Lima, P.M.V., "On the Distributed Parallel Simulation of Hopfield's Neural Networks", Software - Practice and Experience, vol. 20, no. 10, pp. 967–983, 1990.

[6] d'Acierno, A. and Vaccaro, R., "Hopfield's Binary Networks: Simulations on Tree-Connected Transputer Networks", Proc. ICANN'92, vol. 2, pp. 1409–1413, Brighton, UK, 1992.

[7] Funabiki, N., Takefuji, Y. and Lee, K. C., "Comparisons of Seven Neural Network Models on Traffic Control Problems in Multistage Interconnection Networks", IEEE Trans. on Computers, vol. 42, no. 4, pp. 497-501, 1993.

[8] Greening, D.R., "Parallel Simulated Annealing Techniques", Physica D, Vol. 42, pp. 293–306, 1990.

[9] Hopfield, J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", Proc. Nat. Acad. of Sciences USA, vol. 79, pp. 2554-2558, 1982.

[10] Hopfield, J.J and Tank, D.W., "Neural Computation of Decisions in Optimization Problems", Biological Cybernetics, vol. 52, pp. 141-152, 1985.

[11] Likas, A., and Stafylopatis, A., "Group Updates and Multiscaling: An Efficient Neural Network Approach to Combinatorial Optimization", IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics, vol. 26, pp. 222-232, 1996.

[12] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. and Teller, E., "Equation of State Calculations by Fast Computing Machines", Journal of Chemical Physics, vol. 21, no. 6, pp. 1087-1092, 1953.

[13] Roussel-Ragot, P. and Dreyfus, G., "A Problem Independent Parallel Implementation of Simulated Annealing: Models and Experiments", IEEE Trans. on Computer-Aided Design, vol. 9, No. 8, pp. 827–835, 1990.

[14] M. Skubiszewski, "A Hardware Emulator for Binary Neural Networks", Proc. Int. Neural Network Conf., Paris, July 1990, Vol. 2, 555–558.

[15] M. Skubiszewski, "An Exact Hardware Implementation of the Boltzmann Machine", Research Report, DEC Paris Research Laboratory, 1992.

[16] Szu, H. and Hartley, R., "Fast Simulated Annealing", Physics Letters A, vol. 122, pp. 157-162, 1987.

[17] Szu, H., "Fast TSP Algorithm Based on Binary Neuron Output and Analog Neuron Input Using the Zero-diagonal Interconnection Matrix and Necessary and Sufficient Constraints of the Permutation", International Joint Conf. on Neural Networks, vol. 2, pp. 259-266, San Diego, CA, 1988.

[18] Takefuji, Y. and Szu, H., "Design of Parallel Distributed Cauchy Machines", International Joint Conf. on Neural Networks, vol. 1, pp. 529-532, Washington D.C., 1989.

[19] Takefuji, Y., "Parallel Algorithms for Tiling Problems", IEEE Trans. on Neural Networks, vol. 1, no. 1, pp. 143-145, 1990.

[20] Takefuji, Y., "Parallel Algorithms for Finding a Near-Maximum Independent Set of a Circle Graph", IEEE Trans. on Neural Networks, vol. 1, no. 3, pp. 263-267, 1990.

[21] Ticknor, A. J. and Barret, H. H., "Optical implementations in Boltzmann Machines", Optical Engineering, vol. 26, pp. 16–21, 1987.

[22] Zissimopoulos, V., Paschos, V. and Pekergin, F., "On the Approximation of NP-complete Problems by Using the Boltzmann Machine Method. The Cases of Some

Covering and Packing Problems", IEEE Trans. on Computers, vol. 40, no. 12, pp. 1413-1419, 1991.