

ANN ee543

CHAPTER IV

Combinatorial Optimization by Neural Networks

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

Introduction

- 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, 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.
- 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 $NP=P$. However, their parallel nature and good performance in finding approximate solution make the neural optimizers interesting.

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

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.

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.1. *Combinatorial Optimization Problems*

- An **instance** of a combinatorial optimization problem can be formalized as a pair (S, g) .
- The **solution space**, denoted by 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, $g: S \rightarrow \mathbb{R}$.
- 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.

ANN ee543**CHAPTER IV : *Combinatorial Optimization by Neural Networks*****4.1. *Combinatorial Optimization Problems: P, NP***

- 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 time** and denoted by *P*.
- The set of polynomial time problems, 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.

ANN ee543**CHAPTER IV : *Combinatorial Optimization by Neural Networks*****4.1. *Combinatorial Optimization Problems: NP_complete***

- An important subclass of NP is the **NP_complete** problems. They are problems in NP-P 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.
- Empirically, the time it takes to solve an NP_complete problem tends to scale exponentially with the size of the problem.

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.1. *Combinatorial Optimization Problems: NP-complete*

- The probable situations for P and NP-complete problems are sketched in Figure 4.1.

(a) (b)

Figure 4.1 The class of NP problems a) assuming that $P \neq NP$
 b) if any $p \in NP_complete$ becomes $p \in P$, then implies $P=NP$

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.1. *Combinatorial Optimization Problems: Traveling Salesman*

- 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.
- 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.

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.1. Combinatorial Optimization Problems: *Traveling Salesman*

- In the TSP, the shortest closed path traversing each city under consideration exactly once is searched (Figure 4.2).
- For TSP, the number of cities determines the size of the problem.

Figure 4.2 The traveling salesman problem

- an instance with 4 cities
- the optimum solution
- a nonoptimum solution
- a non feasible solution having some unvisited cities

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.1. Combinatorial Optimization Problems: *Vertex Cover*

- 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.

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

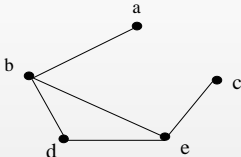
4.1. Combinatorial Optimization Problems: *Vertex Cover*

- The formal problem can be stated as follows:
 - Let $G=(V,E)$ be a graph where $V=\{v_1, v_2, \dots, v_N\}$ 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 $|C^*| \leq |C|$.

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.1. Combinatorial Optimization Problems: *Vertex Cover*



- Covers:

$C_1=(a,b,c,d,e)$
 $C_2=(a,b,c,d)$
 $C_3=(a,b,c,e)$
 $C_4=(a,b,d,e)$
 $C_5=(a,c,d,e)$
 $C_6=(b,c,d,e)$
 $C_7=(a,b,e)$
 $C_8=(a,d,e)$
 $C_9=(b,c,d)$
 $C_{10}=(b,c,e)$
 $C_{11}=(b,d,e)$
 $C_{12}=(b,e)$

- Minimal Cover: $C_{12}=(b,e)$

ANN ee543**CHAPTER IV : *Combinatorial Optimization by Neural Networks*****4.1. *Combinatorial Optimization Problems: Vertex Cover***

- If we have to solve an NP_complete problem, then a very long computation may be needed for an exact solution.
- The optimum solution of the vertex cover problem can be obtained by enumerating all the covers and then selecting the minimum one.
- However such an enumerative search for the exact optimum solution have a time complexity of $O(2^n)$, where n is the number of vertices in the graph.
- As vertex cover is an NP_complete problem, finding the exact minimum cover of G is not practical when the number of vertices is very large.
- Thus, in some cases, approximate algorithms are preferred.

ANN ee543**CHAPTER IV : *Combinatorial Optimization by Neural Networks*****4.2. *Mapping an Optimization Problems onto Neural Networks***

- Solving a combinatorial optimization problem aims to find the "best" or "optimal" solution among a finite or countably infinite number of alternative solutions.
- As an alternative to the conventional optimization methods, neural networks are used for the solution of combinatorial optimization problems.
- In general, a neural optimizer is a neural network whose neurons are affecting the problem solution.
- For instance neurons affecting the (city, position) pair of the tour can be used in the neural optimizer for solving the traveling salesman problem.
- If a neuron is "on", this implies that the corresponding city should be visited in the given position in the **approximately optimal** solution.

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.2. Mapping an Optimization Problems onto Neural Networks

- Then strongly inhibitory links are established between neurons, which represent incompatible elements of the solution; for example, a city should not be visited twice, and a position should not be occupied by two different cities.
- Furthermore, inhibitory links representing the cost are placed between neurons. For example, the intensity of inhibitory links can represent the distances between cities in the traveling salesman problem.
- Once the model is set up, it is allowed to relax dynamically to a steady-state which should be of "minimum energy" representing a quasi-minimal cost solution.
- Hopfield network, Boltzmann machine, mean field network, Gaussian machine and several other neural networks can be used as neural optimizers.
- The units in these networks tend to optimize a global function of the state space, by using only local information.
- Mean Field, Boltzmann and Gaussian machines are stochastic in nature and allow escaping from local optima.

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.2. Mapping an Optimization Problems onto Neural Networks

- An instance of a combinatorial optimization 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 satisfy the constraints of the problem;
- $g: S \rightarrow \mathbb{R}$, 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.

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.2. Mapping an Optimization Problems onto Neural Networks

- 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.

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.2. Mapping an OPs onto NNs: *Order Preservation*

- In The energy function E of a neural network that implements a minimization problem (S, S', g) is called **order-preserving** if for all $\mathbf{x}^k, \mathbf{x}^l \in X$ with $m(\mathbf{x}^k), m(\mathbf{x}^l) \in S'$, we have

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

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*4.2. Mapping an OPs onto NNs: *Feasibility*

- 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 all local minima of the energy function correspond to feasible solutions, that is

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

where

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

- 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 $\mathbf{x} \in X^*$

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*4.2. Mapping an OPs onto NNs: *Feasibility, Order Preservation*

- Note that, if the energy function is order preserving, then the energy will be minimal for configurations corresponding to an optimal solution (Figure 4.3).
- Furthermore, if the energy function is feasible, the network is guaranteed to converge to a feasible solution.

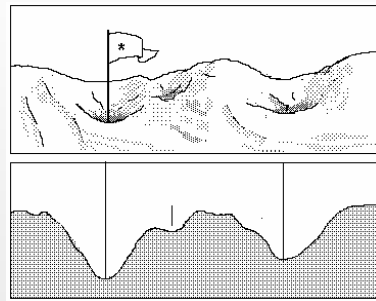


Figure 4.3: The goal of a *neural optimizer* is to converge to the global minimum of the energy function

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.2. Mapping an OPs onto NNs: *Feasibility, Order Preservation*

- 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.
- Further notice that if $\{S^*\} \subseteq S'-m(X^*)$, where S^* is the minimum solution. In such a case, the neural network will never converge to a state corresponding to the minimum solution, but to a near minimum one.
- The neural optimizers are usually designed such that $m(X^*)=S'$

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.3. Hopfield Network as Combinatorial Optimizer

- 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:

$$a_i = \sum_{j=1}^N w_{ji} x_j + \theta_i \quad (4.3.1)$$

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

$$x_i = f(a_i) = \frac{1}{2}(1 + \tanh(\kappa a_i)) \quad (4.3.2)$$

- As in the case of associative memory given in Chapter 3, we will consider again an extreme case. However, the output transfer function of the neurons here is shifted so that it takes values between 0 and 1, instead of -1 and 1.

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.3. Hopfield Network as Combinatorial Optimizer

- Still in the binary case, the energy function is:

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

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

- 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 formalized also as:

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

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.3. Hopfield NN as Combinatorial Optimizer : *Vertex cover*

- 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 should satisfy the followings
 - Every edge in the graph must be adjacent to at least one of the vertices in the cover (necessary for feasibility),
 - There should be as few vertices in the cover as possible (necessary for the minimization of cost function).

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.3. Hopfield NN as Combinatorial Optimizer : *Vertex cover*

- 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 whether 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.
- We are thus dealing with a special case of a very general class of problems, namely to find the minimum 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 minimum of the cost function automatically satisfies the constraints for the feasibility.

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.3. Hopfield NN as Combinatorial Optimizer : *Vertex cover*

- 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.
- Consider summation

$$\sum_{i=1}^N \sum_{j=1}^N e_{ij} - 2 \sum_{i=1}^N \sum_{j=1}^N x_i e_{ij} + \sum_{i=1}^N \sum_{j=1}^N x_i x_j e_{ij}$$
- The two terms related to $i=m, j=n$ and $i=n, j=m$ results in

$$e_{mn} + e_{nm} - 2x_m e_{mn} - 2x_n e_{nm} + x_m x_n e_{mn} + x_n x_m e_{nm}$$

ANN ee543

CHAPTER IV : Combinatorial Optimization by Neural Networks

4.3. Hopfield NN as Combinatorial Optimizer : Vertex cover

- Case $e_{mn}=e_{nm}=1$

$$e_{mn}+e_{nm}-2x_m e_{mn}-2x_n e_{nm}+x_m x_n e_{mn}+x_n x_m e_{nm}=2-2x_m-2x_n+x_m x_n+x_n x_m$$

x_m	x_n	$2-2x_m-2x_n+x_m x_n+x_n x_m$
0	0	2
0	1	0
1	0	0
1	1	0
- Case $e_{mn}=e_{nm}=0$

$$e_{mn}+e_{nm}-2x_m e_{mn}-2x_n e_{nm}+x_m x_n e_{mn}+x_n x_m e_{nm}=0$$

ANN ee543

CHAPTER IV : Combinatorial Optimization by Neural Networks

4.3. Hopfield NN as Combinatorial Optimizer : Vertex cover

- Below, the cost function to be minimized is formulated as 0-1 programming

$$C(\mathbf{x}) = A\left(\sum_{i=1}^N \sum_{j=1}^N e_{ij} - 2\sum_{i=1}^N \sum_{j=1}^N x_i e_{ij} + \sum_{i=1}^N \sum_{j=1}^N x_i x_j e_{ij}\right) + B\sum_{i=1}^N x_i \tag{4.3.5}$$
- The term with coefficient A in Eq. (4.3.5) goes to zero when the requirement 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 amount proportional to the number of vertices in the cover, emphasizing minimality.
- Dropping the constant part, equation (4.3.5) becomes

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

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.3. Hopfield NN as Combinatorial Optimizer : *Vertex cover*

- By comparing the energy function with the cost function,

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

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

we obtain:

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

and

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

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.3. Hopfield NN as Combinatorial Optimizer : *Vertex cover*

- 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.

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.3.Hopfield NN as Combinatorial Optimizer: *Traveling salesman*

- Now we will try to solve some more complicated traveling salesman problem. 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 .

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.3.Hopfield NN as Combinatorial Optimizer: *Traveling salesman*

- Introducing a square matrix containing $n \times n$ binary elements, the solution can be represented in 0-1 programming (Figure 4.4).
- 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 column contains exactly one entry having value "1".

Figure 4.4. Representation of the tour by an $n \times n$ matrix, in which the rows corresponds to the cities while columns are indicating the order of visit

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.3.Hopfield NN as Combinatorial Optimizer: *Traveling salesman*

- When a matrix of neurons is used to represent the problem, the energy of the network becomes:

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

or

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

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
 $\theta_{\alpha i} = w_{\alpha i, \alpha i} / 2$ is bias of neuron αi .

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.3.Hopfield NN as Combinatorial Optimizer: *Traveling salesman*

- For a TSP solution , the followings should be satisfied
 - Each city should be visited exactly once;
 - At each position of the travel route, there should be exactly one city;
 - The length of the tour should be the minimum.

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.3.Hopfield NN as Combinatorial Optimizer: *Traveling salesman*

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

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

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

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.3.Hopfield NN as Combinatorial Optimizer: *Traveling salesman*

- The cost function can be written as

$$C(\mathbf{x}) = \frac{A}{2} \sum_{\alpha} \left(\sum_i \sum_j x_{\alpha i} x_{\alpha j} - 2 \sum_i x_{\alpha i} + 1 \right) + \frac{B}{2} \sum_i \left(\sum_{\alpha} \sum_{\beta} x_{\alpha i} x_{\beta i} - 2 \sum_{\alpha} x_{\alpha i} + 1 \right) + \frac{D}{2} \sum_{\alpha} \sum_{\beta \neq \alpha} \sum_i d_{\alpha\beta} x_{\alpha i} (x_{\beta, i+1} + x_{\beta, i-1}) \quad (4.3.11)$$

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.3.Hopfield NN as Combinatorial Optimizer: *Traveling salesman*

- In order to have the cost function in the format of the energy function given it can be reorganized as:

$$\begin{aligned}
 C(\mathbf{x}) &= \frac{A}{2} \sum_{\alpha}^n \sum_i^n \sum_{\beta}^n \sum_j^n \delta_{\alpha\beta} x_{\alpha i} x_{\beta j} - A \sum_{\alpha}^n \sum_i^n x_{\alpha i} + \frac{A}{2} \sum_{\alpha}^n 1 \\
 &+ \frac{B}{2} \sum_{\alpha}^n \sum_i^n \sum_{\beta}^n \sum_j^n \delta_{ij} x_{\alpha i} x_{\beta j} - B \sum_{\alpha}^n \sum_i^n x_{\alpha i} + \frac{B}{2} \sum_i^n 1 \\
 &+ \frac{D}{2} \sum_{\alpha}^n \sum_i^n \sum_{\beta}^n \sum_j^n (1 - \delta_{\alpha\beta})(\delta_{i,j+1} + \delta_{i,j-1}) d_{\alpha\beta} x_{\alpha i} x_{\beta j}
 \end{aligned}$$

ANN ee543

CHAPTER IV : *Combinatorial Optimization by Neural Networks*

4.3.Hopfield NN as Combinatorial Optimizer: *Traveling salesman*

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

$$\begin{aligned}
 C(\mathbf{x}) &= \frac{A}{2} \sum_{\alpha}^n \sum_i^n \sum_{\beta}^n \sum_j^n \delta_{\alpha\beta} x_{\alpha i} x_{\beta j} - A \sum_{\alpha}^n \sum_i^n \sum_{\beta}^n \sum_j^n \delta_{\alpha\beta} \delta_{ij} x_{\alpha i} x_{\beta j} \\
 &+ \frac{B}{2} \sum_{\alpha}^n \sum_i^n \sum_{\beta}^n \sum_j^n \delta_{ij} x_{\alpha i} x_{\beta j} - B \sum_{\alpha}^n \sum_i^n \sum_{\beta}^n \sum_j^n \delta_{\alpha\beta} \delta_{ij} x_{\alpha i} x_{\beta j} \quad (4.3.13) \\
 &+ \frac{D}{2} \sum_{\alpha}^n \sum_i^n \sum_{\beta}^n \sum_j^n (1 - \delta_{\alpha\beta})(\delta_{i,j+1} + \delta_{i,j-1}) d_{\alpha\beta} x_{\alpha i} x_{\beta j}
 \end{aligned}$$

ANN ee543

CHAPTER IV : Combinatorial Optimization by Neural Networks

4.3.Hopfield NN as Combinatorial Optimizer: Traveling salesman

- Compare the energy function

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

and the cost function

$$\begin{aligned} C(\mathbf{x}) = & \frac{A}{2} \sum_{\alpha} \sum_i \sum_{\beta} \sum_j \delta_{\alpha\beta} x_{\alpha i} x_{\beta j} - A \sum_{\alpha} \sum_i \sum_{\beta} \sum_j \delta_{\alpha\beta} \delta_{ij} x_{\alpha i} x_{\beta j} \\ & + \frac{B}{2} \sum_{\alpha} \sum_i \sum_{\beta} \sum_j \delta_{ij} x_{\alpha i} x_{\beta j} - B \sum_{\alpha} \sum_i \sum_{\beta} \sum_j \delta_{\alpha\beta} \delta_{ij} x_{\alpha i} x_{\beta j} \\ & + \frac{D}{2} \sum_{\alpha} \sum_i \sum_{\beta} \sum_i (1 - \delta_{\alpha\beta}) (\delta_{i, j+1} + \delta_{i, j-1}) d_{\alpha\beta} x_{\alpha i} x_{\beta j} \end{aligned}$$

ANN ee543

CHAPTER IV : Combinatorial Optimization by Neural Networks

4.3.Hopfield NN as Combinatorial Optimizer: Traveling salesman

- Setting the weights as:

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

makes the energy function order preserving.

- The constraints can be made equally weighted by setting $A=B$. In such a case the connection weights become:

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

- In order to have a feasible energy function, the inequality

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

should be satisfied [Abe 91].