CHAPTER IV

Combinatorial Optimization by Neural Networks

Several authors have suggested the use of neural networks as a tool to provide approximate solutions for combinatorial optimization problems such as graph matching, the traveling salesman problem, task placement in a distributed system, etc.

In this chapter, we first give a brief description of combinatorial optimization problems. Next we explain in general how neural networks can be used in combinatorial optimization and then introduce Hopfield network as optimizer for two well known combinatorial optimization problems: the graph partitioning and the traveling salesman. Hopfield optimizer solves combinatorial optimization problems by gradient descent, which has the disadvantage of being trapped in local minima of the cost function.

The efficiency of neural networks in solving of NP_hard combinatorial optimization problems has been investigated by several researchers [Bruck and Goodman 88, 90, Yao 92]. It has been shown that even finding approximate solutions to NP_hard problems is not an easy task. By the use of techniques of complexity theory, it has been proved that no network of polynomial size exists to solve the traveling salesman problem unless NP=P [Bruck and Goodman 1990]. However, their parallel nature and good performance in finding approximate solution make the neural optimizers interesting.

4.1 Combinatorial Optimization Problems

The problems typically having a large but finite set of solutions among which we want to find the one that minimizes or maximizes a cost function are often referred as ***combinatorial optimization problems.*** Since any maximization problem can be reduced to a minimization problem simply by changing the sign of the cost function, we will consider only the minimization problem with no loss of generality. An ***instance*** of a combinatorial optimization problem can be formalized as a pair (S,g). The ***solution space,*** denoted S, is the finite set of all possible solutions. The ***cost function,*** denoted by g, is a mapping from the set of solutions to real numbers, that is, g: $S \rightarrow$R [Aarts and Korst 89]. In the case of minimization, the problem is to find a solution $S^* \in S$, called ***globally-optimal solution***, *which* satisfies

$$S^* = \min_{S_i \in S} g(S_i). \quad (4.1.1)$$

Notice that for a given instance of the problem, such an optimal solution may not be unique.

Optimization problems can be divided into classes according to the time required to solve them. If there exists an algorithm that solves the problem in a time that grows only polynomially with the size of the problem, then it is said to be polynomial. The set of polynomial time problems, denoted P, is a subclass of another class called NP. Here NP stands for ***non-deterministic polynomial,*** implying that a polynomial time algorithm exists for a ***nondeterministic Turing machine***. However for the problems in NP but not P, there exists neither a polynomial time algorithm for deterministic ***Turing machine*** (although it exists for nondeterministic Turing Machine) nor a proof the non-existence of such an algorithm. In spite of unavailability of polynomial time algorithms to solve this

kind of problems, a "guess" of the solution can be tested in polynomial time to find out whether or not it is the right one.

Exercise: Explain nondeterministic polynomial problems in terms of Turing machine very briefly.

An important subclass of NP is the NP_complete problems. They are problems in NP and characterized by the fact that each problem in the class can be reduced to any other member in polynomial time. Therefore, if one could find a deterministic algorithm that solves one of the NP_complete problems in polynomial time, then all of the NP_complete problems could be solved in polynomial time. In that case, P and NP_complete would be the same class. The probable situations are sketched in Figure 4.1. Emprically, the time it takes to solve an NP_complete problem tends to scale exponentially with the size of the problem [Hertz et al 91, Garey and Johnson 79]

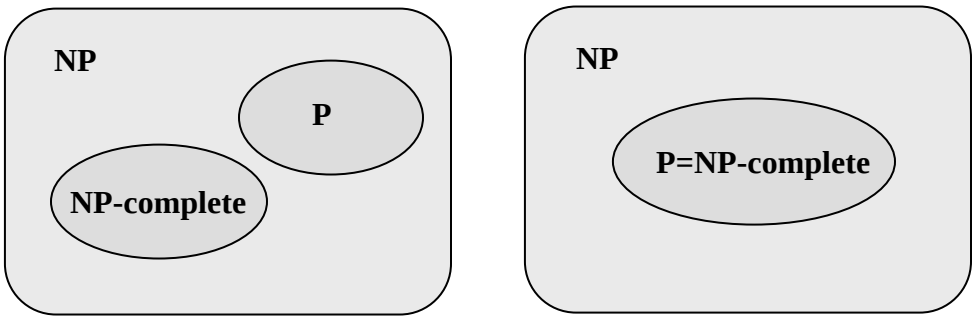

Figure 4.1 The class of NP problems a) assuming that P≠NP
b) if any p∈NP_complete becomes p∈P, then it implies P=NP

Exercise: Explain the reduction of one problem to another in polynomial time.

A combinatorial optimization problem of a major theoretical and practical interest, is the ***traveling salesman*** problem (TSP), and it has been subject of much work [Lawler et al 85]. This problem is NP_complete, and therefore computationally intractable for large instances of the problem. It is of great practical use in various important areas such as

circuit placement in VLSI, tool motion in manufacturing, network design etc. Thus the development of methods searching for solutions that are close to the optimum, and yet not excessively time consuming, is the source for continued research. In the TSP, the shortest closed path traversing each city under consideration exactly once is searched. For TSP, the number of cities determines the size of the problem (Figure 4.2).

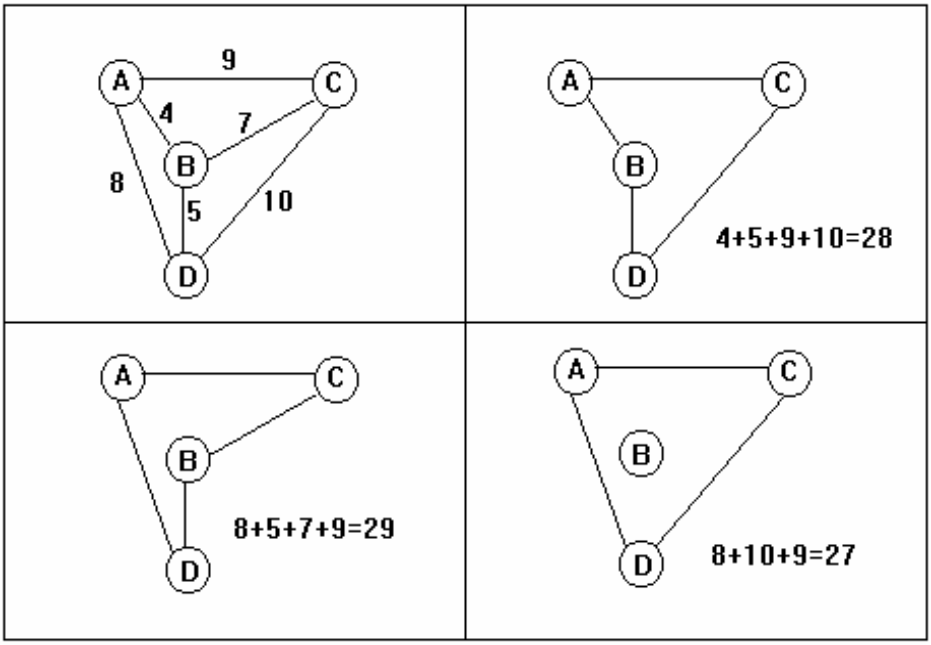

Figure 4.2 The traveling salesman problem a) an instance with 4 cities b) the optimum solution b) a nonoptimum solution c) non feasible solution having some unvisited cities

Another problem that we will consider in this chapter because of its simplicity in designing a neural optimizer is the vertex cover problem. It is also an NP_complete problem, therefore no efficient algorithms for its exact solution is available when the number of nodes in the graph is large. The problem size is determined by the number of nodes in the graph for which a minimum cover is searched.

The formal problem can be stated as follows: Let $G=(V,E)$ be a graph where $V=\{v_1, v_2,..,v_N\}$ is the vertices and $E=\{(v_i,v_j)\}$ is the edges of the graph. A ***cover*** C of G is a subset of V such that for each edge (v_i, v_j) in E, either v_i *or* v_j is in C. A ***minimum cover***

of G is a set C^* such that the number of nodes in C^* is the minimum among all the covers of G, that is $|C^*| \leq |C|$.

For exaple for the sam ple graph given in Figure 4.3, the covers are C_1=(a,b,c,d,e), C_2=(a,b,c,d), C_3=(a,b,c,e), C_4=(a,b,d,e), C_5=(a,c,d,e), C_6=(b,c,d,e), C_7=(a,b,e), C_8=(a,d,e), C_9=(b,c,d), C_{10}=(b,c,e), C_{11}=(b,d,e), C_{12}=(b,e) and the minimal cover is C_{12}=(b,e).

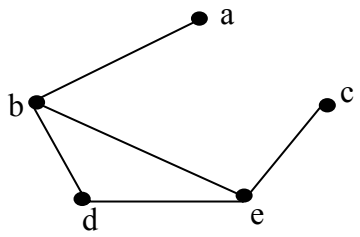

Figure 4.3 A sample graph

If we have to solve an NP _complete problem , then a very long com putation m ay be needed for an exact solution. The optim um solution of the vertex cover problem can be obtained by enum erating all the covers and then selecting the m inimum one. However such an enum erative search for the exact optim um solution have a tim e com plexity of $O(2^n)$, where n is the num ber of vertices in the graph. Being an NP com plete problem, finding the exact m inimum cover of G is not practical when the num ber of vertices is very large. Thus, in some cases, approximate algorithms are preferred [Will 86].

Exercise: Explain what approximate solutions may be used for vertex cover problem.

A heuristic solution to the problem using a greedy approach m ay be established as follows: First, the node having the highest de gree in G is selected and included in C^+, that is the cover being generated. Then, the node and all its adjacent edges all together with the related term inal nodes are rem oved from G and the procedure is repeated until all nodes in G have been removed.

4.2 Mapping an Optimization Problems onto Neural Networks

Solving a com binatorial optim ization problem aim s to f ind the "best" or "optim al" solution am ong a finite or countably infinite num ber of alternative solutions. As an alternative to the conventional optim ization methods, neural networks are used for the solution of com binatorial optim ization problem s. In general, a neural optim izer is a neural network whose neurons are affecti ng the problem solution. For instance neurons affecting the (city, position) pair of the tour can be used in the neural optim izer for solving the traveling salesm an problem . If a neuron is "on", this im plies that the corresponding city should be visite d in the given position in the ***approximately optimal*** solution. Then strongly inhibitory links are established between neurons, which represent incompatible elem ents of the solution; for ex ample, a city should not be visited twice, and a position should not be occupied by two di fferent cities. Furtherm ore, inhibitory links representing the cost are placed between neurons. For exam ple, the intensity of inhibitory links can represent the distances between cities in the traveling salesm an problem. Once the m odel is set up, it is allowed to relax dynam ically to a steady-state which should be of "m inimum energy" re presenting a quasi-m inimal cost solution [Hopfield and Tank 85, Gelenbe 94].

Hopfield network, Boltzm ann m achine, m ean field network, Gaussian m achine and several other neural networks can be used as neural optim izers. The units in these networks tend to optim ize a global functi on of the state space, by using only local information. Mean Field, Boltzm ann and Gaussian machines are stochastic in nature and allow escaping from local optima.

In the following, how neural networks can be used to solve com binatorial optim ization problems is explained in general: An instan ce of a com binatorial optim ization problem can be considered as a tuple (S,S',g) where S *is* the *finite* set of solutions; S' is the set of feasible solutions that satisf y the constraints of the problem ; and g: $S \rightarrow R$ is the cost

function assigning a real value to each solution. The aim is to find a feasible solution for which the cost function is optimal [Aarts and Korst 89].

In order to use a neural optimizer to solve combinatorial optimization problems, the state space of the network is mapped onto the set of solutions. The state space X of a neural optimizer is the set of all possible state vectors $\mathbf{x}$ whose components correspond to the neuron outputs. For this purpose, first the given problem is formulated as a 0-1 programming problem. Then, a neural network is defined such that the state of each unit determines the value of a 0-1 variable. Thus, the neural network implements a bijective (one to one and onto) function $m: X \rightarrow S$. The next step is to determine the strengths of the connections such that the energy function is order-preserving.

The energy function E of a neural network that implements a minimization problem (S,S',g) is called ***order-preserving*** if

$$g(m(\mathbf{x}^k)) < g(m(\mathbf{x}^l)) \Rightarrow E(\mathbf{x}^k) < E(\mathbf{x}^l). \qquad (4.2.1)$$

for any $\mathbf{x}^k, \mathbf{x}^l \in X$ with $m(\mathbf{x}^k), m(\mathbf{x}^l) \in S'$

Exercise: Explain order preservation in terms of traveling salesman problem.

Another desired property of the network is feasibility. Let X^* to denote the set of stable states of a neural network. The energy function E of the neural network is called ***feasible*** if each local minimum of the energy function corresponds to feasible solution, that is

$$m(X^*) \subseteq S' \qquad (4.2.2)$$

where

$$m(X^*) = \{ S_i \in S \mid \exists \mathbf{x}^k \in X^* : m(\mathbf{x}^k) = S_i \} . \qquad (4.2.3)$$

Feasibility of the energy function implies that the solution achieved by the network will always be a feasible one, since a neural optimizer always converges to a configuration $\mathbf{x} \in X^*$

Exercise: Explain feasibility in terms of traveling salesman problem

Note that, if the energy function is order preserving, then the energy will be minimal for configurations corresponding to an optimal solution (Figure 4.4). Furthermore, if the energy function is feasible, the network is guaranteed to converge to a feasible solution. Hence, feasibility and order-preservation of the energy function imply that the network will tend to find an optimal feasible solution for the given instance of the combinatorial optimization problem.

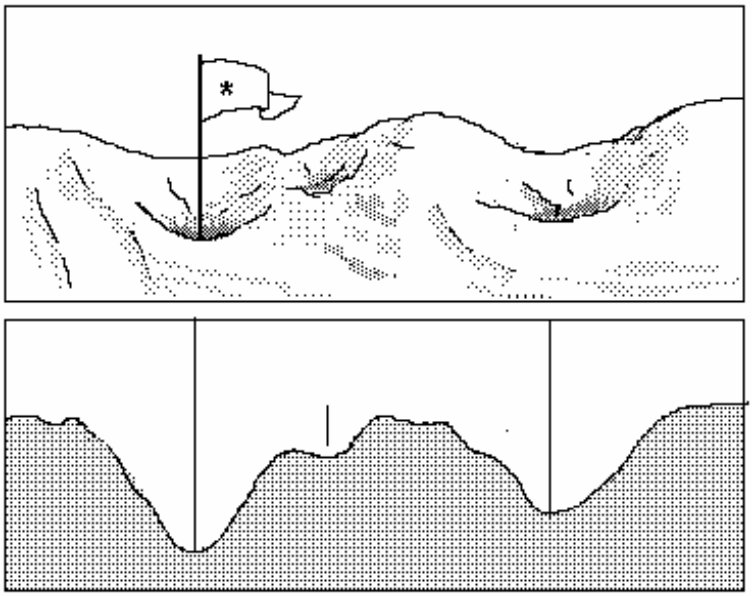

Figure 4.4: The goal of a *neural optimizer* is to converge to the global minimum of the energy function

Further notice that if $\{S^*\} \subset S'\text{-}m(X^*)$, where S^* is the minimum solution as defined by Eq. (4.1.1), in such a case, the neural network will never converge to a state

corresponding to the m inimum solution, but to a near m inimum one. The neural optimizers are usually designed such that $m(X^*)=S'$

4.3. Hopfield Network as Combinatorial Optimizer

In Section 4.2, we explained how neural ne tworks could be used for com binatorial optimization in general. In this section, we will consider the Hopfield network in particular. We will explain the design process, that is how the num ber of units and the weights of the connections are decided, th rough the NP-com plete problems provided in Section 4.1.

Consider the continuous valued asynchronous Hopfield model in which the outputs of the neurons are computed from its inputs using the sigmoidal relation. That is, for neuron i, it is in the form:

$$x_i = f(a_i) = \frac{1}{2}(1+\tanh(\kappa a_i)) \qquad (4.3.1)$$

where κ is the ***gain constant*** and a_i is the activation determined by the equation:

$$a_i = \sum_{j=1}^{N} w_{ji} x_j + \theta_i \qquad (4.3.2)$$

As in the case of associative m emory give n in Chapter 3, we will consider again the extreme case $\kappa \to \infty$. However, the output transfer functi on of the neurons here is shifted so that it takes values between 0 and 1, in spite of -1 and 1.

Still in this case, the energy function is [Hopfield 84, Hopfield and Tank 85]:

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ji} x_i x_j - \sum_{i=1}^{N} x_i \theta_i \qquad (4.3.3)$$

where x_i is the output of neuron i, w_{ji} is the connection weight from neuron j to neuron i, θ_i is the input bias to neuron i and N is the number of neurons in the network.

Notice that the energy function is bounded and has negative derivative when $x_i \in \{0,1\}$, so it is a Lyapunov function. Therefore, the energy is to be minimized by the Hopfield network's state transitions. Furthermore notice that $x=x^2$ whenever $x \in \{0,1\}$, hence the energy can be reorganized as:

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ji} x_i x_j \quad (4.3.4)$$

where $w_{ji}=2\theta_i$

Exercise: What happens to the restriction $w_{ii}=0$? Is it still necessary for binary state Hopfield Network?

Now our goal is to represent the vertex cover problem by a Hopfield network so that the cost of the problem will be minimized as the energy of the network decreases at each step.

A solution to the vertex covering problem has the following constraints:

1. Every edge in the graph must be adjacent to at least one of the vertices in the cover,
2. There should be as few vertices in the cover as possible.

The first constraint is necessary for the feasibility. The second one is, in fact, not a constraint but a statement for the minimization of cost function. The problem can be represented by a neural network in which each neuron corresponds to a vertex in the

graph. The outputs of neurons indicate whethe r the corresponding vertex is included in the cover or not. The case x_i=1 indicates that vertex i is in the cover while x_i=0 indicates it is not.

The energy function should be formed so that it satisfies the constraints that we discussed above. W e are thus dealing with a special case of a very general class of problem s, namely to find the m inimum of a function in the presence of constraints. The standard method of solution is to introduce the constraint via constants called Lagrange multipliers into the cost function, so the m inimum of th e cost function autom atically satisfies the constraints for the feasibility.

Let a 0-1 variable e_{ij} *be* assigned value 1 if there is an edge from vertex i to vertex j in the graph, and it is 0 otherwise. Below, the cost function to be m inimized is formulated as 0-1 programming (Ghanwani 94):

$$C(\mathbf{x}) = A(\sum_{i=1}^{N}\sum_{j=1}^{N} e_{ij} - 2\sum_{i=1}^{N}\sum_{j=1}^{N} x_i e_{ij} + \sum_{i=1}^{N}\sum_{j=1}^{N} x_i x_j e_{ij}) + B\sum_{i=1}^{N} x_i \qquad (4.3.5)$$

The term with coefficient A in Eq. (4.3.5) is zero when the requirem ent for a valid cover has been met. That is, all the edges in the graph are adjacent to at least one of the vertices in the cover. The term with coefficient B increases the energy by an am ount proportional to the num ber of vertices in the cover, em phasizing minimality. The constant part of the cost function can be dropped without affec ting the solution. Hence the cost function becomes

$$C(\mathbf{x}) = A(-2\sum_{i=1}^{n}\sum_{j=1}^{n} x_i e_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j e_{ij}) + B\sum_{i=1}^{n} x_i \quad (4.3.6)$$

By comparing the energy function given in Eq. (4.3.3) with the cost function in Eq. (4.3.6), we obtain:

$$w_{ij} = -2Ae_{ij} \quad (4.3.7)$$

and

$$\theta_i = 2A\sum_{i=1}^{n} e_{ij} - B \quad (4.3.8)$$

By this setting of the connection weights and thresholds, the energy function minimizes the cost function. In asynchronous network, the trajectory of states is highly dependent not only on the initial state of the network, but also on the order in which the processing elements are updated. Incorporation of randomness in the update order of the neurons usually yields to better results. Note that, although the order in which the neurons updated is decided at random, the outputs of neurons are still computed deterministically. The network is observed to converge almost instantly even for a large number of neurons. It is reported in [Ghanwani et al 94] that computing a set of solutions and then choosing the best among them dramatically improves the performance of the network, especially on smaller graphs.

Exercise: What is the relation between A and B for having a feasible solution at each local minima?

Now we will try to solve the traveling salesman problem using Hopfield network. TSP is a benchmark attempted by almost all methods developed for combinatorial optimization. This problem is also the one attempted by the Hopfield optimizer proposed in the classical paper [Hopfield 85].

TSP aims to find best order among the n cities to be visited. Expressed in a slightly different way, the visit order i in the tour should be determined for the each city α.

Introducing a square m atrix containing *nxn* binary elem ents, the solution can be represented in 0-1 program ming (Figure 4.5) . An entry having value "1" in the i^{th} position of row α indicates that the visit order of city α is *i*. The matrix corresponds to a feasible solution if and only if each row and colum n contains exactly one entry having value "1" [Muller 90].

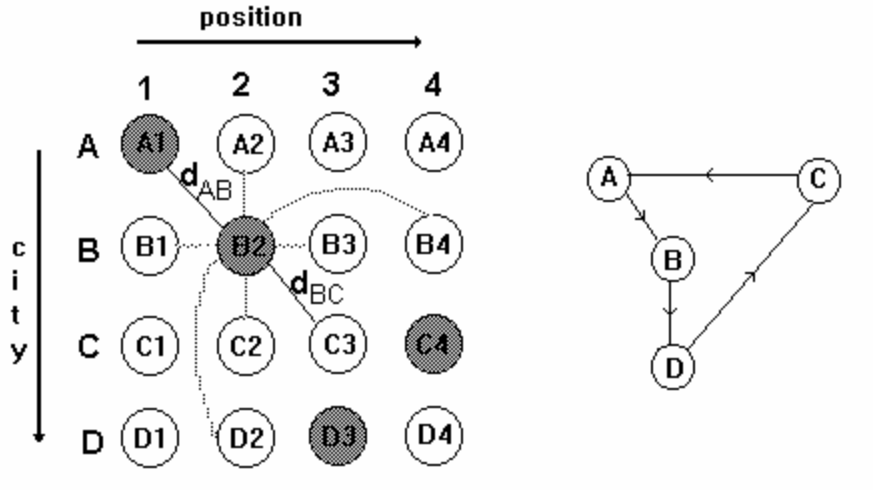

Figure 4.5. Representation of the tour of Figure 4.2.b by an nxn matrix, in which the rows corresponds to the cities while columns are indicating the order of visit

When a m atrix of neurons is used to repr esent the problem , the energy of the network becomes:

$$E = -\frac{1}{2}\sum_{\alpha=1}^{n}\sum_{i=1}^{n}\sum_{\beta=1}^{n}\sum_{j=1}^{n} w_{\alpha i,\beta j} x_{\alpha i} x_{\beta j} \qquad (4.3.9)$$

where $x_{\alpha i}$ is the output of neuron αi, $w\alpha i,\beta j$ is the connection strength between the units αi and βj while $w_{\alpha i,\alpha i}$ is reletad to the bias $\theta_{\alpha i}$ such that $w_{\alpha i,\alpha i}=2\theta_{\alpha i}$

For TSP, we have the following constraints:

1. Each city should be visited exactly once;

2. At each position of the travel route, there is exactly one city;
3. The length of the tour should be the minimum.

An appropriate choice for the cost function is [Abe 91]:

$$C(\mathbf{x}) = \frac{A}{2}\sum_{\alpha}^{n}(\sum_{i}^{n} x_{\alpha i} - 1)^2 + \frac{B}{2}\sum_{i}^{n}(\sum_{\alpha}^{n} x_{\alpha i} - 1)^2 + \frac{D}{2}\sum_{\alpha}^{n}\sum_{\beta\neq\alpha}^{n}\sum_{i}^{n} d_{\alpha\beta} x_{\alpha i}(x_{\beta,i+1} + x_{\beta,i-1}) \tag{4.3.10}$$

where A, B are the Lagrange multipliers used to combine the constraints in the cost function.

The cost function can be written as

$$C(\mathbf{x}) = \frac{A}{2}\sum_{\alpha}^{n}(\sum_{i}^{n}\sum_{j}^{n} x_{\alpha i}x_{\alpha j} - 2\sum_{i}^{n} x_{\alpha i} + 1) + \frac{B}{2}\sum_{i}^{n}(\sum_{\alpha}^{n}\sum_{\beta}^{n} x_{\alpha i}x_{\beta i} - 2\sum_{\alpha}^{n} x_{\alpha i} + 1) + \frac{D}{2}\sum_{\alpha}^{n}\sum_{\beta\neq\alpha}^{n}\sum_{i}^{n} d_{\alpha\beta} x_{\alpha i}(x_{\beta,i+1} + x_{\beta,i-1}) \tag{4.3.11}$$

In order to have the cost function of Eq. (4.3.11) in a form similar to the energy function given in Eq. (4.3.9) it can be reorganized as:

$$C(\mathbf{x}) = \frac{A}{2}\sum_{\alpha}^{n}\sum_{i}^{n}\sum_{\beta}^{n}\sum_{j}^{n}\delta_{\alpha\beta}x_{\alpha i}x_{\beta j} - A\sum_{\alpha}^{n}\sum_{i}^{n}x_{\alpha i} + \frac{A}{2}\sum_{\alpha}^{n} 1 + \frac{B}{2}\sum_{\alpha}^{n}\sum_{i}^{n}\sum_{\beta}^{n}\sum_{j}^{n}\delta_{ij}x_{\alpha i}x_{\beta j} - B\sum_{\alpha}^{n}\sum_{i}^{n} x_{\alpha i} + \frac{B}{2}\sum_{i}^{n} 1 + \frac{D}{2}\sum_{\alpha}^{n}\sum_{i}^{n}\sum_{\beta}^{n}\sum_{i}^{n}(1-\delta_{\alpha\beta})(\delta_{i,j+1} + \delta_{i,j-1})d_{\alpha\beta}x_{\alpha i}x_{\beta j} \tag{4.3.12}$$

Since the constant terms have no effect on the location of the minima of the cost function, they can be elim inated. Furtherm ore, $x_{\alpha i}=x_{\alpha i}^2$ whenever $x_{\alpha i}\in\{0,1\}$. Therefore, the cost function can be written as:

$$C(\mathbf{x})=\frac{A}{2}\sum_{\alpha}^{n}\sum_{i}^{n}\sum_{\beta}^{n}\sum_{j}^{n}\delta_{\alpha\beta}x_{\alpha i}x_{\beta j}-A\sum_{\alpha}^{n}\sum_{i}^{n}\sum_{\beta}^{n}\sum_{j}^{n}\delta_{\alpha\beta}\delta_{ij}x_{\alpha i}x_{\beta j}$$
$$+\frac{B}{2}\sum_{\alpha}^{n}\sum_{i}^{n}\sum_{\beta}^{n}\sum_{j}^{n}\delta_{ij}x_{\alpha i}x_{\beta j}-B\sum_{\alpha}^{n}\sum_{i}^{n}\sum_{\beta}^{n}\sum_{j}^{n}\delta_{\alpha\beta}\delta_{ij}x_{\alpha i}x_{\beta j} \qquad (4.3.13)$$
$$+\frac{D}{2}\sum_{\alpha}^{n}\sum_{i}^{n}\sum_{\beta}^{n}\sum_{j}^{n}(1-\delta_{\alpha\beta})(\delta_{i,j+1}+\delta_{i,j-1})d_{\alpha\beta}x_{\alpha i}x_{\beta j}$$

Compare the energy function given by Eq. (4.3 .9) and the cost function given in Eq. (4.3.13). Setting the weights as:

$$w_{\alpha i,\beta j}=-A\delta_{\alpha\beta}(1-2\delta_{ij})-B\delta_{ij}(1-2\delta_{\alpha\beta})-Dd_{\alpha\beta}(1-\delta_{\alpha\beta})(\delta_{j,i+1}+\delta_{j,i-1}) \quad (4.3.14)$$

makes the energy function order preserving. The constraints can be m ade equally weighted by setting $A=B$. In such a case the connection weights become:

$$w_{\alpha i,\beta j}=-A(\delta_{\alpha\beta}+\delta_{ij}-4\delta_{\alpha\beta}\delta_{ij})-Dd_{\alpha\beta}(1-\delta_{\alpha\beta})(\delta_{j,i+1}+\delta_{j,i-1}) \quad (4.3.15)$$

In order to have a feasible energy function, the inequality

$$A>2D\max_{\alpha,\beta}(d_{\alpha\beta}) \quad (4.3.16)$$

should be satisfied [Abe 91].

The method proposed in [Aiyer et al 90] al so optimizes the Hopfield and Tank approach. This m ethod is based on the eigenvalue anal ysis of the connection m atrix used in [Hopfield 85]. The improved weight matrix proposed in (Aiyer et al 1990) is:

$$w_{\alpha i,\beta j} = -A\delta_{\alpha\beta}(1-\delta_{ij}) - B\delta_{ij}(1-\delta_{\alpha\beta}) - 2A_1\delta_{\alpha\beta}\delta_{ij} - C + \frac{2(A_n - A - A_1)}{n^2} - Dd_{\alpha\beta}(\delta_{j,i+1} + \delta_{j,i-1}) \quad (4.3.17)$$