International Journal of Modelling and Simulation

ISSN: 0228-6203 (Print) 1925-7082 (Online) Journal homepage: http://www.tandfonline.com/loi/tjms20

# A Neural Networks Algorithm for the Minimum Colouring Problem Using FPGAs†

Haidar Harmanani, Jean Hannouche & Nancy Khoury

**To cite this article:** Haidar Harmanani, Jean Hannouche & Nancy Khoury (2010) A Neural Networks Algorithm for the Minimum Colouring Problem Using FPGAs†, International Journal of Modelling and Simulation, 30:4, 506-513

**To link to this article:** http://dx.doi.org/10.1080/02286203.2010.11442597

# A NEURAL NETWORKS ALGORITHM FOR THE MINIMUM COLOURING PROBLEM USING FPGAs†

Haidar Harmanani,* Jean Hannouche,* and Nancy Khoury*

**Abstract**

This paper presents a hardware implementation to solve the *graph colouring problem* (chromatic number $\chi(G)$) for arbitrary graphs using the Hopfield neural network (HNN) model of computation. The graph colouring problem, an $\mathcal{NP}$-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 $k$ colours. The algorithm was implemented using VHSIC hardware description language (VHDL) and downloaded on a field programmable gate array (FPGA) device. The resulting hardware was simulated and tested on various graphs, all yielding optimum solutions.

**Key Words**

Neural networks, combinatorial optimization, graph colouring, FPGAs

## 1. Introduction

Artificial neural networks (ANNs) have been studied since 1943 when the first such model was suggested by McCulloch-Pitt [1]. Recent years have witnessed a considerable interest in analog Very-large-scale integration (VLSI) neural networks to solve a variety of *hard* optimization problems [2]. 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 colouring problem is to determine the minimum number of colours needed to colour a given graph. Formally, the problem is defined as follows [3]:

**Instance**: Graph $G=(V,E)$ and a positive integer $K \leq |V|$

**Question**: Is $G$ $K$-colourable, i.e., does there exist a function $f : V \rightarrow \{1, 2, \ldots, K\}$ such that $f(u) \neq f(v)$ whenever $\{u, v\} \in E$?

The graph $K$-colourability problem has been proven to be solvable in polynomial time for $K=2$ but remains $\mathcal{NP}$-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 $\mathcal{NP}$-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 3 [3].

The *graph colouring problem* is an important problem and has wide applications and uses beyond map colouring. For example, Leighton [4] and Welsh *et al.* [5] showed that the graph colouring problem can be used as an abstraction of time-tabling problems. Garey *et al.* [6] showed that the graph colouring problem can be used to detect whether there is a short circuit on a printed circuit board. Chaitin [7] reduced the register allocation problem to graph colouring. Funabiki *et al.* [8] and Smith [9] showed that graph colouring can be used to solve the frequency assignment problems.

Several software-based approximation algorithms were developed to solve the *graph colouring problem*. The most common approximation algorithm is that of successive augmentation. In this approach, a partial colouring is found on a small number of vertices, and this is extended vertex by vertex until the entire graph is coloured [4, 5, 10]. Johnson *et al.* [11] formulated the graph colouring problem using three simulated annealing techniques, and an augmentation algorithm. Davis *et al.* [12] proposed a genetic algorithm solution; however, the algorithm gave poor results. Kirovski *et al.* [13] proposed a graph colouring algorithm that uses two distinct algorithms and a hybrid. The first is based on successive augmentation while the second is based on a lottery-scheduling-driven iterative improvement. The hybrid algorithm uses initially the augmentation-based algorithm to reduce the search space before proceeding into the lottery-based algorithm. Takefuji *et al.* [14] used a discrete Hopfield-type network

* Department of Computer Science and Mathematics, Lebanese American University, Byblos 1401 2010, Lebanon; e-mail: haidar@lau.edu.lb, {JeanHannouche5, Nancykhoury}@yahoo.com

†This work has appeared in abridged form in Proceedings of the IASTED International Conference on Applied Simulation and Modeling (ASM 2003), Marbella, Spain, 152–156, September 2003.

*Recommended by Prof. A. Houshyar*
*(10.2316/Journal.205.2010.4.205-4377)*

to solve the problem using four colours while Berger [15] considered the general problem using $k$ colours. Gassen and Carothers [16] extended these models to minimize the number of colours required for the colouring of a given graph. Harmanani [a] formulated the data path allocation problem in high-level synthesis using a Hopfield neural network computational model. The data path allocation problem was formulated using clique-partitioning, and solved using a software-based parallel simulation model.

To improve neural computation processing rates, several researchers explored implementing neural networks using field programmable gate arrays (FPGAs). Omondi *et al.* [17] explored various techniques in FPGA implementations for neural networks using *independent component neural networks*. Lim *et al.* [18] presented two FPGA hardware implementations for two independent component neural networks: one that maximizes mutual information between input and output signals and one that minimizes the divergence at the output. Li *et al.* [19] presented a reconfigurable approach for neural hardware. The approach explores possible issues in FPGA implementation of neural networks. Maeda *et al.* [20] presented an FPGA implementation for a Hopfield neural network (HNN) system with learning capability using the simultaneous perturbation learning rule. Watanabe *et al.* [21] proposed a hardware implementation to solve the minimum $p$-quasi clique cover problem and its implementation on an FPGA. Abramson *et al.* [22] presented an implementation of the HNNs for solving constraint satisfaction problems using FPGAs. However, the approach used unrealistically small and simplistic examples to solve the *eight* queens problem. Varma *et al.* [23] presented a neural network solution for the travelling salesperson's problem, and proposed a mapping to an FPGA.

This paper presents a hardware method to solve the *graph colouring problem* (*chromatic number* $\chi(G)$)[1], an $\mathcal{NP}$-complete problem, for arbitrary graphs based on the HNN model of computation. 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 [1].

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 colouring problem using the HNN, and describe the neurons motion equation. Section 4 presents the parallel neural networks algorithm while Section 5 describes the hardware implementation using an FPGA in addition to the hill-climbing mechanism. We present experimental results in Section 6, and conclude with remarks in Section 7.

## 2. Hopfield Neural Network: Background

In 1982, Hopfield introduced the collective computational property of an ANN and later proposed in 1984 a continuous output model, and showed the circuit for implementing it. He used the *travelling salesman problem* (TSP) to illustrate his model [24, 25].

[1] The chromatic number is the smallest number of colours that can be used to colour a graph.

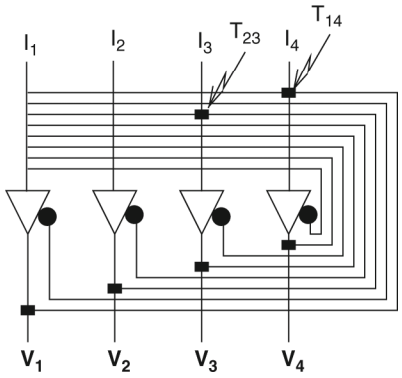



Figure 1. Simplified Hopfield network with four neurons.

The HNNs [24, 25] 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 Fig. 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:

$$V_i = g(U_i) \tag{1}$$

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 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:

$$E = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} T_{ij}V_iV_j - \sum_{i=1}^{N} I_iV_i \tag{2}$$

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:

$$\frac{dU_i}{dt} = -\frac{U_i}{\tau} + \sum_{i\neq j} T_{ij}V_j + I_i \tag{3}$$

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

Takefuji [26] showed 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 (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 colouring problem is described as follows. Let $G = (V, E)$ be an undirected graph where $V = \{v_1,$

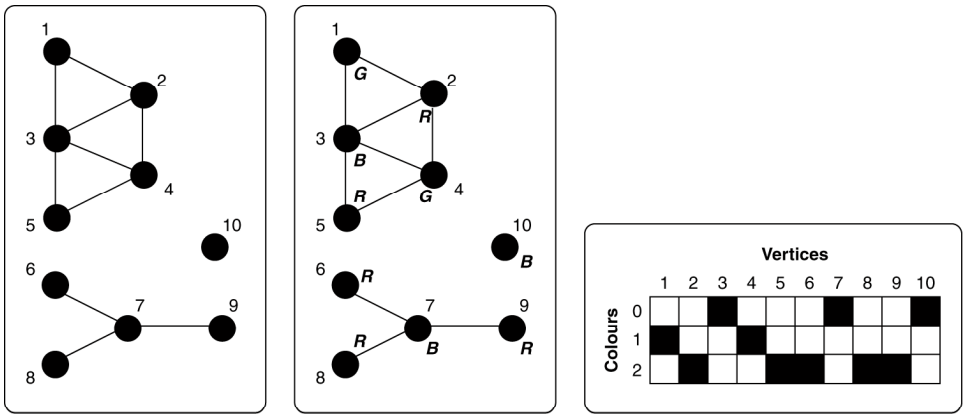



Figure 2. Neural representation for the graph colouring problem: (a) initial graph, (b) solution ($K=3$), and (c) neural representation.

$v_2, \ldots, v_n\}$ 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 $k$-colouring of $G$ is a function $c : V \rightarrow \{1, 2, \ldots, K\}$ such that $c(u) \neq c(v)$ whenever $\{u, v\} \in E$. In other words, the numbers $1, 2, \ldots, k$ represent $k$ colours, and adjacent vertices must have different colours. The graph colouring problem is to determine the minimum number of colours needed to colour a given graph.

To solve the *graph colouring problem*, it is first necessary to formulate the problem using the Hopfield model. Let $n = |V|$ and $k$ be the number of colours. The adjacency matrix of $G$ is denoted as $Adj = (a_{ij})$ 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 $(v_i, v_j) \notin E$. We formulate the *graph colouring problem* using a neural representation that uses a two-dimensional array composed of $n \times k$ neurons (Fig. 2). Every array cell corresponds to a neuron connected to every other neuron using a conductance. A neuron in the array fires if it is to be a part of a specific shared resource. A row in the network corresponds to set of vertices that have the same colour, while a column corresponds to a vertex. The firing rules in the network are determined using an energy function that will be described next. For example, the graph in Fig. 2(a) can be coloured using three colours as shown in Fig. 2(c), an optimal colouring in this case. Thus, vertices 3, 7, and 10 are coloured using the same colour.

### 3.1 Motion Equation

To enable the neurons to compute a solution to our problem, we must describe the network using an *energy function* that describes the motion of the $i$th neuron in the network. This will be done using the two kinds of forces that exist in neural networks, *excitatory* and *inhibitory*. For example, if vertices $v_i$ and $v_j$ are to be assigned a colour $k$, then neurons $(i, k)$ and $(j, k)$ are encouraged to fire as the excitatory force. However, if vertices $v_i$ and $v_j$ are connected by an edge, then either neuron is discouraged from firing as the inhibitory force.

The graph colouring problem definition leads to the following two optimization criteria:

1. There should be only one colour assigned to each vertex.
2. Adjacent vertices cannot be assigned the same colour.

The first criterion can be expressed by the following term in the motion equation:

$$\sum_{j=1}^{k} V_{xj} - 1 \tag{4}$$

This will encourage one and only one neuron to fire in each column of our neural network representation.

The second criterion can be fulfilled by ensuring that neurons that fire in the same row are not mutually adjacent. This can be expressed with the following term in the motion equation:

$$\sum_{y=1, y \neq x}^{n} Adj_{yx} V_{yi} \sum_{l=1}^{n} Adj_{yl} \tag{5}$$

where $Adj$ is the adjacency matrix representation for graph $G$.

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

$$V_{xi} = 0 \text{ if } U_{xi} \geq \text{Max}(U_{ml}), \ \ \forall m, l \tag{6}$$

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

$$\frac{dU_{xi}}{dt} = -A \left( \sum_{j=1}^{k} V_{xj} - 1 \right) - B \left( \sum_{y=1, y \neq x}^{n} Adj_{yx} V_{yi} \sum_{l=1}^{n} Adj_{yl} \right) \tag{7}$$

where $A$ and $B$ are positive connection weights.

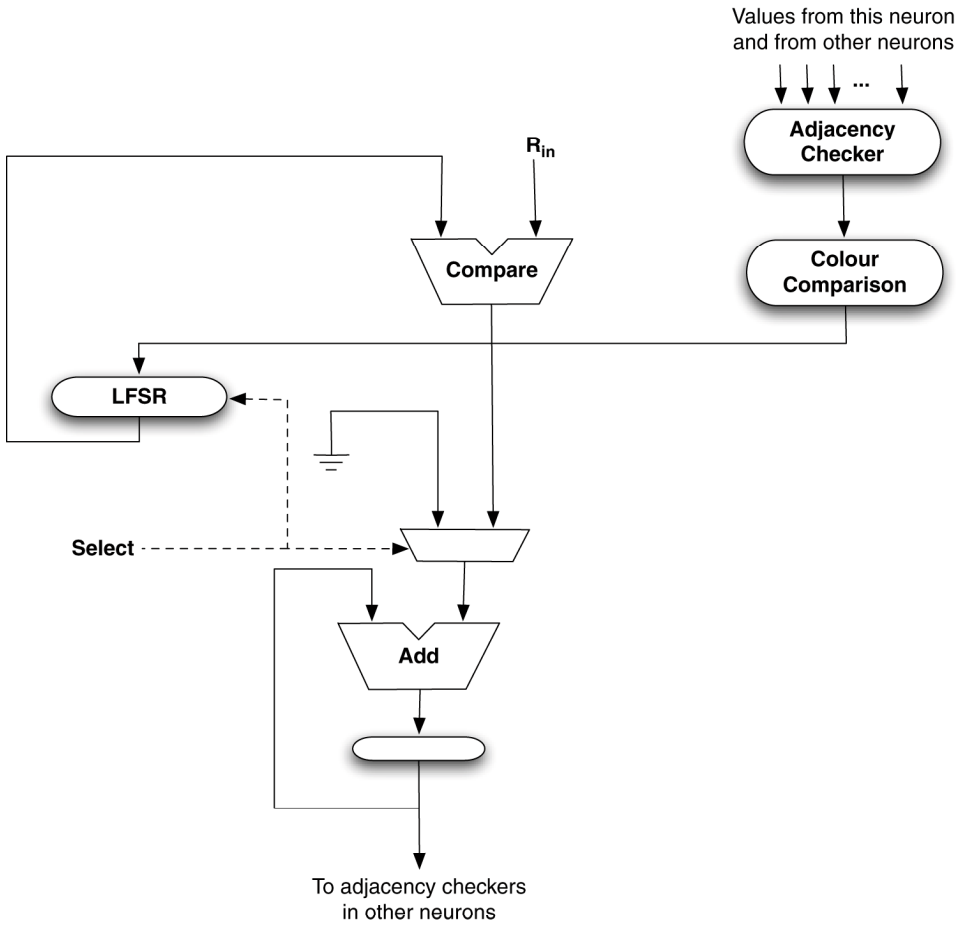



Figure 3. Neuron design.

## 4. The Colouring Algorithm

The proposed algorithm can be implemented in parallel as illustrated below, based on the first-order Euler method.

*Step 1*: Initialize.
Read the graph to be coloured.
*Step 2*: Initialize the neural network
Set $t=0$; $\Delta t=1$; $A=B=1$
Randomize the initial values of $U_{xi}(t)$ in the range $(-\omega, 0)$ where $x=1,2,\ldots,n$ and $i=1,2,\ldots,n$ and $\omega=120$
*Step 3*: Evaluate $V_{xi}(t)$ based on the binary function:

$$V(X_i(t)) = f(U_{xi}(t)) = \begin{cases} 0 & \text{if } U_{xi} \leq 0 \\ 1 & \text{otherwise} \end{cases}$$

*Step 4*: Use the motion equation in (6) to compute $dU_{xi}(t)$.
*Step 5*: Compute $U_{xi}(t+1)$ using the first-order Euler method
$U_{xi}(t+\Delta t) = U_{xi}(t) + \Delta U_{xi}(t)\Delta(t)$ where $x=1,2,\ldots,n$ and $i=1,2,\ldots,n$ and $\Delta U_{xi} = \frac{dU_{xi}}{dt}$
*Step 6*: Update.
Increment $t$ by 1. If the system is in equilibrium or $t=T$ then terminate. Else go to step 3.

## 5. Hardware Implementation

The main issue in the hardware solution for the graph colouring 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 colour the graph. Adjacent neurons check the colours of neighbouring neurons, and generate a different colour. The process of checking will stop when all adjacent neurons have different colours, or when the neurons outputs remain stable implying that a solution was found.

Neurons were implemented using VHDL.[2] As we support up till seven colours, we represent a colour using a 3-bit word. The neuron design, shown in Fig. 3, consists of several components including an adjacency checker, a colour comparator, a linear feedback shift register (LFSR), an adder, a general purpose comparator, a register and a multiplexer. Neurons are then interconnected to create a HNN based on the graph adjacency matrix and the defined energy function. In what follows, we describe these components and their function in detail.

- **Adjacency Checker**: The *adjacency checker* determines the nodes that are adjacent in the graph. The checker consists of groups of XNOR gates. Each group

[2] Short for very high speed integrated circuits hardware description language.

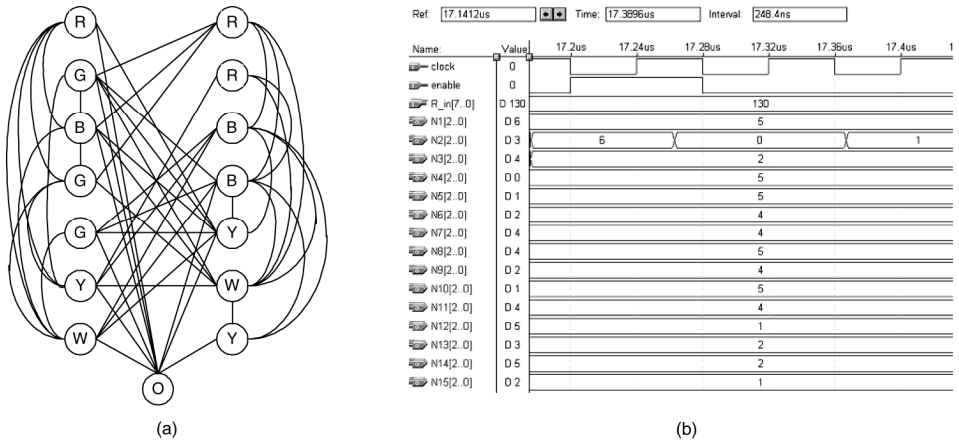



Figure 4. Random graph 1 generated using the DIMACS software. (a) graph and (b) hardware simulation.

consists of *three* gates corresponding to the number of bits used to determine the colour.

- **Colours Comparator**: The colours comparator compares the colours of adjacent neurons. If one or more adjacent neurons have the same colour as the current neuron, a control signal is sent to the LFSR.
- **(LFSR)**: The LFSR is responsible for the generation of random 8-bit numbers and is based on the maximal 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 colours 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 is that input by the user. If the input number were greater than the random number, the comparator's output would be a logical *zero* and a logical *one* otherwise. This result is then passed to the adder.
- **Multiplexer**: The multiplexer chooses one of two data bus, the result of the comparator and a logical *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. When the solution is stable, the second input is passed to the adder.
- **Adder**: The adder is responsible for changing the colour of the neuron. This is done by incrementing the previous colour value stored in the register.
- **Register**: The 3-bit register stores the colour value of the neuron. The value in the register is updated whenever the output of the adder is changed.

### 5.1 Hardware Hill-Climbing Term

Each neuron "repels" its neighbours to colour the graph. The collective behaviour of the system attempts to minimize the motion equation (6). However, the dynamics of the network is unstable, and it oscillates between minima. To simplify determining the solution, we note that the oscillatory behaviour of the neuron states is about a local minimum of $E$. A moving average circuit is used to obtain the state corresponding to the "mini" of $E$. The moving average of a varying input is obtained by simply feeding it into shift registers to store its previous values and then adding up these values.

To escape from local minima, a hill-climbing component was implemented based on a controller that consists of two parts: *collect* and *choose*. The *collect* part is responsible for collecting the colour index of each neuron in the graph. The collected values are then compared and the neuron with the greater colour index is found. The result is then passed to the second part of the controller that resets the colour of the corresponding neuron. Thus, the value stored in the register of the neuron will be reset and forced to change its colour. If a colour 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 colours is reduced one. The controller accepts an input that determines how many times it can activate and reset the value of a certain neuron. After testing the hill-climbing controller on all graphs, all optimal solutions were found. The controller allows local changes in the graph, thus moving towards the optimal solution.

## 6. 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 to display the original graph as well as the resulting coloured graph.

To verify our design, two sets of examples were attempted. The number of vertices in the graphs varied from

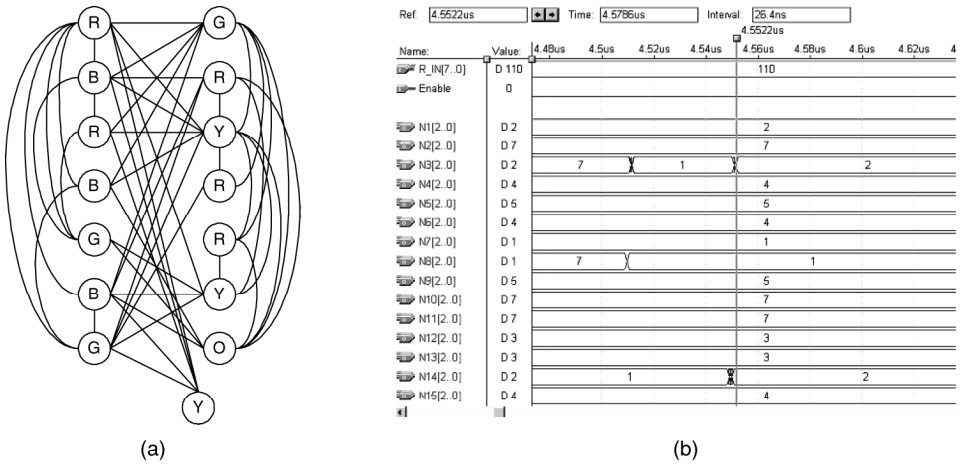



Figure 5. Random graph 2 generated using the DIMACS software. (a) graph and (b) hardware simulation.

Table 1
Graph Colouring Results

| Example | Graphs Characteristics | | | | # Iterations | # Neurons | Time ($\mu$s) |
|---|---|---|---|---|---|---|---|
| | # Vertices | # Edges | # Colours | $\chi(G)$ | | | |
| Simple graph | 6 | 9 | 3 | 3 | 2 | 18 | 1.49 |
| Tseng data path | 10 | 10 | 3 | 3 | 43 | 30 | 31.52 |
| Register allocation | 11 | 23 | 4 | 4 | 2 | 44 | 1.03 |
| Differential equation | 10 | 20 | 3 | 3 | 4 | 30 | 1.75 |
| DIMACS random graph 1 | 15 | 48 | 4 | 4 | 8 | 60 | 17.37 |
| DIMACS random graph 2 | 15 | 51 | 6 | 6 | 2 | 90 | 4.55 |
| DIMACS random graph 3 | 15 | 50 | 5 | 5 | 5 | 75 | 8.59 |
| DIMACS random graph 4 | 15 | 47 | 5 | 5 | 6 | 75 | 13.97 |

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* [27], *the Differential Equation example* [27] that solves a second-order differential equation, and *the register optimization example* [28] where registers that are used in the same clock cycle are coloured using two different colours.

The second set of examples were randomly generated using the graph generator for the colouring 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. Figures 4(a) and 5(a) illustrate two of the random DIMACS graphs while Figures 4(b) and 5(b) show the simulation results for the same graphs, respectively. The graphs were coloured in 17.37 and 4.55 $\mu$s, respectively.

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 colour all attempted graphs with the optimum number of colour. This is consistent with the HNNs 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 colours that was used to colour the graph while $\chi(G)$ shows the minimum number of colours needed to colour the graph. Note that all graphs were coloured with the least number of colours in a very small time. It should be noted that although the graph colouring problem is $\mathcal{NP}$-complete, the running time did not increase exponentially as the size of the input increased.

## 7. Conclusion

We have presented a *hardware-based parallel algorithm* to solve the *graph colouring problem* based on the HNN model of computation. The method expands the work in [b] by improving the hill-climbing terms while presenting more substantial results. The 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.

## References


[1] W.S. McCulloch & W.H. Pitts, A logical calculus of ideas imminent in nervous activity, *Bulletin Mathematical Biophysics*, *5*, 1943, 115–133.

[2] K. Smith, Neural networks for combinatorial optimization: A review of more than a decade of research, *INFORMS Journal on Computing*, *11*(1), 1999, 15–34.

[3] M. Garey & D. Johnson, *Computers and intractability – A guide to the theory of NP-completeness* (New York, NY, USA: W. H. Freeman and Company, 1979).

[4] F. Leighton, A graph colouring algorithm for large scheduling problems, *Journal of Research of the National Bureau of Standards*, *84*, 1979, 489–506.

[5] D. Welsh & M. Powell, An upper bound on the chromatic number of a graph and its application to timetabling problems, *Computer*, *10*, 1967, 85–86.

[6] M. Garey, D. Johnson, & H. So, An application of graph colouring to printed circuit testing," *IEEE Transactions on Circuits and Systems 23*, 1976, 591–599.

[7] G. Chaitin, Register allocation and Spilling via graph colouring, *Proc. of the ACM SIGPLAN 82 Symposium on Compiler Construction*, Boston, MA, 1982, 98–105.

[8] N. Funabiki & Y. Takefuji, A neural network parallel algorithm for channel assignment problems in cellular radio networks, *IEEE Transactions on Vehicular Technology*, *41*, 1992, 430–437.

[9] K. Smith & M. Palaniswami, Static and dynamic channel assignment using neural networks, *IEEE Journal on Selected Areas in Communications*, *15*, 1997, 238–249.

[10] D. Johnson, Approximation algorithms for combinatorial problems, *Journal of Computer System Sciences*, *9*, 1974, 256–278.

[11] D. Johnson, C.R. Aragon, L.A. McGeoch, & C. Schevon, Optimization by simulated annealing: An experimental evaluation; Part II, Graph Colourinig and number partitioning, *Operations Research*, *39*(3), 1991, 378–406.

[12] L. Davis, *Handbook of genetic algorithms* (New York: Van Nostrand Reinhold, 1991).

[13] M. Potkonjak & D. Kirovski, Efficient colouring of a large spectrum of graphs, *Proc. of the 35th ACM/IEEE Design Automation Conference*, San Francico, CA, 1998, 427–431.

[14] Y. Takefuji & K.C. Lee, Artificial neural networks for four-colouring map problems and K-colorability problems, *IEEE Transactions on Circuits and Systems*, *38*(3), 1991, 325–333.

[15] M.O. Berger, K-Colouring vertices using a neural network with convergence to valid solutions, *Proc. International Conf. on Neural Networks*, Nagoya, Japan, 1994, 4514–4517.

[16] D.W. Gassen & J.D. Carothers, Graph colour minimization using neural networks, *Proc. International Joint Conf on Neural Networks*, Nagoya, Japan, 1993, 1541–1544.

[17] A. Omondi & J. Rajapakse, Neural networks in FPGAs, *Proc. of the 9th International Conference on Neural Information Processing (ICONIP'OZ)*, Orchid Country Club, Singapore, 2002.

[18] A.B. Lim, J.C. Rajapakse, & A.R. Omondi, Comparative study of implementing ICNNs on FPGAs, *Proc. International Joint Conference on Neural Networks*, Washington, DC, USA, 2001, 177–182.

[19] A. Li, Q. Wang, Z. Li, & Y. Wan, A reconfigurable approach to implement neural networks for engineering application, *Proc. of the 6th World Congress on Intelligent Control and Automation*, Dalian, China, 2006, 2939–2943.

[20] Y. Maeda & Y. Fukuda, FPGA implementation of pulse density hopfield neural network, *Proc. of International Joint Conference on Neural Networks*, Orlando, Florida, USA, 2007.

[21] S. Watanabe, J. Kitamichi, & K. Kuroda, A hardware algorithm for the minimum P-quasi clique cover problem, *International Conference on Field Programmable Logic and Applications*, Amsterdam, Netherlands, 2007, 139–144.

[22] D. Abramson, K. Smith, P. Logothetis, & D. Duke, FPGA based implementation of a Hopfield neural network for solving constraint satisfaction problems, *Proc. of the 24th Conference on EUROMICRO*, Vaesteraas, Sweden, 1998.

[23] A. Varma & Jayadeva, A novel digital neural network for the travelling salesman problem, *Proc. of the 9th International Conference on Neural Information Processing*, Orchid Country Club, Singapore, 2002.

[24] J. Hopfield, Neurons with graded response have collective computational properties like those of two state neurons, *Proceedings National Academy of Science 79*, 1982, 2554–2558.

[25] J. Hopfield & D.W. Tank, Neural computation of decision in optimization problems, *Biological Cybernetics*, *52*, 1985, 141–152.

[26] Y. Takefuji & K. Lee, A Near optimum parallel planarization algorithm, *Science*, *245*, 1989, 1221–1223.

[27] G. De Michelli, *Synthesis and optimization of digital circuits* (New York, NY, USA: McGraw-Hill, 1993).

[28] D. Thomas, E. Lagnese, R. Walker, J, Nestor, R. Rajan, & R. Blackburn, *Algorithmic and RT synthesis: The system architect's workbench* (Norwell, Massachusetts: Kluwer, 1990).

[29] Y. Takefuji, *Neural network parallel computing* (Boston: Kluwer, 1992).

[30] A. Hertz & D. de Werra, Using tabu search techniques for graph colouring, *Computing*, *39*(4), 1987, 345–351.

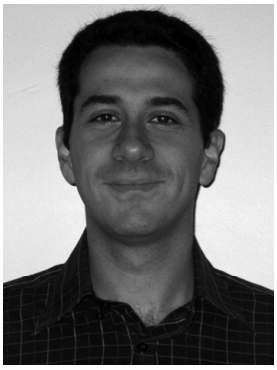
[31] M. Chams, A. Hertz, & D. de Werra, Some Experiments with simulated annealing for colouring graphs, *European Journal of Operational Research*, *32*(2), 1987, 260–266.

[a] H. Harmanani, A Neural Networks Algorithm for Data Path Synthesis, *International Journal of Computers and Electrical Engineering*, Elsevier Science, *29*(6), June 2003, 535–551.

[b] H. Harmanani, J. Hannouche, & N. Khoury, A Neural Networks Algorithm for the Minimum Coloring Problem using FPGAs, *Proceedings of the IASTED International Conference on Applied Simulation and Modeling (ASM 2003)*, Marbella, Spain, September 2003, 152–156.


## Biographies

*Haidar Harmanani* received his B.S., M.S., and Ph.D. degrees all in Computer Engineering from Case Western Reserve University, Cleveland, Ohio, in 1989, 1991, and 1994, respectively. He joined the Lebanese American University, Byblos, Lebanon, in 1994 as an assistant professor of computer science. Currently, he is an associate professor of computer science and the Chair of the Department of Computer Science and Mathematics at LAU, Byblos. He has been on the program committee of various international conferences including the IEEE NEWCAS Conference, the IEEE Midwest Symposium on Circuits and Systems, the IEEE International Conference on Electronics, Circuits, and Systems, the 14th IEEE International Conference on Microelectronics, and the IEEE Design Automation and Test in Europe. His research interests include electronic design automation, high-level synthesis, design for testability, and cluster parallel programming. He is a senior member of IEEE and ACM.



*Jean Hannouche* received his B.E. degree in Computer Engineering from the Lebanese American University in 2002, and the M.S. and Ph.D. degrees in Computer Engineering from Syracuse University, USA, in 2005 and 2009, respectively. His research interests include CAD for power and area minimization, CAD for place and route, parallel algorithms and neural networks. He is a member of the Phi Beta Delta Honor Society at Syracuse University since 2003, a member of the Golden Key Honor Society since 2007, and received the Best Graduate Student Award in 2009 from Syracuse.

*Nancy Khoury* received her B.E. degree in Computer engineering from the Lebanese American University in 2002, and the M.S. and Ph.D. degree in Computer Engineering from Syracuse University, USA, in 2005 and 2009, respectively. Her research interests are timing analysis and optimization in combinational circuits, high-level synthesis, CAD for place and route, high-level synthesis and neural networks. She is currently employed at Intel and is working as the leading average power analyzer for Intel's platforms. She is a member of the Phi Beta Delta Honor Society at Syracuse University since 2003, a member of the Golden Key Honor Society since 2007, and received the Best Graduate Student Award in 2009 from Syracuse.