A Neural Networks Algorithm for the Minimum Coloring Problem Using FPGAs

Haidar Harmanani, Jean Hannouche, and Nancy Khoury
Departments of Computer Engineering and Science
Lebanese American University
P. O. Box 36
Byblos, 1401 2010
Lebanon

Abstract

This paper presents a hardware implementation to solve the graph coloring problem *(chromatic number $\chi(G)$) for arbitrary graphs using the hopfield neural network model of computation. The graph coloring problem, an NP-hard problem, has important applications in many areas including time tabling and scheduling, frequency assignment, and register allocation. The proposed algorithm has a time complexity of $O(1)$ for a neural network with n vertices and k colors. The algorithm was implemented using VHDL and downloaded on a Field Programmable Gate Array (FPGA) device. The algorithm was simulated and tested on various graphs, all yielding optimum solutions.*

KEY WORDS: Neural Networks, Combinatorial Optimization, graph coloring, FPGAs.

1 Introduction

Artificial Neural Networks (ANN) have been studied since 1943 when the first such model was suggested by McCulloch-Pitt [12]. Recent years have witnessed a considerable interest in analog VLSI neural networks to solve a variety of *hard* optimization problems [13]. The difficulty in such optimization problems is that the *best* solution is computationally very hard to find and the time they require to solve on any computer grows exponentially with the input size.

The *graph coloring problem* is an important problem and has wide applications and uses beyond map coloring. It can be used as an abstraction of time-tabling problems ([11, 19]). Garey, Johnson, and So [6] showed that the graph coloring problem can be used for short circuit testing for printed circuits. Chaitin [2] reduced register allocation to graph coloring. Funabiki and Takefuji [4] and Smith [14] showed that the graph coloring problem can be used to solve the frequency assignment problems. Takefuji and Lee [16] have used a discrete Hopfield-type network to solve the problem using four colors while Berger [1] has considered the general problem using k colors. Gassen and Carothers [7] extended these models to try to minimize the number of colors required for the coloring of a given graph.

The graph K-colorability problem has been proven to be solvable in polynomial time for $K = 2$ but remains NP-complete for all fixed $K \geq 3$ and, for $K = 3$, for planar graphs having no vertex degree exceeding 4. For an arbitrary K, the problem is NP-complete for circle graphs and circular arc graphs (even their representation as families of arcs), although for circular arc graphs the problem is solvable in polynomial time for any fixed K. The general problem can be solved in polynomial time for comparability graphs, for chordal graphs, for (3, 1) graphs, and for graphs having no vertex degree exceeding $three$ [5].

Several approximations and search algorithms were developed to solve the *graph coloring problem*. The most common approximation algorithm is that of successive augmentation. In this approach a partial coloring is found on a small number of vertices and this is extended vertex by vertex until the entire graph is colored [19, 11, 10].

This paper presents a parallel hardware method to solve the *graph coloring problem (chromatic number)*[1], an NP-complete problem, for arbitrary graphs based on a neural network model of computation, the Hopfield Neural Network (HNN). We use a large number of simple processing elements or *neurons*. We assume the McCulloch-Pitts binary neuron that performs the function of a simplified biological neuron [12].

The remainder of the paper is organized as follows. In section 2 we give some background on the HNN. In section 3 we formulate the graph coloring problem using the HNN and describe the neuron motion equations while section 4 describes our hardware implementation. Finally, experimental results are presented in section 5. We conclude with remarks in section 6.

2 Hopfield Neural Network: Background

In 1982 Hopfield introduced the collective computational property of an artificial neural network and later proposed in 1984 a continuous output model and showed the circuit

[1] The chromatic number is the smallest number of colors that can be used to color a graph

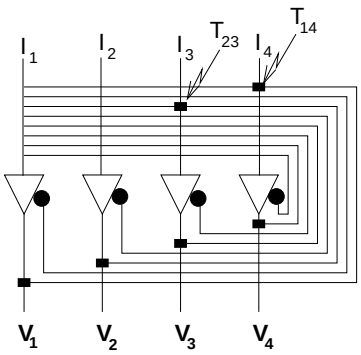

Figure 1. Simplified Hopfield Network with four neurons

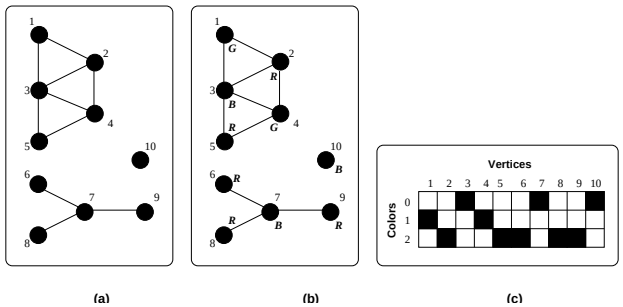

Figure 2. Neural Representation for the graph coloring problem: a) Initial graph, b) Solution (K=3), c) Neural representation

for implementing it. He used the *traveling salesman problem* (TSP), to illustrate his model [8, 9].

The Hopfield Neural Networks [8, 9] are single layer networks with output feedback consisting of simple processors or neurons that can collectively provide good solutions to difficult optimization problems. A simple Hopfield Network with four neurons is shown in Figure 1. A connection between two neurons is established through a conductance T_{ij} that transforms the voltage outputs of amplifier j to current input for amplifier i. Externally supplied bias current I_i is also present in every processor j. Each neuron i receives a weighted sum of the activation of other neurons, and updates its activation according to the rule:

$$V_i = g(U_i),$$

where $g(U_i)$ can be either a binary or a threshold function for the case of the McCulloch-Pitts neurons.

Hopfield showed that in the case of symmetric connection $(T_{ij} = T_{ji})$ the motion equation for the activation of the neurons of a HNN always leads to a convergence to a stable state, in which the output voltages of all the amplifiers remain constant. The stable states of a network of N neuron units are the local minima of the energy function:

$$E = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} T_{ij}V_iV_j - \sum_{i=1}^{N} I_iV_i, \qquad (1)$$

where V_i is the output of the i^{th} neuron and I_i is the externally supplied input or bias to the i^{th} neuron. E is referred to as the computational energy of the system. The equation of motion for the i^{th} neuron maybe described in terms of the energy function E as follows:

$$\frac{dU_i}{dt} = -\frac{U_i}{\tau} + \sum_{i\neq j} T_{ij}V_j + I_i, \qquad (2)$$

where $\tau = RC$ is the time constant of the RC circuit connected to neuron i.

Takefuji showed [15] that the above function performs the parallel gradient descent method. In fact, as long as the motion equation of the binary neurons is given by equation (2), the energy function E monotonically decreases. The state of the neural network is guaranteed to converge to the local minimum under the discrete numerical simulation.

3 Problem Formulation

The graph-coloring problem is to determine the minimum number of colors needed to color a given graph. The problem is described as follows. Let $G = (V, E)$ be an undirected graph where $V = \{v_1, v_2, ..., v_n\}$ is the set of vertices in G, and E $\subseteq$ $V \times V$ is the set of edges in G. Let K be a positive integer such as $K \leq |V|$. A k-coloring of G is a function $c : V \rightarrow \{1, 2, ..., K\}$ such that $c(u) \neq c(v)$ whenever $\{u, v\} \in E$. In other words, the numbers $1, 2, ..., k$ represent k colors, and adjacent vertices must have different colors. The graph coloring problem is to determine the minimum number of colors needed to color a given graph.

In order to solve the *graph coloring problem*, it is first necessary to formulate the problem using the hopfield model. Let $G = (V, E)$ be a graph, and let $n = |V|$ and k be the number of colors. The adjacency matrix of G is denoted $Adj = (a_{ij})$ where $a_{ij} = 1$ if (v_i, v_j) is an edge in G, i.e., $(v_i, v_j) \in E$ and $a_{ij} = 0$ if $(v_i, v_j) \notin E$. Formally, the graph coloring problem is defined as follows [5]:

> **Instance**: Graph *G = (V, E)* and a positive integer $K \leq |V|$
>
> **Question**: Is G K-colorable, i.e., does there exist a function $f : V \rightarrow \{1, 2, ..., K\}$ such that $f(u) \neq f(v)$ whenever $\{u, v\} \in E$?

The above definition leads to the following two optimization criteria:

1. One and only one color is to be assigned to each vertex.
2. No two adjacent vertices should have the same color.

The first criterion can be expressed by the following cost function

$$\sum_{j=1}^{k} V_{xj} - 1 \quad (3)$$

This will encourage one and only one neuron to fire in each column of our neural network representation. The second criterion can be expressed as the following cost function:

$$\sum_{y=1,y\neq x}^{n} Adj_{yx} V_{yi} \sum_{l=1}^{n} Adj_{yl} \quad (4)$$

where *Adj* is the adjacency matrix representation for graph G. The second requirement ensures that neurons fired in the same row are not mutually adjacent.

The above two terms in equations (3) and (4) ensure that the network converges to a local minimum. In order to ensure an optimal solution, the system must be able to escape local minima and to converge to global minima. This is fulfilled using an additional *hill-climbing* term in the motion equation. The hill-climbing term is activated by reseting the highest *color* to a *zero*. It is excitatory and will encourage the vertex with the highest color to fire a new color thus causing the whole network to re-evaluate. The hill-climbing term is:

$$[V_{xh} \neq \sum_{l=1}^{n} \sum_{m}^{k} Max(V_{ml})] \quad (5)$$

where the bracketed notation $[S]$ is 1 if S is true and 0 otherwise.

Based on equations (3), (4) and (5), the motion equation for the i^{th} neuron can be described by:

$$\begin{aligned} \frac{dU_{xi}}{dt} &= -A(\sum_{j=1}^{k} V_{xj} - 1) \\ &\quad -B(\sum_{y=1,y\neq x}^{n} Adj_{yx} V_{yi} \sum_{l=1}^{n} Adj_{yl}) \\ &\quad +C[V_{xh} \neq \sum_{l=1}^{n} \sum_{m}^{k} Max(V_{ml})], \end{aligned} \quad (6)$$

where A, B, and C are positive connection weights and the bracketed notation $[S]$ is 1 if S is true and 0 otherwise.

4 Hardware Implementation

The main issue in the hardware solution for the graph coloring problem is the design of a neuron that will represent a node in an arbitrary graph. A neuron communicates with other neurons in the network in order to form a solution. Thus, adjacent neurons check each others color and generate a new color that is not equal to their neighbors. This process of checking will stop when all adjacent neurons have different colors or when the neurons output remain stable implying that a solution was found.

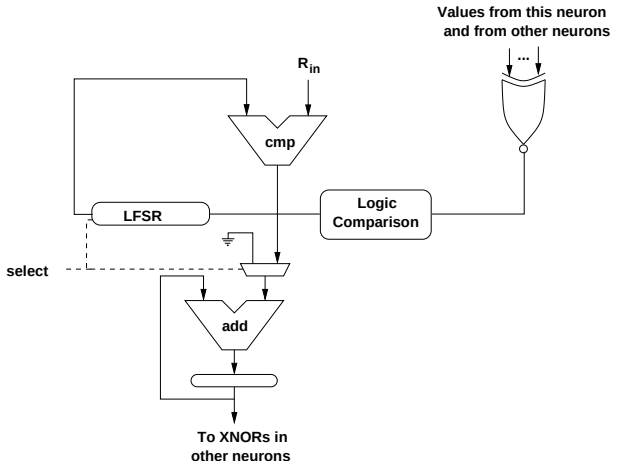

Figure 3. Neuron Design

The neuron was implemented using VHDL[2]. Since we support up till eight colors, we represent a color using a 3-bit word. The neuron design, shown in Figure 3, consists of several components including a color comparator, a linear feedback shift register (LFSR), an adder, a general purpose comparator, a register and a multiplexer. Neurons are then interconnected in order to create a Hopfield Neural Network based on the graph adjacency matrix and the defined energy function. In what follows, we describe these components in detail.

- **Colors Comparator:** The colors comparator compare the colors of adjacent neurons. If adjacent neurons have the same color, a control signal is sent to the LFSR. Since we support eight colors only, this component was implemented using six groups of gates.

- **Linear Feedback Shift Register (LFSR):** The LFSR is responsible for the generation of random 8-bit numbers and is based on the polynomial: $P(x) = x^8 + x^6 + x^5 + x^3 + x + 1$. The polynomial allows the LFSR to generate 90 pseudo-random numbers before the sequence restarts all over again. Each neuron has an LFSR with a different seed value. The LFSR generates random number when it receives a control signal from the colors comparator. The random value is sent next to the main comparator.

- **Main Comparator:** The *main comparator* compares the random number received from the LFSR with the number that input by the user. If the input number were greater than the random number, the comparator's output would be a logial $zero$ and a logical one otherwise. This result is then passed to the adder.

[2]Short for Very High Speed Integrated Circuits Hardware Description Language

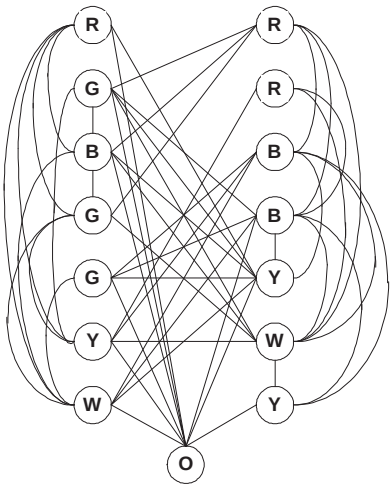

Figure 4. Random graph 1 generated using the DIMACS graph coloring generator

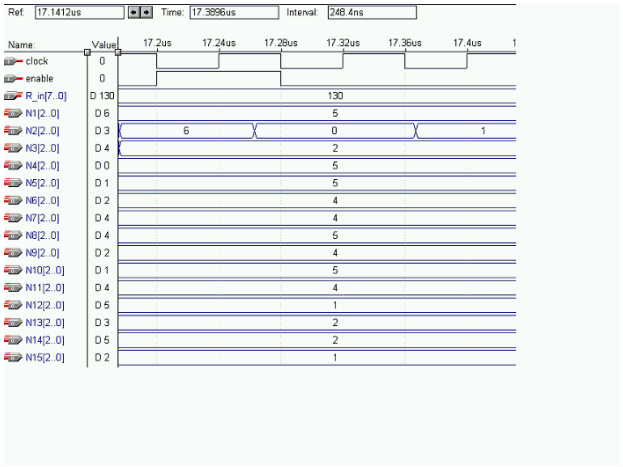

Figure 5. Random graph 1 simulation

- **Multiplexer:** The multiplexer chooses one of two data bus, the result of the comparator and a logcal $zero$. The select line of the multiplexer is connected to the enable of the LFSR such that when the select is '1', the result of the comparator is passed to the adder. Otherwise, that is when the solution is stable, the second input is passed to the adder.
- **Adder:** The adder is responsible for changing the color of the neuron. This is done by incrementing the previous color value stored in the register.
- **The Register:** The 3-bit register stores the color value of the neuron. The value in the register is updated whenever the output of the adder is changed.

4.1 Hardware Hill-Climbing Term

In order to escape from local minima, a hill-climbing component was implemented. The controller consists of two parts: *collect* and *choose*. The first part is responsible for collecting the color index of each neuron in the graph. The collected values are then compared and the neuron with the greater color index is found. The result is then passed to the second part of the controller which resets the color of the corresponding neuron. Thus, the value stored in the register of the neuron will be forced to change its color. If a color violation results from this process then the neural network will try to find a new solution. Otherwise, if no violation occurred, then the number of colors is reduced one.

5 Experimental Results

The proposed parallel algorithm was implemented for verification purposes using VHDL and downloaded on an Altera UP1 Board. We used the Altera EPF10K20 device which is based on a static SRAM elements. The device has 1,152 logic elements and six embedded array blocks (EABs). Each EAB provides 2,048 bits of memory and can be used to implement logic functions. The device has the equivalent of 20,000 logical gates. The UP1 board was connected to a VGA monitor in order to display the original graph as well as the resulting colored graph.

In order to verify our design, two sets of examples were attempted. The number of vertices in the graphs varied from 7 to 15, the largest number that we could handle due to the limited number of gates in the EPF1K20 device.

The first set is from the high-level synthesis benchmark suite. In specific, *the Tseng Datapath Circuit* [3], *the Differential Equation example* [3] that solves a second order differential equation, and *the register optimization example* [18] where registers that are used in the same clock cycle are colored using two different colors.

The second set of examples were randomly generated using the graph generator for the coloring problem from the *Center for Discrete Mathematics & Theoretical Computer Science* (DIMACS). The generator was ran with a 50% edge probability. Four graphs were generated in this set. Figure 4 illustrates one of the random DIMACS graphs while figure 5 shows the simulation results for this graph. The graph was colored in 17.37 μs

Table 1 illustrates the specification of these examples as well as the results. At each run, the network was randomly initialized by the LFSR, using a random seed for each neuron. The network was able to color all attempted graphs with the optimum number of color. This is consistent with the Hopfield Neural Networks that deterministically converge to a "minimum energy"after a series of iterations. Note that R_{in} is an initial value entered by the user. The table also shows the number of colors that was used to color the graph while $\chi(G)$ shows the minimum number of col rs needed to color the graph. Note that all graphs were colored with the least number of colors in a very small time.

Example	Graphs Characteristics				# Iterations	Time
	# Vertices	# Edges	# Colors	$\chi(G)$		(μs)
Simple Graph	6	9	3	3	2	1.49
Tseng Data Path	10	10	3	3	43	31.52
Register Allocation	11	23	4	4	2	1.03
Differential Equation	10	20	3	3	4	1.75
DIMACS Random Graph 1	15	48	7	7	8	17.37
DIMACS Random Graph 2	15	51	7	7	2	4.55
DIMACS Random Graph 3	15	50	7	7	5	8.59
DIMACS Random Graph 4	15	47	7	7	6	13.97

Table 1. Attempted graph coloring results

6 Conclusion

We have presented a *hardware based parallel algorithm* to solve the *graph coloring problem* based on the Hopfield Neural Network model of computation. Our method has shown promising results in solving a hard combinatorial optimization problem in a reasonably fast time. The interesting part about our parallel algorithm versus "classical" algorithms is that in our case, convergence time did not depend on the problem size as it is clear in Table 1. Hence, the $O(1)$ execution time for the parallel algorithm is apparent.

References

[1] M. O. Berger, "K-Coloring Vertices using a Neural Network with Convergence to Valid Solutions,"*Proc. Inter. Conf. on Neural Networks*, 4514-4517, 1994.

[2] G. Chaitin, "Register Allocation and Splling via Graph Coloring," *Proc. of the ACM SIGPLAN 82 Symposium on Compiler Construction*, 98-105, 1982.

[3] G. De Michelli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill Inc., 1993.

[4] N. Funabiki, Y. Takefuji, "A Neural Network Parallel Algorithm for Channel Assignment Problems in Cellular Radio Networks, *IEEE Trans. on Vehicular Technology*, Vol. 41, 430-437, 1992.

[5] M. Garey, D. Johnson, *Computers and Intractability – A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, 1979.

[6] M. Garey, D. Johnson, H. So, "An Application of graph coloring to printed circuit testing," *IEEE Trans. on Circuits and Systems*, Vol. 23, pp. 591-599, 1976.

[7] D.W. Gassen, J.D. Carothers, "Graph Color Minimization Using Neural Networks,"*Proc. Inter. joint Conf. on Neural Networks*, 1541-1544, 1993.

[8] J. Hopfield, "Neurons with Graded Response Have Collective Computational Properties Like Those of Two State Neurons," *In Proc. Nat. Acad. Science*, Vol. 79, pp. 2554-2558, 1982.

[9] J. Hopfield, D. W. Tank, "Neural Computation of Decision in Optimization Problems," *Biological Cybernetics*, Vol. 52, pp. 141-152, 1985.

[10] D. Johnson, "Approximation Algorithms for Combinatorial Problems," *J. of Computer System Sciences*, Vol. 9, pp. 256-278, 1974.

[11] F. Leighton, "A Graph Coloring Algorithm for Large Scheduling Problems," *J. of Research of the National Bureau of Standards*, Vol. 84, pp. 489-506, 1979.

[12] W. S. McCulloch, W. H. Pitts, "A Logical Calculus of Ideas Imminent in Nervous Activity," *Bullet. Math. Biophys.*, Vol. 5, pp. 115-133, 1943.

[13] K. Smith, "Neural Networks for Combinatorial Optimization: A Review of More than a Decade of Research,"*INFORMS J. on Comp.*, Vol. 11, Nb. 1, 1999.

[14] K. Smith, M. Palaniswami, "Static and Dynamic Channel Assignmnent Using Neural Networks," *IEEE Journal on Selected Areas in Communications*, Vol. 15, pp. 238-249, 1997.

[15] Y. Takefuji, K Lee, "A Near Optimum Parallel Planarization Algorithm," *Science*, Vol. 245, pp. 1221-1223, 1989.

[16] Y. Takefuji, K.C. Lee, "Artificial Neural Networks for Four-Coloring Map Problems and K-Colorability Problems," *IEEE Trans. on Circuits and Systems*, Vol. 38, Nb. 3, pp. 325-333, 1991.

[17] Y. Takefuji, *Neural Network Parallel Computing*, Boston, Kluwer, 1992.

[18] D. Thomas, E. Lagnese, R. Walker, J, Nestor, R. Rajan, R. Blackburn, *Algorithmic and RT Synthesis: The System Architect's Workbench*, Kluwer, 1990.

[19] D. Welsh, M. Powell, "An Upper Bound on the Chromatic Number of a Graph and its Appl. to Timetabling Problems," *Computer*, Vol. 10, pp. 85-86, 1967.