Mapping Combinatorial Optimization Problems onto Neural Networks

J. RAMANUJAM
Dept. of Electrical & Computer Engineering
Louisiana State University, Baton Rouge, LA 70803

and

P. SADAYAPPAN
Department of Computer and Information Science
The Ohio State University, Columbus, OH 43210

ABSTRACT

Neural networks have been proposed as a model of computation for solving a wide variety of problems in fields as diverse as combinatorial optimization, vision, and pattern recognition. The ability to map and solve a number of interesting problems on neural networks motivates a proposal for using neural networks as a highly parallel model for general-purpose computing. We review this proposal, and show how to map a number of interesting combinatorial optimization problems from graph theory, VLSI placement, and operations research.

1 INTRODUCTION

The subject of combinatorial optimization encompasses a large set of problems that are central to many disciplines in science and engineering. Research in this area is aimed at optimizing functions of many independent variables that represent different parts of the configuration of a complex system. The function usually referred to as the cost function represents in some sense the "goodness" of the system configuration, and hence depends on the complex interaction among the parts of the system. Considering the number and importance of these problems, it is unfortunate that many of these are NP-complete, examples of which include the traveling salesman problem (TSP), graph partitioning, etc. Heuristic methods, which are problem-specific, have been developed which require running times that are polynomial functions of the size of the problems; since the problems involve many variables, heuristics which use time proportional to high-order polynomial on the number of variables are unacceptable. In addition, heuristics for one NP-complete problem are not necessarily effective for another. Simulated annealing (SA) [14, 15] has been proposed as a general technique for attempting to solve combinatorial optimization problems. Although SA has been effective in many practical problems such as those arising in the design of VLSI chips, TSP, and graph partitioning (GPP), in terms of the quality of solutions, it requires unacceptably large computing times. Furthermore, the simulated annealing procedure is inherently sequential, and efforts to parallelize it have not met with more than moderate success [15]. From an engineering point of view, one is usually interested in very good solutions and not necessarily the "best," often because of a certain degree of arbitrariness in the meaning of the "best" solution and how that meaning is translated into the form of the cost function. Also, it is highly desirable to arrive at good solutions in **short time.**

Neural networks have been proposed as a model of computation for solving a wide variety of problems in operations research, vision, speech and image processing, and pattern recognition [3, 4, 10, 11, 20]. In addition, from a parallel processing viewpoint, they form an attractive model for an efficient fine-grained distributed memory massively parallel machine for such domains. The ability to map and solve a number of useful problems on neural networks motivates the proposal for using neural networks as a highly parallel model for general-purpose computing.

Solving a combinatorial optimization problem on a neural net requires a mapping of the problem onto the neural networks in such a way that one can decipher a solution from the outputs of the neurons. In this paper, we consider mapping problems onto the Hopfield model of analog neurons. The aim of this paper is to illustrate that a variety of combinatorial optimization problems that have wide application in task allocation for distributed computing [3], the design of Very Large Scale Integrated (VLSI) circuits, and operations research [15] can be mapped onto neural networks. In Section 2, we review the neural model of computation; in Section 3, we outline and illustrate the various steps in the task of mapping using GPP; in the next section, we show mappings for several problems such as graph K-partitioning, vertex cover, maximum independent set and clique problems, the number partitioning problem, graph coloring and matching, and the set cover problem. Section 5 concludes with a discussion of computing model, the important issues in mapping various problems, and the relationship between the problem representation and complexity of the resulting network.

2 COMPUTATION USING NEURAL NETWORKS

We review the computing model underlying neural networks here. To solve problems using neural networks, one must cast them into a neural network model; we illustrate the capabilities of neural networks through the Hopfield analog neural network [10] defined by the **energy function**:

$$E(\vec{V}) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} T_{ij} V_i V_j - \sum_{i=1}^{N} V_i I_i \tag{1}$$

where the vector $\vec{V} = (V_1, \ldots, V_n)$ represents the state of the neural network. The V_i represent the output voltage of the N neurons which are the basic processors in the neural network; V_is are constrained to be between 0 and 1. T_{ij} represents the strength of the synaptic connection between neurons V_i and V_j with $T_{ii} = 0$ and I_i represents the input bias currents; the input voltage to the neurons is denoted as u_i. The evolution of the neurons is defined by

$$\frac{du_i}{dt} = -u_i + \sum_{j=1}^{N} T_{ij} V_j + I_i \tag{2}$$

$$V_i = g_i(u_i) \tag{3}$$

With symmetric connections (T_{ij}=T_{ji}), a local minimum of the energy function, E, is reached at the stable states. By proper choice of the gain function, the local minima of E can be constrained to occur with all the values of $V_i = 0$ or 1. This is especially useful when we are interested in digital solutions (0 or 1) to problems. The gain function is chosen to be

$$V_i = \frac{(1 + \tanh(u_i))}{2} \tag{4}$$

The stable states of the networks are completely determined by specifying the bias currents and the initial values of the input voltages because one could view the sets of differential equations as defining an initial value problem. The values T_{ij} define the set of all local minima of the quantity E, and the specific values of the steady-state output voltages obtained from the time evolution of the network are determined by the initial values of the input voltages u_i. In contrast to to the statistical hill-climbing technique known as simulated annealing (a powerful technique for solving many difficult combinatorial optimization problems), the equations of motions are deterministic, i.e., the steady state solutions are completely defined by the initial conditions. It has been shown that the equations of motion that define the behavior of the neural network is a mean field approximation to simulated annealing [3].

From a parallel processing viewpoint, neural networks form a definite proposal for an efficient small grain size distributed memory massively parallel machine dedicated to some special class of problems. Such a proposal is strengthened by the ability to map and solve problems of great interest in such diverse fields as artificial intelligence, engineering, and operations research. In subsequent sections, we propose mappings for such problems.

3 ISSUES IN MAPPING PROBLEMS ONTO NEURAL NETWORKS

In their pioneering paper, Hopfield and Tank [10] proposed a mapping of the traveling salesman problem onto neural networks. To map a problem onto a neural network, we need to perform the following steps:

1. Choose a representation scheme which allows the outputs of the neurons to be decoded into a solution to the problem.

2. Choose an energy function whose minimum value corresponds to the "best" solutions to the problem to be mapped.

3. Assign values to parameters in the cost function – this will determine the relative weights assigned to different parts of the cost function and this plays a *crucial* role in the kind of solutions that the network can find.

4. Derive connectivities (T_{ij}) and input bias currents (I_i) from the energy function; these should represent the instance of the specific problem to be solved.

5. Set up initial values for the input voltages (u_i) which completely determines the stable output voltages of the neurons in the network.

Steps $1--4$ are very important in mapping the problem onto the network and will play a critical role in deciding whether a specific problem can be solved easily using neural networks or not. In some cases, using this mapping procedure may require higher order neural networks for solving the problem. There is no direct method for mapping constrained optimization problems onto a neural network except through addition of terms in the energy function E which penalize violation of the constraints. In such a case, one has represent the energy function as a weighted sum of the problem function to be minimized and penalty for violating constraints; this implies that the extent to which the solutions from the network make sense depends on the weights and the problem itself and decides whether solutions that violate the constraints "moderately" are still "sensible." Thus step 3 plays a vital role in the mapping; usually, determining a good set of parameters for the cost function is at least as hard as solving the original problem. Hegde et. al [5] have presented an experimental approach to determining the parameters for the traveling salesman problem. The authors have proposed a modular feedback network that adapts the parameters to the instance of the problem being solved.

3.1 *GRAPH PARTITIONING NETWORK*

The graph partitioning problem (GPP) is an extensively studied combinatorial optimization problem because of its importance in applications like task allocation for distributed computing, design of Very Large Scale Integrated (VLSI) circuits etc. Given a graph $G = (V, E)$ with n vertices with vertex weights w_i and edges between vertices with weights e_{ij}, the problem is to partition the graph into two partitions of nearly equal weight (i.e., the sums of the weights of vertices assigned to the two partitions are nearly equal) such that the *cutsize* (the number of edges with an endpoint in each partition) is minimized. We discuss graph bisection here and refer the reader to [17] for experimental results on the mapping.

In the steady state of the network, the vertices in the graph whose corresponding neurons all have the same output belong to the same partition. Since the gain function forces the output of the neurons to be 0 or 1, we end up with only two partitions, one which consists vertices corresponding to those neurons whose output is 0 and the other whose output is 1. The energy function for for GPP is as follows:

$$E = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} e_{ij}(V_i + V_j - 2V_iV_j) + \sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j (1 - V_i - V_j + 2V_iV_j) \quad (5)$$

The first term minimizes the weighted sum of edges which belong to the *cut* and the second term will have a minimum value when the sum of vertex weights assigned to the two partitions are equal. From the E function it is easy to derive values of T_{ij} and I_i as follows:

$$T_{ij} = 2e_{ij} - 4w_iw_j \qquad \text{and} \qquad I_i = -\sum_{j=1}^{n} e_{ij} + 2w_i\sum_{j=1}^{n} w_j \qquad (6)$$

We chose to illustrate the procedure for mapping problems onto a neural network using GPP because of its simplicity. The next section deals with mapping of complex problems.

4 MAPPING OF VARIOUS COMBINATORIAL OPTIMIZATION PROBLEMS

In this section, we briefly describe the mapping of various combinatorial optimization problems onto neural networks. The mappings are used to illustrate the feasibility of modeling a large class of interesting and important problems using neural networks, thereby strengthening the viability and importance of neural networks for deriving rapid solutions to many practical problems.

4.1 *GRAPH K-PARTITIONING (GKP)*

The problem is the same as defined in the last section except that the graph is to be divided to p partitions where p is in general not a power of 2. In this case, we model the problem as an assignment problem where each node should belong to exactly one partition. We use a total of $N = np$ neurons where n is the number of nodes in the graph and p is the number of required partitions. Let V_{ij} denote the assignment of node i to partition j; if V_{ij} then node i is assigned to partition j otherwise not. We require that each node be assigned to exactly one partition and that with equal sized-partitions, the weighted sum of edges which belong to the *cut* (i.e., edges whose endpoints belong to different partitions) is minimized. The energy function for this problem is defined as

$$\begin{aligned} E \quad = \quad & \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{p}\sum_{k=1,k\neq j}^{p} V_{ij}V_{ik} + \frac{1}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{p} V_{ij} - n\right)^2 + \\ & \frac{1}{2}\sum_{i,j=1}^{n}\sum_{k,l=1}^{p} e_{ij}(V_{ik} + V_{jl} - 2V_{ik}V_{jl}) + \sum_{k=1}^{p}\sum_{i=1}^{n}\sum_{j=1}^{n} V_{ik}V_{jk}w_iw_j \quad (7) \end{aligned}$$

from which we could derive the values of $T_{ik,jl}$ and I_i as follows:

$$T_{ij,kl} = -\delta_{ij}(1-\delta_{kl}) - 1 + 2e_{ij} - 2w_iw_j \qquad \text{and} \qquad I_i = +n - \sum_{j=1}^{n} e_{ij} \quad (8)$$

The above energy function maps the GKP onto the Hopfield type network. On setting values of the initial input voltages of the neurons, the time evolution of the network will result in steady output of the neurons from which we can infer a solution to the GKP.

4.2 *A HIGHER ORDER NETWORK FOR GRAPH K-PARTITIONING (GKP)*

In the case of partitioning a graph into p partitions, a network that uses $n \log_2 p$ neurons in stead of np neurons can be derived (for p a power of 2) at the expense of higher order terms in the cost function, which are prohibitively expensive from the point of silicon implementation of networks. The idea behind the network is that in order to specify the partition to which a node is assigned, one needs $\log_2 p$ bits each of which takes on binary values. Let V_{ij} indicate whether bit j is on or not in the assignment of node i; if bit j is on, then $V_{ij} = 1$ else $V_{ij} = 0$. Thus, the partition to which node i is assigned is calculated as

$$\sum_{j=1}^{\log_2 p} 2^{j-1} V_{ij}$$

Any assignment of values to V_{ij} is a legal assignment; therefore, there is no need for terms in the cost function to enforce the syntactic validity of solutions. An edge between two nodes contributes to the cut even if any one of the bits in the corresponding positions in the assignment of the end-points of the edge is different. The cost of the cut is thus,

$$\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} e_{ij} \left(1 - \prod_{k=1}^{\log_2 p} (1 - V_{ik} - V_{jk} + 2V_{ik}V_{jk}) \right)$$

Let W_j indicate the sum of weights of vertices assigned to partition j. If the partitions have to balanced, then the sum of the squares of deviations from the ideal weight of the partitions must be a minimum – its minimum value is 0 and occurs when all partitions have equal weight. Thus, in terms of weights of partitions,

$$\sum_{j=1}^{p} (W_j - \overline{W})^2$$

has to be minimized; but

$$\sum_{j=1}^{p} (W_j - \overline{W})^2 = \sum_{j=1}^{p} \left((W_j)^2 - 2\overline{W}W_j + (\overline{W})^2 \right) = \sum_{j=1}^{p} (W_j)^2 - p(\overline{W})^2$$

Since $p(\overline{W})^2$ is constant, we attempt to minimize $\sum_{j=1}^{p} (W_j)^2$. Thus, the balance constraint is represented as

$$\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j \prod_{k=1}^{\log_2 p} (1 - V_{ik} - V_{jk} + 2V_{ik}V_{jk})$$

Fox and Furmanski [3] use a neural network that consists of $n \log p$ neurons for the problem of mapping n tasks onto p-node hypercube.

4.3 *VERTEX COVER, MAXIMUM INDEPENDENT SET AND MAXIMUM CLIQUE PROBLEMS*

Given a graph $G = (V, E)$, one is often interested in finding subsets of the set V of vertices of G which possess some predefined property; we consider only undirected graphs here. The problems considered in this subsection are defined below:

- A *vertex cover* is a subset C of the vertices of the graph G such that all edges of G are adjacent to at least one vertex in the set C. The vertex cover problem is to find the vertex cover of the least size given a graph $G = (V, E)$.
- An *independent set* is a subset of the vertices of G such that there is no edge between any pair of vertices which belong to that subset. The maximum independent set problem is find the largest independent set of a graph G.
- A *clique* is a subset of the vertices of the graph G such every pair of vertices in that subset is connected by an edge. The maximum clique problem is to find the largest clique of a graph G.

These problems find important applications in project scheduling, cluster analysis, location of facilities and other problems in operations research.

We assume that the graph G is given in its adjacency matrix form, i.e., given as a matrix $A = (a_{ij})$ where $a_{ij} = 1$ if there is an edge between vertices i and j, otherwise $a_{ij} = 0$. In the case of the vertex cover problem, we associate a neuron with each vertex in the graph; the output voltage of the ith neuron, V_i is 1 if vertex i belongs to the vertex cover, else $V_i = 0$. Thus, for an N-vertex graph, we use $N = n$ neurons from which we can decipher the solution as the set of vertices of the graph whose associated neurons have a 1 output in the steady state. In the vertex cover and the other problems discussed here, a 1 output indicates membership in the subset of interest and in all the cases we use one neuron for each vertex in the graph G.

4.3.1 *Vertex Cover*

The energy function E that we have arrived at for the vertex cover problem is given below - the first term minimizes the number of vertices in the vertex cover and the second term enforces the syntactic constraint that the nodes included in the cover do in fact form a vertex cover i.e., every edge in the graph has a vertex adjacent to it in the vertex cover and those edges which have both end-vertices in the vertex cover are not counted twice; we want to maximize the set of edges covered. As for the first term, we could also have used $\sum_{i=1}^{n} V_i$ as opposed to $\sum_{i=1}^{n}\sum_{j=1}^{n} V_i V_j$; we chose the latter because we wanted to emphasize the minimality of the vertex cover i.e., the product form introduces an inhibitory component to the connectivity among the neurons which the sum form does not. The energy function, E is

$$E = A\sum_{i=1}^{n}\sum_{j=1}^{n} V_i V_j - \frac{B}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{n}(1-V_i)(1-V_j)a_{ij}\right) \tag{9}$$

where A and B are positive constants whose values decide the relative importance of the constraints and the function to be minimized. The first term ensures minimality of the cover; the second term uses the fact that if a subset of nodes $C \subseteq V$ forms a vertex cover, its complement $V - C$ forms an independent set in graph G. The second term will be zero, if those nodes which are not in the vertex cover have no edges between them. On rearrangement of terms in the E function and on dropping constant terms we get

$$E = A\sum_{i=1}^{n}\sum_{j=1}^{n} V_i V_j - B\sum_{i=1}^{n}\sum_{j=1}^{n} V_i a_{ij} - \frac{B}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} V_i V_j a_{ij} \tag{10}$$

From the energy function, the values of T_{ij} and I_i can be derived as follows:

$$T_{ij} = -2A - Ba_{ij} \qquad \text{and} \qquad I_i = B\sum_{j=1}^{n} a_{ij} \tag{11}$$

4.3.2 *Maximum Independent Set*

For the maximum independent set problem, the steady state solution to the network of neurons guided by the energy function E find maximal size subset of vertices which are independent. The energy function is given as

$$E = -A\sum_{i=1}^{n}\sum_{j=1}^{n} V_i V_j + \frac{B}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} V_i V_j a_{ij} \tag{12}$$

where A and B are positive constants whose values decide the relative importance of the constraints and the function to be minimized. The first term minimizes the number of vertices in the independent set and the second term enforces the constraint that the vertices included in the subset do in fact form a independent set i.e., there is no edge in the graph between any pair of vertices that are in the independent set. In fact, if there are any edges between the vertices in the chosen subset, it is penalized heavily by the choice of a high positive value of B. From the energy function, the values of T_{ij} can be derived as follows:

$$T_{ij} = 2A - Ba_{ij} \tag{13}$$

In this case the bias currents can be set to zero. If the first term in the energy function is chosen as $\sum_{i=1}^{n} V_i$, we would have non-zero bias currents; the relative merit of these two formulations will become clearer through experiments.

4.3.3 *Maximum Clique*

Given a graph $G = (V, E)$ if a set S of vertices form a clique then they form an independent set of G_c, the complement of G where the complement G_c has the same vertices as G and the edges of G_c are those which do not occur in G; in the case of G given in the adjacency matrix form $A = (a_{ij})$, the adjacency matrix of the complement is given as $A_c = (1 - a_{ij})$. In order to find solutions to the maximum clique problem, we could use the same neural network that was defined for the maximum independent set problem previously with a_{ij} replaced with $1 - a_{ij}$.

4.4 *NUMBER PARTITIONING*

The number partitioning problem is defined as follows: Given a set of n numbers $r_1, r_2, \ldots, r_n \in S$, partition them into two sets S_1 and S_2 such that the cost function, given by

$$C(S_1, S_2) = \left| \sum_{i \in S_1} r_i - \sum_{j \in S_2} r_j \right| \tag{14}$$

is minimized. The neural network designed to solve the number partitioning problem uses $N = n$ neurons; the output voltage of each neuron V_i indicates the set to which number r_i is assigned. $V_i = 0$ indicates that r_i is assigned to the set S_1 and $V_i = 1$ indicates that r_i is assigned to the set S_2. The required cost function is minimized when the sum of the squares of the total value of the numbers assigned to the different partition is minimized. With this in mind, the following energy function E models the number partitioning problem using neural networks:

$$E = \sum_{i=1}^{n} \sum_{j=1}^{n} (1 + 2V_iV_j - V_i - V_j) r_i r_j \tag{15}$$

and since $\sum_{i=1}^{n}\sum_{j=1}^{n} r_i r_j$ is a constant we rewrite the energy function as

$$E = \sum_{i=1}^{n}\sum_{j=1}^{n}(2V_iV_j - V_i - V_j)r_ir_j \tag{16}$$

The term $(1 + 2V_iV_j - V_i - V_j)$ ensures that only numbers which belong to the same partition contribute to the sum. From the above energy function, the T_{ij} and I_i values are derived easily and are shown below:

$$T_{ij} = -4r_ir_j \qquad \text{and} \qquad I_i = \sum_{j=1}^{n} r_j \tag{17}$$

The 0/1-knapsack problem is a special case of the number partitioning problem, and finds a lot of application in scheduling problems. Thus the mapping of the partitioning problem opens the door for a variety of problems in operations research as well.

4.5 *MAXIMUM MATCHING*

The *maximum matching* problem is a generalization of the network flow problems which have a large number of practical applications. The problem has attracted the interest of researchers during the last two decades because it can be stated simply and has an intuitive appeal, in addition to the fact that it has many applications. A *matching* M of a graph is a subset of the edges with the property that no two edges of M share the same vertex. Given a graph $G = (V, E)$, the matching problem is to find matching of the highest cardinality. In this case, we assume that the graph G is represented as an *incidence matrix*. Given an undirected graph G of n vertices and m edges, the *incidence matrix* of G is denoted by $B = (b_{ij})$ and is an $n \times m$ matrix defined as follows: $b_{ij} = 1$ if edge a_j is incident at vertex x_i, otherwise it is zero. In addition we assume that edges have the same weight (equal to 1) associated with them so that the neural networks find the maximum cardinality matching. In the case that edge weights are allowed to vary, the network would find the maximum weight matching. We model the maximum cardinality matching below. We associate a neuron with each edge in the graph to indicate if the edge belongs in the matching or not. Hence, we want to maximize $\sum_{j=1}^{m} V_j$ subject to the constraint that the edges that are included in the set form a matching i.e., $\sum_{j=1}^{m} b_{ij}V_j \leq 1 \; \forall i = 1, \cdots, n$. This constraint can be expressed as a term whose value is zero if the edges form a matching i.e., no two edges in the matching are incident at the same vertex – this is expressed as a penalty

$= \sum_{i=1}^{m} \sum_{j=1}^{m} V_i V_j \sum_{k=1}^{n} b_{ki} b_{kj}$. We derive the energy function as follows:

$$E = -\sum_{j=1}^{m} V_j + \sum_{i=1}^{m} \sum_{j=1}^{m} V_i V_j \sum_{k=1}^{n} b_{ki} b_{kj} \tag{18}$$

From this one can derive the $\{T\}$ and I_i's to be given by

$$T_{ij} = -2 \sum_{k=1}^{n} b_{ki} b_{kj} \qquad \text{and} \qquad I_i = 1 \tag{19}$$

We note that problems in operations research (OR) such the transportation problem and the assignment problem are special cases of the weighted matching problem and the network proposed for matching can be used with these OR problems.

4.6 *SET COVER PROBLEM*

The Set Cover Problem (SCP) derives its name from the following set-theoretic interpretation: Given a set $R = \{r_1, \dots, r_m\}$, and a family $\mathcal{S} = \{S_1, \dots, S_n\}$ of sets $S_j \subset R$ any sub-family $\mathcal{S}' = \{S_{j1}, S_{j2}, \dots, S_{jk}\}$ of $\mathcal{S}$ such that

$$\bigcup_{i=1}^{k} S_{ji} = R$$

is referred to as a *set-covering* of R, and the S_{ji} are referred to as the *covering sets.* If, with each $S_j \in \mathcal{S}$, we associate a positive cost $c_j = |S_j|$, the size of the set, the SCP is to find a covering of R which has the minimum cost, the cost of $\mathcal{S}' = \{S_{j1}, S_{j2}, \dots, S_{jk}\}$ being $\sum_{i=1}^{k} c_{ji}$.

In this case, the set cover problem instance can be represented by a an adjacency matrix $A = a_{ij}$ of m rows and n columns, whose entries are defined as

$$a_{ij} = \begin{cases} 0 & \text{if } \mathrm{r_i} \text{ is not in } \mathrm{S_j} \\ 1 & \text{if } \mathrm{r_i} \text{ is in } \mathrm{S_j} \end{cases}$$

Given the adjacency matrix A, the set-covering problem is to find the minimum number of columns that cover all the rows.

We associate a neural variable V_j with each set S_j to indicate if the set is in the cover. We minimize $\sum_{j=1}^{n} V_j$ (the cardinality of the the union of the sets in the cover) subject to the constraint that all the elements r_i of R are included in some member of the cover i.e., $\forall i = 1, 2, \dots, m$, $\sum_{j=1}^{n} a_{ij} V_j \geq 1$. We derive the energy function as follows:

$$E = \alpha * \sum_{j=1}^{n} V_j \quad + \quad \beta * \sum_{i=1}^{m} \prod_{j=1}^{n} \{(1 - a_{ij}) + (1 - V_j)\} \tag{20}$$

where α and β are positive constants. The first term in the energy function represents the cardinality of the union of the sets in the cover and the second term represents the penalty for lack of complete covering. The second term will have a value 0, which is its lowest, in the case of complete cover.

The *product* form in the energy function requires a k-th order neural network, where $k = \max_i \sum_{j=1}^{n} a_{ij}$, the largest number of sets S_j that cover a single element $r_i \in R$. A k-th order neural network admits up to k-way connections among the neurons [21]. With symmetric k-way connections, the energy function of such a network can be shown to be non-increasing with time. Higher order networks are extremely difficult to realize in silicon. In the next subsection, we discuss graph coloring, which is closely related to set covering; though, the problems are quite similar when formulated as 0/1 linear programming problems, the size and connectivity of the networks required are vastly different.

4.7 *GRAPH COLORING*

Given a graph $G = (V, E)$, one is often interested in finding the minimum number of subsets of the set V of vertices of G such that no two vertices that are connected by an edge belong to the same subset. The vertices which belong to each subset can be thought of as being assigned the same color; thus, no two vertices that have an edge connecting them can be colored using the same color – hence the name graph coloring. Let V_{ik} indicate that vertex i is colored using color k. From a result in graph theory, the number of colors necessary is less than or equal to one plus the maximum vertex degree, say ν i.e., the maximum number of edges incident at any vertex in the graph [2]; in general, $1 \leq \nu \leq n$ where n is the number of vertices in the graph. The condition that a coloring is legal is represented by the term

$$\frac{1}{2}\sum_{k=1}^{\nu}\sum_{i=1}^{n}\sum_{j=1}^{n} V_{ik}V_{jk}e_{ij}$$

This term will have a value zero when the coloring is legal, otherwise the term is positive. Given that $\sum_{k=1}^{\nu}\sum_{i=1}^{n} V_{ik}$ is constant, it can be shown that $\sum_{k=1}^{\nu}\sum_{i=1}^{n}\sum_{j=1}^{n} V_{ik}V_{jk}$ is maximal if a minimal number of subsets is non-empty *i.e.,* minimal number of colors are used. Thus the cost function used is

$$E = \frac{1}{2}\sum_{k=1}^{\nu}\sum_{i=1}^{n}\sum_{j=1}^{n} V_{ik}V_{jk}e_{ij} - \sum_{k=1}^{\nu}\sum_{i=1}^{n}\sum_{j=1}^{n} V_{ik}V_{jk} \qquad (21)$$

Thus,

$$T_{ij} = 1 - \frac{e_{ij}}{2} \tag{22}$$

and the bias currents may be set to zero.

When formulated as 0/1 linear programming problems, set cover and graph coloring are similar but their neural embeddings are quite different.

4.8 *RELATED WORK*

In addition to the work by Hopfield and Tank [8, 9, 10, 11], several researchers have addressed the problem of mapping combinatorial optimization problems onto neural networks. Wilson and Pawley [22] demonstrated the inadequacy of the Hopfield and Tank approach for the traveling salesman problem; they showed that as network size increased, either the solutions generated were invalid tours, or the network does not find the global optima. Hertz et al. [7] present an alternate energy function for the traveling salesman problem, which scales well with network size; in addition, they present mappings for other combinatorial optimization problems. Hellstrom and Kanal [6] present a method of mapping problems onto first-order asymmetric neural networks; they illustrate the use of such non-Hamiltonian energy functions in mapping the knapsack problem. Maa and Shanblatt [16] address the shortcoming of the Hopfield and Tank approach for embedding constrained optimization problem; they present a two-phase approach to alleviate this problem. Protzel et al. [18] discuss the performance and fault-tolerance of neural networks that solve optimization problems. They model faults as additional constraints and provide experimental evidence of graceful degradation of the network; this is an important issue in hardware realizations of neural networks. Ali and Kamoun [1] have designed neural networks for adaptive routing in packet-switched computer networks. The recent work of Kak [13] on complex-valued neuron models appears to be promising for combinatorial optimization problems; the ability to exploit global information will be useful in solving constrained optimization problems.

5 SUMMARY

In this paper, we have reviewed the basic model of neural computation and showed how to solve many problems of interest using this model. With the advent of parallel processing and the ability to design electronic "neurons" through advances in semiconductor processing, neural networks seem to be a viable model of an efficient fine-grained distributed-memory massively parallel machine dedicated to some special class of problems. We addressed the issue of the ability to map problems onto and solve them using neural networks and showed how to map a class of combinatorial optimization problems. Though a number of graph problems were considered, the input representation plays a role in the choice of the cost function – some lending a "natural" representation of the problems – resulting in easily decipherable neuromorphic data structures. More work needs to be done towards identifying classes of problems from the point of view of the structure of the network. It is also important to derive networks which use second order connections, if one hopes to derive VLSI implementations for the problems – for example, the set cover problem as formulated here requires higher order connections; this requirement should also influence the choice of the energy function and hence the neural representation of the problem.

It is noted that in some cases, as in vertex cover, competition can be introduced among neurons by proper choice of the energy function. It can be seen that the mappings have been motivated partly by 0/1 linear programming and partly by "pictures" or "images" of the solutions given the chosen problem representations (as in TSP). Compactness of representations (as in the $n \log_2 n$ network for GKP) results in higher order networks. In addition, it is not easy for the network to afford an "explanation" of the solution in the case of higher order networks. Inquiries into trade-offs, if any, among compactness of representation, ability to explain and ease of VLSI implementation should be undertaken. The complex neuron model of Kak [13] is a promising approach for optimization problems and needs a detailed study.

Typically, constrained optimization problems are mapped onto neural networks as unconstrained optimization problems with parameterized penalty terms in the cost function to account for violation of constraints; in many cases, the quality of the solution depends crucially on parameters of the network. Thus, one is faced with the more complex problem of tuning the network by adjusting the parameters – which has thus far been ad hoc. Modular feedback network architectures [19] in conjunction with complex neuron models may lead to dynamic parameter determination while attempting to the solve the problem.

References

[1] M. Ali and F. Kamoun, Neural networks for shortest path computation and routing in computer networks, *IEEE Transactions on Neural Networks* 4(6):941-954 (Nov. 1993).

[2] R. Brooks, On coloring the nodes of a network, *Proc. Cambridge Philosophical Society* 37:194-198 (1941).

[3] G. Fox and W. Furmanski, Load balancing by a neural network, Caltech Concurrent Computation Project Report no. 363, 1986.

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

[5] S. Hegde, J.Sweet and W. Levy, Determination of parameters on a Hopfield/Tank computational network, *Proc. 2nd Annual IEEE International Conference on Neural Networks* San Diego, 1988.

[6] B. Hellstrom and L. Kanal, Knapsack packing networks, *IEEE Transactions on Neural Networks* 3(2):302–307, (Mar. 1992).

[7] J. Hertz, A. Krogh, and R. Palmer, *Introduction to the Theory of Neural Computation* Addison-Wesley, Redwood City, CA, 1991.

[8] J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, *Proceedings of the National Academy of Sciences USA* 79:2554-2558 (1982).

[9] J. Hopfield, Neurons with graded response have collective computational properties like those of two-state neurons, *Proceedings of the National Academy of Sciences USA* 81:3088-3092 (1984).

[10] J. Hopfield and D. Tank, Neural computation of decisions in optimization problems, *Biological Cybernetics* 52:141-152 (1985).

[11] J. Hopfield and D. Tank, Computing with neural circuits: A model, *Science* 233:625-633 (1986).

[12] D. Johnson, C. Aragon, L. McGeoch and C. Schevon, Optimization by simulated annealing: An experimental evaluation, *Operations Research* (1989).

[13] S. C. Kak, Feedback neural networks: New characteristics and a generalization, *Circuits Systems and Singal Processing* 12(2):263-278 (1993).

[14] S. Kirkpatrick, C. Gelatt and M. Vecchi, Optimization by simulated annealing, *Science* 220:671-680 (1983).

[15] P.J.M. van Laarhoven and E.H.L. Aarts, *Simulated annealing: Theory and applications* D. Reidel Publishing Company, 1987.

[16] C. Maa and M. Shanblatt, A two-phase optimization neural network, *IEEE Transactions on Neural Networks* 3(6):1003-1009 (Nov. 1992).

[17] C. Petersen and J. Anderson, Neural networks and NP-complete optimization problems: A performance study on the graph bisection problem, *Complex Systems* 2(1):59-89 (Feb. 1988).

[18] P. Protzel, D. Palumbo and M. Arras, Performance and fault-tolerance of neural networks for optimization, *IEEE Transactions on Neural Networks* 4(4):600-614 (July 1993).

[19] J. Ramanujam and P. Sadayappan, Parameter identification for constrained optimization using neural networks, *Proceedings of 1988 Connectionist Models Summer School* Morgan Kaufman, San Mateo, CA, June 1988, pp. 154-161.

[20] D. Rumelhart and J. McClelland, *Parallel distributed processing: Explorations in the microstructure of cognition* vol. 1, MIT Press, 1986.

[21] T. Sejnowski, Higher-order Boltzmann machines, in *Neural Networks for Computing, AIP Conference Proceedings 151*, 1986, pp. 398-403.

[22] G. Wilson and G. Pawley, On the stability of the traveling salesman problem algorithm of Hopfield and Tank, *Biological Cybernetics* 58:63-70 (1988).