Neural Network Models in Combinatorial Optimization

Mujahid N. Syed and Panos M. Pardalos

Contents

M.N. Syed (✉)
Industrial and Systems Engineering, University of Florida, Gainesville, FL, USA
e-mail: smujahid@ufl.edu; snumujahid@gmail.com

P.M. Pardalos
Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL, USA
e-mail: pardalos@ufl.edu

P.M. Pardalos et al. (eds.), *Handbook of Combinatorial Optimization*,
DOI 10.1007/978-1-4419-7997-1_65,

Abstract

This chapter reviews the theory and application of artificial neural network (ANN) models with the intention of solving combinatorial optimization problems (COPs). Brief introductions to the theory of ANNs and to the classical models of ANNs applied to COPs are presented at the beginning of this chapter. Since the classical ANN models follow gradient-based search, they usually converge to a local optimal solution. To overcome this, several methods that extend the capability of ANNs to avoid the local minima have been reviewed in this chapter. Apart from that, not all the ANNs converge to a local minimum; thus, stability and/or convergence criteria of various ANNs have been addressed. The thin wafer that divides continuous and discrete optimization problems while applying ANNs to solve the COPs is highlighted. Applications of ANNs to solve the general optimization problems and to solve the discrete optimization problems have been surveyed. To conclude, issues regarding the performance behavior of the ANNs are discussed at the end of this chapter.

1 Introduction

Combinatorial optimization problems (COPs) are special problems in the field of operations research, where a real-valued mathematical function is optimized over a given domain (feasible set). Either some or all the variables of these problems are discrete in nature. The main criterion in proposing a solution method to solve the COPs is to provide a solution strategy better than the complete enumeration method. The conventional methods that are used to find the solution of a COP include branching, bounding, cutting planes, etc. Usually, COPs are NP-Hard in nature; the solution time of the conventional solution methods grows exponentially with the problem size. A curiosity to improve the solution time of finding an optimal solution of a COP directed the research toward unconventional methods. The primary unconventional methods for solving the COPs are the heuristics, and the most

prominent ones among them are the metaheuristics, such as simulated annealing, genetic and evolutionary algorithms, tabu search, and greedy randomized adaptive search procedure [15, 38, 44, 46, 53, 87, 94, 98, 115]. On the other hand, there were simultaneous research [32, 112] to use the analog circuits instead of the conventional computers for solving COPs. A method for solving the linear and the quadratic programs using resistors, diodes, transformers, current sources, and voltage sources (the basic building blocks of the analog circuit) was proposed in [32]. However, this method was impractical due to the unavailability of an ideal diode. Thus, this method remained in the theory until improved methods were proposed in [23, 62]. The first known analog circuit implementations which were capable of solving linear and quadratic programming problems were presented in [24, 131]. However, these methods were not ideal due to the use of negative resistors [131]. Later in 1985, Hopfield and Tank [121] proposed an analog circuit consisting of neurons,[1] called artificial neural networks (ANNs), to solve the COPs. This method caught the attention of many researchers, when Hopfield and Tank [121] solved some sufficiently large instances of the symmetric traveling salesman problem (TSP). The proposed method illustrated an alternate approach to solve the COPs, which is independent of the digital computers. After this illustration, many researchers were curious to utilize ANNs as a tool to solve various COPs.

1.1 Objective

The main objective of this chapter is to present different ideas of applying ANNs in solving the COPs. Furthermore, the issues related to the optimality and/or the stability analysis of various ANNs will be dealt. In addition to that, the implementation and the mapping of ANNs to solve the COPs will be illustrated by examples. Nevertheless, a brief history about the ANNs will be reviewed. In addition to that, usage of ANNs in solving the general optimization problems will be addressed.

1.2 Outline

In this chapter, ANN models which are applied in solving the COPs are described. This chapter is organized as follows: an introduction of ANNs in general and a review of the classical models in ANNs that are suitable for solving the COPs in particular are presented in Sect. 2. A systematic method, which illustrates the stability and convergence analysis of ANNs, is described in Sect. 3. Subsequently, the extensions and the improvements that can be incorporated to the classical ANN models are presented in Sect. 4. In addition to that, a methodology of mapping and

[1] A term similar to the smallest processing unit in the brain, in ANNs it consist of a small independent circuit with resistors, diodes, capacitors etc.

solving some of the general optimization problems is discussed in Sect. 5. Moreover, the applications of ANNs for solving various discrete programming problems are surveyed in Sect. 6. Nonetheless, in Sect. 7, the criticism of using ANNs to solve the COPs is presented. Finally, this chapter ends with the concluding remarks in Sect. 8.

2 Review

Human brain and its functionality has been the topic of research for many years, and until now, there has been no model that could aptly imitate the human brain. Curiosity of studying human brain lead to the development of ANNs. Henceforth, ANNs are the mathematical models mimicking the interactions among the neurons[2] within the human brain. Initially, ANNs were developed to provide a novel solution approach to solve the problems which do not have any algorithmic solution approach. The first model that depicted a working ANN used the concept of perceptron[3] [82, 101]. After the invention of perceptron, ANNs were the area of interest for many researchers for almost 10 years. However, this area of research was crestfallen due to the results shown by Minsky and Papert in [85]. Their results showed that the perceptron is incapable to model the simple logical operations. However, their work also indirectly suggested the usage of multilayer ANNs. After a decade, as a requital, this area of research became popular with Hopfield's feedback[4] ANN [56, 57]. A pioneer practical approach to solve the COPs was presented in [58, 121]. The mathematical analysis illustrated by Hopfield demonstrated that ANNs can be used as a gradient descent method in solving optimization problems. Moreover, some instances of traveling salesman problem (TSP) were mapped and solved using ANNs. This approach kindled enthusiasm among many researchers to provide an efficient solution methodology for solving NP-Hard problems. Since then, ANNs were modified and applied in numerous ways to solve the COPs.

2.1 Artificial Neural Networks (ANNs)

Neurons are the primary elements of any ANN. They posses some memory represented by their state. The state of a neuron is represented either by a single state or a dual state (internal and external state). The state of a neuron can take any real value between the interval $[0, 1]$.[5] Furthermore, the neurons interact with each other via links to share the available information. In general, the external state of the

[2] Biological neurons are the basic building blocks of the nervous system.

[3] A simple single-layered artificial neural network invented by Frank Rosenblatt.

[4] Feedback and feed-forward are two main classes of neural networks and will be explained in the following subsections.

[5] Some authors prefer to use the values between $[-1,1]$; however, both the definition are interchangeable and have the same convergence behavior. Hence, in this work [0,1], representation is followed.

neuron takes part in the outer interactions, whereas the internal state of the neuron takes part in the firing[6] decision. The intensity and sense of interactions between any two connecting neurons are represented by the weight (or synaptic[7] weight) of the links.

A single neuron itself is incapable to process any complex information. However, it is the collective behavior of the neurons that results in the intelligent information processing. Usually, neurons update their states based on the weighted cumulative effect of the states of the surrounding neurons. The update rule for the states is

$$\mathcal{I}_i = \sum_{j=1, j\neq i}^{n} w_{ij} x_i - U_i \tag{1}$$

$$x_i = \Psi(\mathcal{I}_i) \tag{2}$$

where x_i and $\mathcal{I}_i$ represent the external and internal states of an ith neuron, respectively, w_{ij} represents the weight between the ith neuron and the jth neuron, and U_i represents the threshold level of the ith neuron. Function $\Psi()$ is called the transfer function.[8] If the collective input to the ith neuron $\left(\sum_{j=1, j\neq i}^{n} w_{ij} x_i\right)$ is greater than the threshold (U_i), then the neuron will fire (value of x_i is set as 1). This is the basic mechanism of any ANN. With appropriate network structure and link weights, ANNs are capable of making any logical operations.

Example 1 Consider the following neural networks:

In Fig. 1, a simple neural network, which performs logical AND and logical OR operations, is depicted. The number over the links represents the weight of the link. The number inside the neurons represents the threshold value. Each neural network has two binary input signals and one binary output signal. Similarly, in Fig. 2, another elementary logical operator XOR is presented (where XOR is logical exclusive OR operator). From this figure, it can be seen that for a specific operation, neural network may not have a unique representation.

From Figs. 1 and 2, it can be seen that although a single neuron does not possess the computation ability, collectively, they can perform any logical operations very easily. Moreover, these are just the elementary form of logical operations that can be performed using simple neural networks. ANNs can be used for the complex logical operations as well. In order to use ANNs for complex logical operations, feedback method is used. Based on the feed mechanism, ANNs can be classified as feed-forward neural networks and Feedback neural networks as shown in Fig. 3.

[6]Whether to set the value of the corresponding external state to 0 or 1

[7]The term synaptic is related to the nervous system and is used in ANNs to indicate the weight between any two neurons.

[8]This function may be continuous or discontinuous based on the type of the neuron. That is, for a discrete case, this function is similar to a signum function, whereas for continuous case, this function is similar to a sigmoidal function.

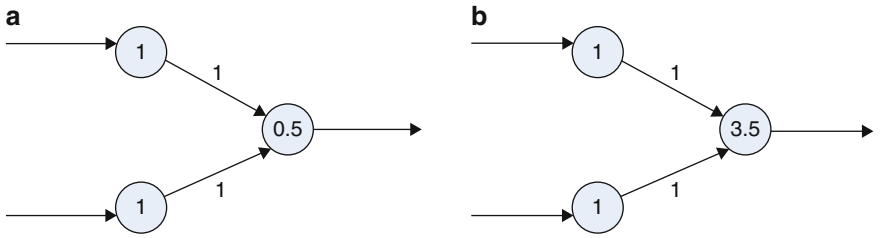

Fig. 1 Example:1 simple logical operations in ANNs. (**a**) OR. (**b**) AND

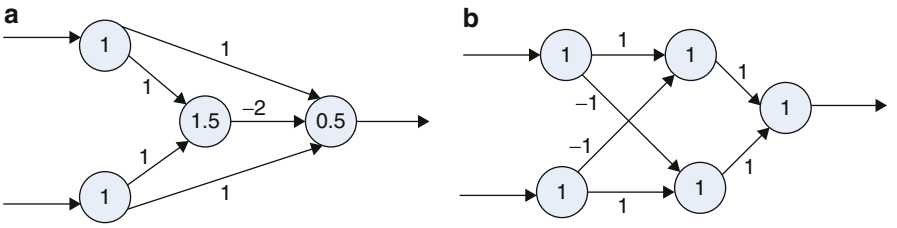

Fig. 2 Example:2 simple logical operations in ANNs. (**a**) XOR. (**b**) XOR

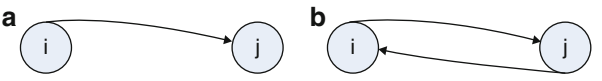

Fig. 3 Major categories of ANNs. (**a**) Feed-forward ANN. (**b**) Feedback ANN

In the feed-forward neural networks, information is processed only in one direction. Whereas in the case of feedback neural networks, information is processed in both the directions. From the literature, typically, there are three main types of applications of ANNs in information processing. They are categorized as:

Type 1: Pattern Recognition

- This is the most explored application of the ANNs [14, 19, 88, 89]. In pattern recognition, two sets of data, namely, *training data*, and *testing data* are given. Generally, the data sets consists of input parameters which are believed to be the cause of the output features. The aim of pattern recognition is to find a relation that maps the input parameters with the output features. This is obtained by learning, where the weights of the neural network are updated by error backpropagation. When the error between the generated features and the required output features falls below some acceptable limit, the weights are frozen or set to the current value. These weight are used to validate the performance of the obtained neural network using the testing data. This approach is used in function approximation, data prediction, curve fitting, etc.

Type 2: Associative Memory

- Advancement in the theory of ANNs from feed-forward neural networks to feedback neural networks led to this application [18, 84, 93]. In associative memory application, a neural network is set to the known target memory pattern. Later, if an incomplete or partly erroneous pattern is presented to the trained

ANN, then the ANN is capable to generate or repair the input pattern to the most resembling target pattern.

This approach is used in handwriting recognition, image processing, etc.

Type 3: Optimization

- Further extensions of associative memory approach led to the application of ANNs in solving the optimization problems. This application is fundamentally different from the pattern recognition application. In this approach, weights of the neural network are fixed, and the inputs are randomly (synchronously or asynchronously) updated such that the total energy of the network is minimized. This minimization is achieved if the network is stable. Based on the different optimality conditions, the system may converge to the local (or global) optimal solution. ANNs have been applied not only for solving the COPs [3, 43, 64, 97, 110, 111] but also for solving the optimization problems in general [39, 86, 90, 104]

In the following subsection, an example implementation of ANNs for solving a general linear programming (LP) problem is presented.

2.2 Example: Implementation

One of the conventional methods of solving any COP is to relax the problem into a series of continuous problems, and solve them until the desired discrete solution is obtained [47]. Although simplex is the most widely used method to solve the linear relaxation of COPs, it has been shown in the literature that other methods like dual-simplex, barrier methods, and interior point methods perform better than simplex on certain occasions [66, 99, 133]. Apart from that, the convergence time of simplex method is exponential in time [69], yet in practice, it performs better than polynomial time interior point method. Thus, finding a polynomial time algorithm (polynomial both in theory and in practice) for solving linear programming problems is still an attractive area of research. Therefore, it becomes an attractive choice to study the method of solving an LP problem using an ANN. Apart from that, the purpose of selecting an LP model to show the implementation of ANNs is due to the simple nature of LP models. Thus, a reader familiar with the theory of LP will find it easy to understand the analog circuit of the LP problem.

Consider a standard form of a linear programming problem described as

$$\begin{aligned} &\text{minimize :} \\ &\qquad \mathbf{c}^T\mathbf{x} \\ &\text{subject to :} \\ &\qquad A\mathbf{x} = \mathbf{b} \\ &\qquad 0 \le x_i \le x_{\max} \quad \forall i = 1, 2, \ldots, n \end{aligned} \tag{3}$$

where $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$, and $\mathbf{x} \in \mathbb{R}^n$. In order to apply an ANN to solve this problem, the above stated problem has to be transformed into an unconstrained optimization problem using either a penalty function or a Lagrange function method. For the sake of simple illustration, a penalty function approach will be presented. Let $\mathbf{x}(t)$ represent the instantaneous or dynamic value of the solution vector at time t. Let $E(\mathbf{x}(t), t)$ represent the modified (penalized) objective function, which is given as

$$E(\mathbf{x}(t), t) = \frac{C_1}{2}(A\mathbf{x}(t) - \mathbf{b})^T(A\mathbf{x}(t) - \mathbf{b}) + C_2\mathbf{c}^T(\overline{\mathbf{x}} + \mathbf{x}(t)e^{-C_3 t}) \tag{4}$$

where C_1, C_2, and C_3 represent the positive scaling parameters of the system. Let the system dynamics be defined as

$$\nabla_t \mathcal{I} = -\nabla_{\mathbf{x}(t)} E(\mathbf{x}(t), t) \tag{5}$$

where $\mathcal{I}$ represents the internal energy of the neuron. Based on the system dynamics defined in Eq. (5), the system can be described as

$$\nabla_t \mathcal{I} = -C_1 A^T A\mathbf{x}(t) + C_1 A^T \mathbf{b} - C_2\mathbf{c}\, e^{-C_3 t} \tag{6}$$

and

$$x_i(t) = \frac{x_{\max}}{1 + e^{(-C_4 \mathcal{I}(t))}} \tag{7}$$

where C_4 is a positive scaling parameter,

$$\nabla_t \mathcal{I} = \left(\frac{d\mathcal{I}_1}{dt}, \frac{d\mathcal{I}_2}{dt}, \ldots, \frac{d\mathcal{I}_n}{dt}\right)^T, \quad \mathcal{I} \in \mathbb{R}^n$$

and

$$\nabla_{\mathbf{x}(t)} E(\mathbf{x}(t), t) = \left(\frac{\partial E(\mathbf{x}(t), t)}{\partial x_1}, \frac{\partial E(\mathbf{x}(t), t)}{\partial x_2}, \ldots, \frac{\partial E(\mathbf{x}(t), t)}{\partial x_n}\right)^T$$

Equations (6) and (7) are the main equations which are used to design the ANN. The first thing in designing the neural network is to know the number of neurons. Since each variable is represented by a neuron, there will be altogether n neurons required for the construction of ANN. Let $\mathcal{I}_i$ represent the internal state of an ith neuron and x_i represent the external state of an ith neuron. At any given time, both the states of an ith neuron can be represented by $[\mathcal{I}_i(t), x_i(t)]$. The next step will be to connect the network using links. These links will have resistors, which represent the weights (synaptic weights) of the link. Based on the coefficients in Eq. (6), the weights are taken as the corresponding coefficients of the first-order term. The constant term in Eq. (6) is taken as the input bias to the neurons. From the Eq. (6), the following weight matrix and bias vector is obtained:

$$W = -C_1 A^T A \quad \text{and} \quad \theta(t) = C_1 A^T \mathbf{b} - C_2 c\, e^{-C_3 t} \tag{8}$$

or

$$w_{ij} = -C_1 \sum_{k=1}^{m} a_{ki} a_{kj} \quad \text{and} \quad \theta_i(t) = C_1 \sum k = 1^m a_{ki} b_k - C_2 c_i \; e^{-C_3 t} \tag{9}$$

where a_{ij} is the ith row jth column element of A and w_{ij} is the ith row jth column element of W.

Now, in order to set the appropriate weights, we use resistors. The absolute value of synaptic weight w_{ij} (between ith and jth neuron) is obtained as the ratio of the resistors R_f and R_{ij}, that is,

$$|w_{ij}| = \frac{R_f}{R_{ij}} \tag{10}$$

The next step is to assign input bias to each neuron. This can be done using the voltage E_i and the resistor R_i, given as

$$\theta_i = \frac{R_f E_i}{R_i} \tag{11}$$

The other part of the input bias, that is, $C_2 c_i e^{-C_3 t}$ is obtained by providing a discharging loop, with an additional set of capacitor C_d, and resistor R_d. The configuration is shown in Fig. 4. The values of R_d and R_f are taken arbitrarily such that $R_d \ll R_f$. The initial value of C_d is assigned as $-C_2 \; c_i \; \forall \; i$. Once these values are set, the other values are given as

$$R_{ij} = \frac{R_f}{w_{ij}} \tag{12}$$

$$C_3 \approx 1/(R_d C_d) \tag{13}$$

The positive (or negative) value of weight is obtained by using the x_ith terminal (or $-x_i$th terminal) of the neuron. For the sake of simplicity, the neuron in Fig. 4 can be represented as depicted in Fig. 5. Once the structure of a single neuron is formed, it is connected to the other neurons to form the ANN which can be represented as in Fig. 6. As the time proceeds, the neurons update their internal and external energies based on the update rules and based on the dynamics of selecting the neurons for the update. If the network is stable and the dynamics is appropriate, the network will converge to a limit point(local minimum), which will give the optimal solution of the LP [39]. Applying ANNs to solve the LP problems has been studied and analyzed in [25–28, 35, 67, 100, 136].

The objective of depicting the method of developing analog circuit for solving LP was to illustrate the reader various complexities involved in the deployment of an analog ANN for solving any COP. In the following subsections, some of the classical models of the ANNs that are applied in solving COPs will be described.

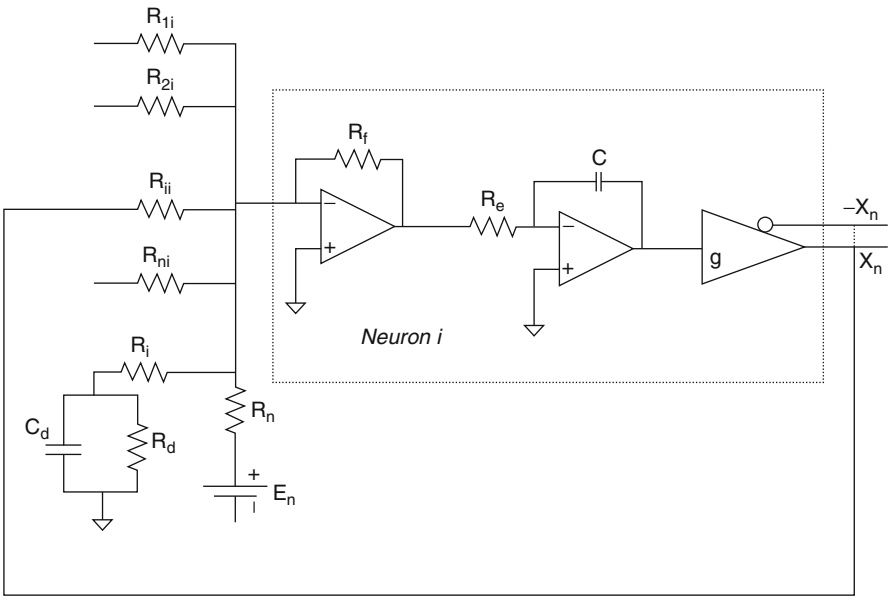

Fig. 4 Circuit representation of a neuron

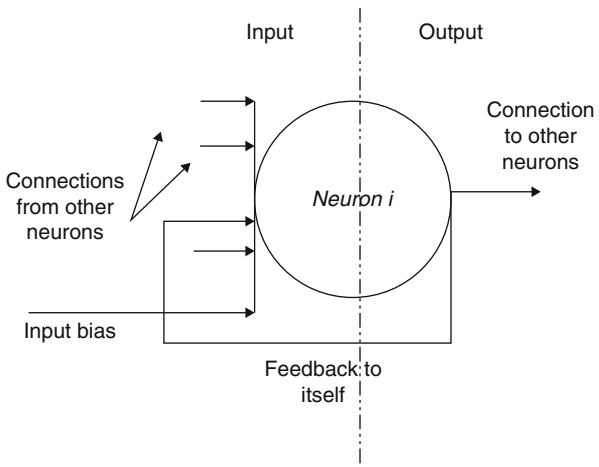

Fig. 5 Block representation of a neuron

These are the basic models proposed in 1980s–1990s as an alternate analog method to solve COPs. These models can be broadly classified as:

- Gradient-based methods
  - Hopfield and Tank's discrete and continuous models
- Self-organizing methods
  - Durbin-Willshaw's and Kohonen's models
- Projective gradient methods
  - Lagrange-based models

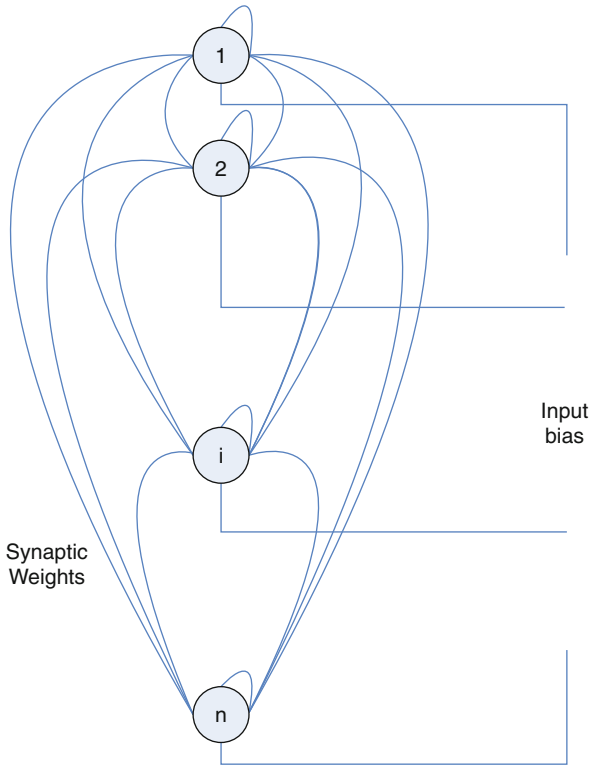

Fig. 6 Interconnected neurons

2.3 Hopfield and Tank (H-T) Models

In 1982, Hopfield [56] proposed a novel feedback type neural network, which is capable of performing computational tasks. Although feedback type models were proposed earlier by Anderson and Kohonen [10, 71], however, Hopfield [57] provided a complete mathematical analysis of feedback type neural networks. Thus, these feedback type neural networks were named as Hopfield networks. There are two different models proposed by Hopfield and Tank [58] for solving COPs. They are described below:

(a) *H-T's Discrete Model*

The discrete model consists of n interconnected neurons. The strength of the connection between the ith neuron and the jth neuron is represented by the weight w_{ij}. A presence of connection between any two neurons is indicated by a nonzero weight. Weights maybe positive or negative, representing the sense of connection (excitatory or inhibitory). In this model, each neuron has been assigned with the two states (external and internal states, like that in McCullough and Pitts [82] model) representing the embedded memory at each neuron. Moreover, the internal state of ith neuron $\mathcal{I}_i$ is continuous valued, whereas the external state x_i is binary valued. The internal and external state of a neuron are related as

$$\mathcal{I}_i(t+1) = \sum_{j=1}^{n} w_{ij} x_j(t) + \theta_i \tag{14}$$

$$x_i(t+1) = \Psi_d(\mathcal{I}_i) \begin{cases} 1 & \text{if } \mathcal{I}_i > U_i \\ 0 & \text{if } \mathcal{I}_i \leq U_i \end{cases} \tag{15}$$

where θ_i is a constant representing an external input bias to the ith neuron. $\Psi_d()$ represents a transfer function, which assign binary values to x_i. U_i represents the threshold value of the ith neuron. If the aggregate sum of the internal energies supplied to the ith neuron is greater than U_i, then this neuron will "fire," that is, x_i will be set to 1. On the other hand, if the aggregate sum of the internal energies is less than the threshold, the neuron will be in a dormant or "not-firing" state. For solving the COP, w_{ij}'s are calculated based on the objective function and the constraints of a given COP. The only decision variables of this model are x_i and $\mathcal{I}_i$. Unless or otherwise specified, the values of U_i's are taken as 0.

Initially, a random value is assigned to each external state, and the neurons update their states based on the updating rules. Usually, the neurons are selected randomly for the update (asynchronous updating). This updating rule is proven to converge to one of the stable states [83], which minimizes the energy function, provided the energy function is given by Eq. (16):

$$E_d = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \mathrm{w}_{ij} x_i x_j - \sum_{i=1}^{n} \theta_i x_i \tag{16}$$

Since one of the states is binary and the update sequence of neurons is random, this model is known as a discrete model with stochastic updates. With the above details, Algorithm 1 summarizes the mechanism of this model.

This model can be used to compute the local minimum of any quadratic function. Hopfield and Tank [58, 121] showed that the way energy function is constructed will always result in decrease of energy and the algorithm leads to one of the stable states. The algorithm proceeds as the gradient descent method, converging to a local minimum.

The main step in solving a COP using H-T's discrete model is to convert the constrained COP into an unconstrained COP using a penalty function method (or Lagrange method). Once the unconstrained penalty objective function is obtained, it is compared with the energy function given in Eq. (16).

(b) *H-T's Continuous Model*

In the subsequent research [57], H-T proposed another model in which both the states of the neuron are continuous. In addition, in this model, two additional properties have been assigned to the neurons, namely, the input capacitance C_i and the transmembrane resistance R_i. Moreover, a finite impedance R_{ij} between the

Algorithm 1 Hopfield Discrete Model

Randomly assign initial values to x_i
termination = false
while (termination) **do**
  Select one neuron randomly
  Update the states $\mathcal{I}_i$ and x_i:
  $\mathcal{I}_i(t+1) = \sum_{j=1}^{n} w_{ij} x_j(t) + \theta_i$
  if $\mathcal{I}_i > 0$ **then**
    $x_i(t+1) = \Psi_d(\mathcal{I}_i) = 1$
  else
    $x_i(t+1) = \Psi_d(\mathcal{I}_i) = 0$
  end if
  Calculate the energy of the system E_d:
  $E_d = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} x_i x_j - \sum_{i=1}^{n} \theta_i x_i$
  if *termination criteria is met* **then**
    termination = true
  end if
end while

ith neuron and the output x_j from jth neuron is assigned on the link ij. The main difference between the continuous model and the discrete model is in the continuous model, the external states of the neurons are continuous valued between 0 and 1. The equations of motion for this model are given as

$$\frac{d\mathcal{I}_i}{dt} = \sum_{j=1}^{n} w_{ij} x_j(t) - \frac{\mathcal{I}_i}{\tau} + \theta_i \tag{17}$$

$$x_i = \Psi_c(\mathcal{I}_i) \tag{18}$$

where $\tau = R_i C_i$ and usually, τ is assigned a value of 1, if the time step of any discrete time simulation is less than unity. A widely used transfer function is continuous sigmoidal function which is of the form, $\Psi_c(\mathcal{I}_i) = \frac{1}{2}(1 + \tanh(\frac{\mathcal{I}_i}{T}))$. The slope of the transfer function is controlled by parameter T. Similar to the previous model, this model will converge to a stable state which will minimize the energy function, given by Eq. (19):

$$E_c = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} x_i x_j - \sum_{i=1}^{n} \theta_i x_i + \int_0^{x_i} \Psi_c^{-1}(a)\, da \tag{19}$$

The only decision variables in this model are x_i and $\mathcal{I}_i$. Algorithm 2 describes the mechanism of this model. From this illustration, it can be seen that a continuous ANN can be applied to solve discrete optimization problems, just by tuning the parameters of the transfer function. In specific, if the value of the slope, T, in the transfer function is set to a very high value, then this model will approximately behave as the discrete model.

Algorithm 2 Hopfield Continuous Model

Randomly assign initial states x_i
termination = false
while (termination) **do**
  Select randomly one neuron
  Update the states $\mathcal{I}_i$ and x_i:
  $\frac{d\mathcal{I}_i}{dt} = \sum_{j=1}^{n} w_{ij} x_j(t) - \frac{\mathcal{I}_i}{\tau} + \theta_i$
  $x_i(t+1) = \Psi_c(\mathcal{I}_i) = \frac{1}{2}(1 + tanh(\frac{\mathcal{I}_i}{T}))$
  Calculate the energy of the system E_d:
  $E_c = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} x_i x_j - \sum_{i=1}^{n} \theta_i x_i + \int_0^{x_i} \Psi_c^{-1}(a)\, da$
  if *termination criteria is met* **then**
    termination = true
  end if
end while

The first attempt to solve the COP using ANN was demonstrated by Hopfield and Tank [58] in 1985. They illustrated the usage of ANN by solving sufficiently large instances of the famous traveling salesman problem (TSP). After this illustration by Hopfield and Tank, numerous approaches and extensions were proposed to solve various COPs using ANNs. Most of the proposed extensions were in the direction of improving H-T's ANN model. In the following subsection, the alternative approaches which are significantly different from H-T's ANN model are presented.

2.4 Durbin-Willshaw (D-W)'s Model

In 1987, Durbin and Willshaw [34] proposed a fundamentally different approach to the H-T's approach for solving the geometric COPs. The proposed method was named as elastic nets and often referred as deformable templates. The origin of this method is the seminal work of Willshaw and Malsburg [130]. In this model, the neurons have no memory or states. Moreover, the network is not arbitrarily connected based on the values of the weights. Instead, it is connected in the form of a ring (elastic ring), where the neurons are placed on the circumference of the ring. Since this model is used to solve the geometric COPs, we will describe the model with respect to TSP.

In this model, there are M neurons on the circumference of the ring, where M is greater than N, the number of cities. The first step is to find the center of gravity of the network. This point is taken as the center for the ring, and iteratively neurons on the rings are pulled apart. In every iteration, neurons are pulled in such a way that the distance between the neurons and the closest city is minimized. However, there is another force from the ring that tries to keep the length of the ring as small as possible. Thus, in the end, when each city has been close enough to a corresponding neuron, a Hamiltonian tour is obtained. The equation describing the movement of jth neuron is given as

$$\Delta y_j = \alpha \sum_{i=1}^{N} w_{ij}(a_i - y_i) + \beta K(y_{j+1} + y_{j-1} - 2y_j) \tag{20}$$

where a_i is the coordinate of the ith vertex of the TSP; α, β, and K are the scaling parameters:

$$w_{ij} = \frac{\phi(d_{a_i y_j}, K)}{\sum_{k=1}^{M} \phi(d_{a_i y_k}, K)} \quad \forall i, j \tag{21}$$

$d_{a_i y_j}$ is the Euclidean distance between the ith city and the jth neuron; and

$$\phi(d_{a_i y_j}, K) = e^{-{d_{a_i y_j}}^2/2K^2} \tag{22}$$

α represents the scaling factor that drives the neurons toward the cities whereas β represents the scaling factor for the force that keeps the neighboring neurons on the ring closer. Parameter K is gradually decreased, so that the neurons get closer to the cities. This parameter acts like temperature in simulated annealing and requires a cooling schedule. The energy of this system is given by Eq. (23):

$$E_{dw} = -\alpha K \sum_{i=1}^{N} ln \sum_{j=1}^{M} \phi(d_{a_i y_j}, K) + \frac{\beta}{2} \sum_{j=1}^{M} d^2_{y_j y_{j+1}} \tag{23}$$

The mechanism of this model can be explained by Algorithm 3 .

2.5 Kohonen Model

Another fundamentally different approach to the H-T's approach was proposed by Kohonen [72] for solving geometric COPs. This model utilizes self-organizing maps (SOMs) [70, 71] which fall under the category of the competitive neural networks. In this model, there are two layers of neurons (input and output). The input layer of the neurons are fully connected to the output layer of the neurons. In general, the neurons at the inner layer represent the input from a high-dimensional space.

Algorithm 3 Durbin-Willshaw model

Find the center of gravity of the system, and assign it as the center of the circle with M equispaced ring points.
termination = false
while (termination) **do**
  Update the coordinates y_i of ring points using equation Eq. (20)
  if $(\min_{1,\ldots,M}\{d_{a_i,y_i}\} \leq \varepsilon)$ or $(K < K_{min})$ **then**
    termination = true
  end if
  $K = \alpha K \quad \alpha \in (0, 1)$
end while

However, the neurons at the output layer represent output to a low-dimensional space. Therefore, the whole model act as a mapping from a higher dimensional space to a lower dimensional space. The property of self-organizing maps is to map the neurons close in the input space to the neurons that are close in the output space. The output layer of the neurons is arranged according to a topological structure, which is based on the geometry of the problem under consideration. For example: for the TSP, the output layer is arranged in a ring structure. Let $a^i = (a_1^i, a_2^i, \ldots, a_n^i)^T \quad \forall i = 1, \ldots, N$ be the coordinates of ith city to be visited. Let $w^j = (w_{1j}, w_{2j}, \ldots, w_{Nj})^T, \quad \forall j = 1, \ldots, M, \ M \geq N$ represent the weight vector of the jth output neuron. Unlike the previous models, in this model, the weights are the decision variables. They are changed over the time using the following equation:

$$w^j = w^j + \mu f_{j,j*}(a^i - w^j), \quad \forall j = 1, 2 \ldots, M, \quad 0 < \mu < 1 \tag{24}$$

where μ is called learning rate, j^* is called winning neuron, defined as $j^* = \text{argmin}_{\forall j}\{d_{a^i w_j}\}$ for a given city i, and the function $f : (j, j^*) \mapsto [0, 1]$ is a decreasing function and represents the lateral distance between output units j and j^*. This function acts like a weight of attraction between two points. Typically, the function is modified as the algorithm proceed to gradually reduce the magnitude of the weight of attraction. This method can be described by Algorithm 4 .

Thus, this iterative approach at the end produces a Hamiltonian tour. Since both Durbin-Willshaw's and Kohonen's models are for geometrical problems, these models have not been widely used in the application of ANNs in solving COPs.

2.6 Lagrangian Model

Another approach which is based on the Lagrange programming is presented by Zhang and Constantinides [139]. The main difference in this method when compared to the Hopfield and Tank [58] approach is that this method is not similar to the gradient descent method. In this method, the constraints and objective function are not coupled using the penalty function. Instead, two types of neurons are used

Algorithm 4 Kohonen Model

Randomly assign initial weights w_j, set i = 0.
termination = false
while (termination) **do**
  Select neuron i, $i = (i + 1) mod (N + 1)$
  Calculate $d_j = d_{a^i w_j} \quad \forall j = 1, \ldots, M$
  Assign $j^* = argmin_j\{d_j\}$ and $d_j = d_{j*}$
  Update the weights as: $w^j = w^j + \mu f(j, j*)(a^i - w^j), \quad \forall j = 1, \ldots, M, \ I \ \& \ 0 < \mu < 1$
  if $min_j\{d_{a^i w^j}\} \leq \varepsilon, \forall i = 1, \ldots, N$ **then**
    termination = true
  end if
end while

(variable type, x, and Lagrangian type, λ). One type of neurons (Lagrangian) looks for a feasible solution, and the other type of neurons (variable) looks for an optimal solution.

The main advantage of this model over the previous model is that the objective function can be free from the Lyapunov function criterion and can be different from a quadratic function. That is, any type of function can be optimized, unlike the quadratic function requirements of H-T's model [58]. For example, consider any general optimization problem defined as

$$\begin{aligned} &\text{minimize :} \\ &\qquad f(\mathbf{x}) \\ &\text{subject to :} \\ &\qquad h_i(\mathbf{x}) = 0 \quad \forall i = 1, \ldots, m \\ &\qquad \mathbf{x} \in \mathbb{R}^n \end{aligned} \tag{25}$$

where $f(\mathbf{x})$, $h_i(\mathbf{x}) : \mathbb{R}^n \mapsto \mathbb{R}$ are any continuous and twice differentiable functions of $\mathbf{x}$. The Lagrange for the problem Eq. (25) is written as

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T h(\mathbf{x}) \tag{26}$$

The equations of motion that describe this model are

$$\nabla_t \mathbf{x} = -\nabla_x L(\mathbf{x}, \boldsymbol{\lambda}) \tag{27}$$

$$\nabla_t \boldsymbol{\lambda} = -\nabla_\lambda L(\mathbf{x}, \boldsymbol{\lambda}) \tag{28}$$

where

$$\nabla_t \mathbf{x} = \left(\frac{dx_1}{dt}, \frac{dx_2}{dt}, \ldots, \frac{dx_n}{dt} \right)^T, \quad x \in \mathbb{R}^n$$

$$\nabla_t \boldsymbol{\lambda} = \left(\frac{d\lambda_1}{dt}, \frac{d\lambda_2}{dt}, \ldots, \frac{d\lambda_m}{dt} \right)^T, \quad \lambda \in \mathbb{R}^m$$

$$\nabla_x L(\mathbf{x}, \boldsymbol{\lambda}) = \left(\frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial x_1}, \frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial x_2}, \ldots, \frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial x_n} \right)^T$$

and

$$\nabla_\lambda L(\mathbf{x}, \boldsymbol{\lambda}) = \left(\frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial \lambda_1}, \frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial \lambda_2}, \ldots, \frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial \lambda_m} \right)^T$$

The above model is iterative as described in Algorithm 5 , similar to that of H-T's continuous model. Both the types of neuron are the decision variables.

The mathematical properties and proofs related to convergence of ANN models will be discussed in Sect. 3 of this chapter.

Algorithm 5 Lagrangian Model

Randomly assign initial states x_i and λ_i
termination = false
while (termination) **do**
  Select randomly one neuron (either variable or lagrangian)
  if *ordinary variable is selected* **then**
    Update the states x_i as:
    $\nabla_t \mathbf{x} = -\nabla_x L(\mathbf{x}, \boldsymbol{\lambda})$
  else
    Update the states λ_i as:
    $\nabla_t \boldsymbol{\lambda} = -\nabla_\lambda L(\mathbf{x}, \boldsymbol{\lambda})$
  end if
  Calculate the energy of the system $L(\mathbf{x}, \boldsymbol{\lambda})$:
  $L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T h(\mathbf{x})$
  if *termination criteria is met* **then**
    termination = true
  end if
end while

2.7 General Methodology

In the previous subsections, a brief picture of different models that can be used to solve a COP is presented with their algorithmic structure. In the following part of this subsection, general guidelines for mapping COPs onto the ANNs models will be presented.

Guidelines for H-T's Model (Discrete and Continuous)

- The output state of the neurons represents the decision variables of the COP.
- Combine the objective function and constraints using the penalty functions, such that the overall modified objective function is quadratic in nature, and it is called as the energy function of the system.
- Obtain the weights and the bias for each neuron. This is the main step that maps the COP onto ANN. This can be done by any one of the following methods:
  - Compare the coefficients of terms in energy function with the coefficients in E_d or E_c functions. This comparison will determine the weights on the links and will determine the input bias of a neuron.
  - Compare the coefficients of degree 1 in the differential update rule to the weights. Similarly, compare the constant term of differential update rule to the bias.
- Assign a random initial state to all the neurons, and proceed as described in Algorithm 1 or 2 .

Note: Only models with linear and/or quadratic objective function can be solved using the classical H-T's model. For solving the general COPs, various extensions are proposed, like the Lagrange function method.

Guidelines for Durbin-Willshaw's Model

- The decision variable of the COP is represented by the Euclidean position of the neurons.
- Find the center of gravity of the given geometry and mark it as the ring's center.
- Initially, all the neurons are placed uniformly on the circumference of the ring.
- Update the position of the neurons as described in Algorithm 3 .

Guidelines for Kohonen's Model

- The decision variables of the COP are represented by the weight vector.
- Initially, all the cities of the input layer are connected with all the neurons of output layer.
- Update the weights of the neurons as described in Algorithm 4 .

Note that only COPs which have a low-dimensional geometrical structure can be solved using the Durbin-Willshaw's model or the Kohonen's model. However, there has been extension proposed for solving other COPs, like self-organizing neural networks (SONNs) [108]. SONNs exploit the fact that solutions to many optimization problems can be seen as finding the best arrangement in the permutation matrix. The best arrangement is the one which is both feasible and optimal to the given COP.

Guidelines for Lagrangian Model

- The output state of the neurons (both Lagrangian and variable neurons) represents the decision variable of the COP.
- Combine the objective function and constraints using the Lagrange method. This overall Lagrange function is called the energy function of the system.
- Obtain the weights and the bias. This is the main step that maps COP onto ANN. This can be done by any one of the following methods:
  - Compare the coefficients of terms in energy function with the coefficients in the $L(\mathbf{x}, \boldsymbol{\lambda})$ function. This comparison will determine the weights on the links and will determine the input bias of a neuron.
  - Compare the coefficients of degree 1 in the differential update rule to the weights. Similarly, compare the constant term of differential update rule to the bias.
- Assign a random initial state to all neurons and proceed as described in Algorithm 5 .

In the following subsection, a detail mapping of a COP using the H-T's model will be presented.

2.8 Example: Mapping

In this subsection, an illustration of mapping a COP to the H-T's model will be presented. A symmetric traveling salesman problem (TSP) is selected for the illustration because this problem has been considered as a standard representative test problem for COPs.

Hopfield Mapping

Consider the TSP formulated as

$$\text{minimize :} \quad \sum_{i=1}^{N} \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{N} d_{ik} x_{ij} (x_{k,i+1} + x_{k,i-1}) \tag{29}$$

subject to :

$$\sum_{i=1}^{N} x_{ij} = 1 \quad \forall j = 1, \ldots, N \tag{30}$$

$$\sum_{j=1}^{N} x_{ij} = 1 \quad \forall i = 1, \ldots, N \tag{31}$$

$$x_{ij} \in 0, 1 \quad \forall i, j = 1, \ldots, N \tag{32}$$

where

$$x_{ij} = \begin{cases} 1 & \text{if city } i \text{ is in the position } j \\ 0 & \text{otherwise} \end{cases}$$

and d_{ik} represents the distance between the cities i and j. The first step in mapping TSP to a H-T's model will be converting the given problem into an unconstrained quadratic programming problem, using the penalty functions. Following transformations are used by Hopfield and Tank to map the problem:

- The function in Eq. (30) is transformed into

$$\sum_{i=1}^{N} \sum_{k=1,k\neq i}^{N} x_{ij} x_{kj}$$

- The function in Eq. (31) is transformed into

$$\sum_{j=1}^{N} \sum_{k=1,k\neq j}^{N} x_{ij} x_{ik}$$

- In order to preserve the equality of 1 in Eqs. (30) and (31), additional penalty is added as

$$\left(\sum_{i=1}^{N} \sum_{j=1}^{N} x_{ij} - N \right)^2$$

Thus, the penalized objective function can be written as

$$E_{\text{tsp}} = \frac{A}{2}\sum_{j=1}^{N}\sum_{i=1}^{N}\sum_{k=1,k\neq i}^{N} x_{ij}x_{kj} + \frac{B}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1,k\neq j}^{N} x_{ij}x_{ik}$$
$$+\frac{C}{2}\left(\sum_{i=1}^{N}\sum_{j=1}^{N} x_{ij} - N\right)^2 + \frac{D}{2}\sum_{i=1}^{N}\sum_{k=1,k\neq i}^{N}\sum_{j=1}^{N} d_{ik}x_{ij}(x_{k,i+1} + x_{k,i-1}) \quad (33)$$

Comparing the Eq. (33) with (1), we have the following values for the synaptic weights and the input bias:

$$\mathrm{w}_{ijkl} = -A\delta_{ik}(1-\delta_{jl}) - B\delta_{jl}(1-\delta_{ik}) - C - D\delta_{ik}(\delta_{l(j+1)} + \delta_{l(j-1)}) \quad (34)$$
$$\theta_{ij} = CN \quad (35)$$

where δ_{ik} is the Kronecker Delta.[9] With the values of weight and bias, H-T's energy function can be written as

$$E_{\text{tsp}} = -\frac{1}{2}\sum_{i,j,k,l=1}^{N} \mathrm{w}_{ijkl}x_{ij}x_{kl} - \sum_{i,j=1}^{N}\theta_{ij}x_{ij} \quad (36)$$

Thus, the system represented by Eqs. (34)–(36) can be implemented via an analog circuit or stepwise simulation on a digital computer. Although this mapping is not the best mapping (see [119] for a better mapping) for TSP, however, the purpose of this mapping was to illustrate the reader about various changes that are needed for mapping a simple COP onto a given ANN.

In this section, some of the classical models of the ANNs applied to solve the COPs were reviewed. However, the conditions under which the ANNs will converge to the optimal solution are not addressed in this section. In Sect. 3, conditions that guarantee the stability and the convergence ability of ANNs while solving the COPs will be presented.

3 Optimality Conditions

In the previous sections, it was stated that the ANNs will perform a gradient descent method. However, there were no mathematical analysis or proofs provided to support the claim. In this section, the stability and convergence analysis of ANNs will be presented. The objective of the such analysis is to prevent the network from oscillation or chaos and to guarantee its convergence to a local minimum [83, 106]. That is, to analyze that under what conditions from any arbitrary initial point, the ANN will converge to a local stable point. Although the stability term is used frequently from the electrical engineering point of view but from the view

[9] $\delta_{ik} = 0$ if $i \neq k$; $\delta_{ik} = 1$ if $i = k$.

of optimization, these are more likely to be called as the optimality conditions. Therefore, in the following subsections, the necessary and sufficient conditions that lead the ANNs to converge at the local minima will be analyzed.

3.1 System Dynamics

The method of selecting neurons for the update in a given ANN is called its system dynamics. In the previous subsection, a random method to select a neuron for update is mentioned. However, it should be a valid question to ask, why to select a neuron randomly for the update? Why not select them in a synchronous, parallel, or consecutive way? This subsection investigates the best methods of selecting neurons. Before analyzing different update rules, in this subsection, different dynamics that can be applied to an ANN are described for a discrete state discrete dynamics system.

Consider a general form of discrete time ANN with the following dynamic equations:

$$\mathcal{I}_i(t+1) = \sum_{j=1}^{n} \mathrm{w}_{ij} x_j(t) + \theta_i \tag{37}$$

$$x_i(t+1) = \Psi_d(\mathcal{I}_i) = \begin{cases} 1 & \text{if } \mathcal{I}_i > U_i \\ -1 & \text{if } \mathcal{I}_i \leq U_i \end{cases} \tag{38}$$

where x_i and $\mathcal{I}_i$ represent the external state and internal state of an ith neuron and w_{ij} represents the weight between an ith neuron and a jth neuron. θ_i is a constant representing an external input to the ith neuron. Function $\Psi_d()$ is called as the transfer function. It assigns binary values to x_i. It should be noted that function $\Psi_d()$ is slightly different from the conventional *sign or signum* function. This is due to the simple observation that a single dormant neuron (i.e., not-firing neuron) will never excite itself (since $w_{ii} = 0$). U_i represents the threshold value of the ith neuron. If the aggregate sum of internal energies supplied to the ith neuron is greater than U_i, then this neuron will "fire," that is, x_i will be set to 1. On the other hand, if the aggregate sum of internal energies is less than the threshold, the neuron will be in a dormant or "not-firing" state. Note that, here the external state of the neurons can take values of -1 or 1. The dynamic equation of the discrete state system can be written in a compact form as

$$x_i(t+1) = \Psi_d\left(\sum_{j=1}^{n} \mathrm{w}_{ij} x_j(t) + \theta_i\right) \tag{39}$$

The three main types of dynamics that can be implemented in any ANN are described as:

- Synchronous Dynamics

$$x_i(t+1) = \Psi_d\left(\sum_{j=1}^{n} w_{ij}\, x_j(t) + \theta_i\right) \quad \forall\ i = 1, \ldots, n \tag{40}$$

- Parallel Dynamics

$$x_i(t+1) = \begin{cases} \Psi_d\left(\sum_{j=1}^{n} w_{ij}\, x_j(t) + \theta_i\right) & \forall\ i \in S(t) \\ x_i(t) & \text{otherwise} \end{cases} \tag{41}$$

- Consecutive Dynamics

$$x_i(t+1) = \begin{cases} \Psi_d\left(\sum_{j=1}^{n} w_{ij}\, x_j(t) + \theta_i\right) & i = K \\ x_i(t) & \text{otherwise} \end{cases} \tag{42}$$

where $S(t)$[10] represents the set of selected neurons at time t and K[11] represents the single selected vertex at time t. The above equations were defined for discrete state discrete dynamic system. That is, if the system is updated in discrete time, then the system is called as the discrete time system. Similarly, if the state (external) of the system is discrete, it is called as the discrete state system. On the other hand, if the system is updated in continuous time, it is called as the continuous time system. Moreover, if it has continuous state, it is called as the continuous state system. Thus, in general, all the ANN can be classified as discrete states discrete dynamics (DSDD), continuous state discrete dynamics (CSDD), continuous state continuous dynamics (CSCD), and discrete state continuous dynamics (DSCD). In the following sub sections, stability and optimality analysis of different systems will be presented.

3.2 Lyapunov Energy Function

When Hopfield introduced the notion of feedback type ANN, Lyapunov function was used to analyze the convergence characteristics of the H-T's model. This is

[10] They can be selected randomly; however, the best way to select them is considering an independent set of neurons. This will be discussed in the following subsections.

[11] There are many ways to select the k based on the distribution function defined by:

- The gradient rule [50]
- The probabilistic-gradient rule
- The Hopfield rule [56]

However, these distribution functions are independent and have the same mean value (n) [83]. Thus, they convey the idea of random selection of the Kth neuron.

done in order to prove that H-T's models are stable in nature and are capable of finding solutions to the optimization problem. Keeping the importance of Lyapunov analysis of ANNs in COPs, some of the main results from the literature will be presented in this subsection. Before proceeding for the Lyapunov analysis, some of the definitions are to be stated.

Basic Definitions:

Consider a dynamic system of the following form:

$$\nabla \mathbf{x}(t) = E(\mathbf{x}(t)), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \in \mathbb{R}^n \tag{43}$$

Definition 1 A point $\mathbf{x}^*$ is called the equilibrium point (or critical point, or steady state point) of the system described by Eq. (43) if $E(\mathbf{x}^*) = 0$.

Definition 2 A function $f : \mathbb{R}^n \mapsto \mathbb{R}$ is said to be a Lyapunov function if it holds the following properties:

- Function f is continuous and differentiable.
- The partial derivatives of function f w.r.t. any element of $\mathbf{x}$ are continuous.
- Function f is nonnegative function over its entire domain.
- The derivative of function f w.r.t. time is nonpositive.

Definition 3 For the system described by Eq. (43), an equilibrium point $\mathbf{x}^*$ is said to be Lyapunov stable (or stable in the sense of Lyapunov) if for any positive scalar ϵ, there exists a positive scalar δ such that

$$||\mathbf{x}(t_0) - \mathbf{x}^*|| < \delta \quad \longrightarrow \quad ||\mathbf{x}(t) - \mathbf{x}^*|| < \varepsilon \quad \forall t \geq t_0$$

Definition 4 An equilibrium point $\mathbf{x}^*$ is said to be asymptotically stable (or asymptotically stable in the sense of Lyapunov) if the following conditions are satisfied:

- $\mathbf{x}^*$ is stable.
- $\lim_{t\to\infty} \mathbf{x}(t) = \mathbf{x}^*$.

For the detailed explanation of the above definitions, see [76].

Analysis of DSDD Systems

Now, let us begin with systems of type DSDD. The two widely known Lyapunov functions for the DSDD systems can be written as

- Lyapunav 1

$$E_{\mathrm{DD1}}(t+1) = -\langle \mathbf{x}(t+1), W\mathbf{x}(t)\rangle + \langle \boldsymbol{\theta}, (\mathbf{x}(t+1) + \mathbf{x}(t))\rangle \tag{44}$$

or

$$E_{\mathrm{DD1}}(t+1) = -\sum_{i=1}^{n}\sum_{j=1}^{n} \mathrm{w}_{ij} x_i(t+1) x_j(t)$$

$$+\sum_{i=1}^{n}\theta_i(x_i(t+1)+x_i(t)) \tag{45}$$

- Lyapunav 2

$$E_{\mathrm{DD2}}(t+1) = -\frac{1}{2}\langle \mathbf{x}(t+1), W\mathbf{x}(t+1)\rangle + \langle \boldsymbol{\theta}, (\mathbf{x}(t+1))\rangle \tag{46}$$

or

$$E_{\mathrm{DD2}}(t+1) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \mathrm{w}_{ij}x_i(t+1)x_j(t+1) + \sum_{i=1}^{n}\theta_i(x_i(t+1)) \tag{47}$$

Theorem 1 *If W is symmetric and update rule is synchronous, then a discrete system defined by Eq. (45) will converge to either a local minima or to a two-edge cycle.*

Proof The change brought to the energy function given by Eq. (45) in one iteration is

$$\begin{aligned}\triangle E_{\mathrm{DD1}}(t+1) &= (E_{\mathrm{DD1}}(t+1)) - (E_{\mathrm{DD1}}(t)) \\ &= \left(-\sum_{i=1}^{n}\sum_{j=1}^{n}\mathrm{w}_{ij}x_i(t+1)x_j(t) + \sum_{i=1}^{n}\theta_i(x_i(t+1)+x_i(t))\right) \\ &\quad -\left(-\sum_{i=1}^{n}\sum_{j=1}^{n}\mathrm{w}_{ij}x_i(t)x_j(t-1) + \sum_{i=1}^{n}\theta_i(x_i(t)+x_i(t-1))\right)\end{aligned} \tag{48}$$

solving Eq. (48) will lead to the following:

$$\begin{aligned}\triangle E_{\mathrm{DD1}}(t+1) &= \left(\sum_{i=1}^{n}\theta_i(x_i(t+1)-x_i(t-1))\right) \\ &\quad +\left(-\sum_{i=1}^{n}\sum_{j=1}^{n}\mathrm{w}_{ij}x_i(t+1)x_j(t) + \sum_{i=1}^{n}\sum_{j=1}^{n}\mathrm{w}_{ij}x_i(t)x_j(t-1)\right)\end{aligned} \tag{49}$$

Changing index in the second term and using symmetry of W matrix, following simplification is obtained:

$$\triangle E_{\mathrm{DD1}}(t+1) = \left(\sum_{i=1}^{n} \theta_i (x_i(t+1) - x_i(t-1))\right) + \left(-\sum_{i=1}^{n}\sum_{j=1}^{n} \mathrm{w}_{ij} x_j(t)(x_i(t+1) - x_i(t-1))\right) \quad (50)$$

Rewriting the above equation,

$$\triangle E_{\mathrm{DD1}}(t+1) = -\sum_{i=1}^{n} (x_i(t+1) - x_i(t-1)) \left(\sum_{j=1}^{n} \mathrm{w}_{ij} x_j(t) - \theta_i\right) \quad (51)$$

For each ith neuron, the contribution to the change in the energy function will be given by

$$(\triangle E_{\mathrm{DD1}}(t+1))_i = -(x_i(t+1) - x_i(t-1)) \left(\sum_{j=1}^{n} \mathrm{w}_{ij} x_j(t) - \theta_i\right) \quad (52)$$

or

$$(\triangle E_{\mathrm{DD1}}(t+1))_i = (x_i(t-1) - x_i(t+1))\, \mathcal{I}_i(t) \quad (53)$$

Thus, for the Lyapunov energy function given by Eq. (45), the change in energy of ith neuron is given by Eq. (3.2).

Since the update rule is synchronous, at every iteration any of the following scenarios may arise:

- If $\mathcal{I}_i(t) > 0 \Rightarrow x_i(t+1) = 1 \Rightarrow (\triangle E_{DD1}(t+1))_i \leq 0 \quad \forall\, i \in N.$
- If $\mathcal{I}_i(t) \leq 0 \Rightarrow x_i(t+1) = 0 \Rightarrow (\triangle E_{DD1}(t+1))_i \leq 0 \quad \forall\, i \in N.$

From the above implication, for every ith neuron, $(\triangle E_{DD1}(t+1))_i \leq 0$. Thus, the system is converging. At the converging limit, $(\triangle E_{DD1}(t+1))_i = 0$. This implies the following cases:

- $\mathcal{I}_i(t) = 0 \Rightarrow x_i(t) = x_i(t+1) = 0 \quad \forall\, i \in N$, a trivial case and can be discarded
- $x_i(t-1) - x_i(t+1) = 0 \quad \forall\, i \in N.$
  - $\Rightarrow x_i(t-1) = x_i(t+1) \neq x_i(t) \quad \forall\, i \in N \quad \Rightarrow$ two-edge cycles.
  - $\Rightarrow x_i(t-1) = x_i(t+1) = x_i(t) \quad \forall\, i \in N \quad \Rightarrow$ local minima.

Thus, the system will converge to a local minimum or to a two-edge cycle. □

The next case to consider will be DSDD system with parallel dynamics. If a similar energy function is used, then following a similar argument, it is observed for the parallel dynamics that if $i \in S(t-1)$ and $i \notin S(t)$, then the sign of Eq. (3.2) is undetermined. Thus, there could be an increase in the energy of the system, if

parallel dynamics is followed. However, this case will not arise, if the set of neurons for parallel update is selected in a specified way and if a different energy function is used. The following theorem will give the *necessary and sufficient* condition for convergence of parallel dynamics.

Theorem 2 *If W is a symmetric matrix with zero diagonals and if the update rule is parallel, then a discrete system defined by Eq. (47) will converge to a local minima if and only if $S(t)$ contains an independent set of neurons for each iteration t. Where $S(t)$ is said to contain independent set of neurons, if $i, j \in S(t)$, then $w_{i,j} = 0$*

Proof Consider a generalized Lyapunov energy function given by Eq. (47). Let $S(t)$ be the independent set of neurons. The change in energy of the system is given by

$$\begin{aligned}\triangle E_{\mathrm{DD2}}(t+1) &= (E_{\mathrm{DD2}}(t+1)) - (E_{\mathrm{DD2}}(t)) \\ &= \frac{1}{2}\left(-\sum_{i=1}^{n}\sum_{j=1}^{n} \mathrm{w}_{ij}\, x_i(t+1)x_j(t+1) + \sum_{i=1}^{n}\theta_i(x_i(t+1))\right) \\ &\quad -\frac{1}{2}\left(-\sum_{i=1}^{n}\sum_{j=1}^{n} \mathrm{w}_{ij}\, x_i(t)x_j(t) + \sum_{i=1}^{n}\theta_i(x_i(t))\right) \qquad (54)\end{aligned}$$

Splitting the above equation for the cases $i, j \in S(t)$ and using symmetric property of W, we get

$$\begin{aligned}\triangle E_{\mathrm{DD2}}(t+1) &= \frac{1}{2}\left(-\sum_{i\in S(t)}\sum_{j\notin S(t)} \mathrm{w}_{ij}\, x_i(t+1)x_j(t+1) + \sum_{i=1}^{n}\theta_i(x_i(t+1))\right) \\ &\quad -\frac{1}{2}\left(-\sum_{i=1}^{n}\sum_{j=1}^{n} \mathrm{w}_{ij}\, x_i(t)x_j(t) + \sum_{i=1}^{n}\theta_i(x_i(t))\right) \qquad (55)\end{aligned}$$

Now, based on the above equation, the following two scenarios may happen:

- If $\mathcal{I}_k(t) > 0 \Rightarrow x_k(t+1) = 1 \Rightarrow \triangle E_{\mathrm{DD2}}(t+1) \leq 0$
- If $\mathcal{I}_k(t) \leq 0 \Rightarrow x_k(t+1) = 0 \Rightarrow \triangle E_{\mathrm{DD2}}(t+1) \leq 0$

Thus, the above system is converging. At the converging limit, $\triangle E_{DD2}(t+1) = 0$, which implies the following cases:

- $\mathcal{I}_k(t) = 0 \Rightarrow x_k(t) = x_k(t+1) = 0 \quad \forall\, k \in N$, a trivial case and can be discarded.
- $x_k(t) - x_k(t+1) = 0 \quad \forall\, k \in N \quad \Rightarrow$ local minima.

Thus, the system will converge to a local minimum. For the proof of the converse, see [83]. □

Theorem 3 *If W is symmetric, the update rule is consecutive, then a discrete system defined by Eq. (47) will converge to a local minimum.*

Proof Consider the Lyapunov energy function given by Eq. (47). The change in energy is given by

$$\begin{aligned}\triangle E_{\mathrm{DDC}}(t+1) &= (E_{\mathrm{DDC}}(t+1)) - (E_{\mathrm{DDC}}(t)) \\ &= \frac{1}{2}\left(-\sum_{i=1}^{n}\sum_{j=1}^{n} \mathrm{w}_{ij}\, x_i(t+1)x_j(t+1) + \sum_{i=1}^{n}\theta_i(x_i(t+1))\right) \\ &\quad -\frac{1}{2}\left(-\sum_{i=1}^{n}\sum_{j=1}^{n} \mathrm{w}_{ij}\, x_i(t)x_j(t) + \sum_{i=1}^{n}\theta_i(x_i(t))\right) \qquad (56)\end{aligned}$$

Splitting the above equation for cases when $i = K$ and when $i \neq K$ and using the property that W is symmetric, we get

$$\triangle E_{\mathrm{DDC}}(t+1) = (x_k(t) - x_k(t+1))\, \mathcal{I}_k(t) \qquad (57)$$

The following two scenarios may happen:

- If $\mathcal{I}_k(t) > 0 \Rightarrow x_k(t+1) = 1 \Rightarrow \triangle E_{\mathrm{DDC}}(t+1) \leq 0$
- If $\mathcal{I}_k(t) \leq 0 \Rightarrow x_k(t+1) = 0 \Rightarrow \triangle E_{\mathrm{DDC}}(t+1) \leq 0$

Thus, the above system is converging. At the converging limit, $\triangle E_{DDC}(t+1) = 0$, which implies the following cases:

- $\mathcal{I}_k(t) = 0 \Rightarrow x_k(t) = x_k(t+1) = 0 \quad \forall\ k \in N$, a trivial case and can be discarded.
- $x_k(t) - x_k(t+1) = 0 \quad \forall\ k \in N \quad \Rightarrow$ local minimum.

Thus, the system will converge to a local minimum. □

Theorem 4 *If W is symmetric positive semidefinite, then a discrete system defined by Eq. (47) will converge to a local minimum.*

Proof See [83]. □

The above analysis is performed for DSDD system. Another important class of system is CSDD system. Similar to the discrete state, three main types of dynamics can be applied for the continuous state system; they can be described as

- Synchronous Dynamics

$$x_i(t+1) = \Psi_c\left(\sum_{j=1}^{n} \mathrm{w}_{ij}\, x_j(t) + \theta_i\right) \quad \forall\ i = 1, \ldots, n \qquad (58)$$

- Parallel Dynamics

$$x_i(t+1) = \begin{cases} \Psi_c\left(\sum_{j=1}^{n} w_{ij} x_j(t) + \theta_i\right) & \forall\ i\ \in\ S(t) \\ x_i(t) & \text{otherwise} \end{cases} \tag{59}$$

- Consecutive Dynamics

$$x_i(t+1) = \begin{cases} \Psi_c\left(\sum_{j=1}^{n} w_{ij} x_j(t) + \theta_i\right) & i = K \\ x_i(t) & \text{otherwise} \end{cases} \tag{60}$$

where $S(t)$ and K have the same meaning as in the DSDD case. The main difference between the above equations and the discrete state equations is the definition of the transfer function. Here the transfer function is not similar to *sign* function. However, here the transfer can be a general real valued monotonically increasing continuous function with the following properties:

$$\begin{aligned} if\ a\ \to\ \infty,\ \Psi_c(a)\ \to\ 1 \quad \forall\ a\ \in\ \mathbb{R} \\ if\ a\ \to\ -\infty,\ \Psi_c(a)\ \to\ -1 \quad \forall\ a\ \in\ \mathbb{R} \end{aligned} \tag{61}$$

Apart from the above properties, the following property is also incorporated to maintain an easy shift from a discrete state to a continuous state neuron. This is one of the important property of the transfer function, which adds the flexibility of using a continuous state model for solving a discrete COP.

$$\Psi_c(\mu a)\ \to\ \Psi_d(a) \quad as\ \mu\ \to\ \infty \tag{62}$$

Even the monotonic increasing criteria is not necessary. To be specific, any monotonic nondecreasing continuous function with the above properties may be used as the transfer function. However, the monotonic increasing continuous function is used only for the sake of simplicity and clearness [83].

There are many standard mathematical functions that appropriately fit the above restriction for the transfer function. The following are some of the widely used transfer functions:

1. $\Psi_c(a) = \tan h(a)$
2. $\Psi_c(a) = \frac{2}{\pi}\tan^{-1}(a)$
3. $\Psi_c(a) = \frac{1-e^{-a}}{1+e^{-a}}$
4. $\Psi_c(a) = \cot h(a) - \frac{1}{a}$

In the following part of this section, some more theorems without proof for the CSDD systems are presented. Interested readers are directed to [83] for the detailed proofs of the following theorems.

Theorem 5 *If W is symmetric, $w_{ii} \geq 0$, and the update rule is synchronous, then a discrete system will converge to a local minima or a two-edge cycle, if the following energy function is used.*

$$E_{\mathrm{CDS}}(t+1) = -\sum_{i=1}^{n}\sum_{j=1}^{n} \mathrm{w}_{ij}\, x_i(t+1)x_j(t) + \sum_{i=1}^{n} \theta_i \left(x_i(t+1) + x_i(t) + \sum_{i=1}^{n} \left(\int_0^{x_i(t+1)} \Psi_{c_i}^{-1}(a)\, d(a) + \int_0^{x_i(t)} \Psi_{c_i}^{-1}(a)\, da \right) \right) \tag{63}$$

Theorem 6 *If W is symmetric and positive semidefinite, then a discrete system will converge to a local minima (irrespective of the type of dynamic used), if the following energy function is used.*

$$E_{\mathrm{CD}}(t+1) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \mathrm{w}_{ij}\, x_i(t+1)x_j(t) + \sum_{i=1}^{n} \theta_i(x_i(t+1)) + \sum_{i=1}^{n} \int_0^{x_i(t+1)} \Psi_{c_i}^{-1}(a)\, da \tag{64}$$

The restriction of positive semidefinite W can be relaxed if Ψ_c and ${\Psi_c}^{-1}$ are differentiable. Let ζ_i be a scalar, defined as

$$\Psi_{c_i}' \leq \zeta_i \quad \text{and} \quad [{\Psi_c}^{-1}]' \geq \frac{1}{\zeta_i} \tag{65}$$

If such ζ_i's exists, then the following theorem holds:

Theorem 7 *Let $\widehat{w}_{ij} = w_{ij} + \frac{\delta_{ij}}{\zeta_{ij}}$, where $\delta_{i,j}$ is Kronecker delta. Let the energy function be defined as*

$$E_{\mathrm{CD}}(t+1) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \mathrm{w}_{ij}\, x_i(t+1)x_j(t) + \sum_{i=1}^{n} \theta_i(x_i(t+1)) + \sum_{i=1}^{n} \int_0^{x_i(t+1)} \Psi_{c_i}^{-1}(a)\, da \tag{66}$$

If $\widehat{W}$ is positive definite, then a discrete system will converge to a local minimum (irrespective of the type of dynamic used).

Furthermore, for the specific case of consecutive dynamics, a more relaxed optimality condition can be used.

Theorem 8 *Let the energy function be defined as*

$$E_{\mathrm{CD}}(t+1) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \mathrm{w}_{ij}\, x_i(t+1)x_j(t) + \sum_{i=1}^{n}\theta_i(x_i(t+1)) + \sum_{i=1}^{n}\int_0^{x_i(t+1)} \Psi_{c_i}^{-1}(a)\, da \qquad (67)$$

If $w_{ii} \geq 0$, $w_{ij} + \frac{1}{\zeta_i} > 0$, *the update rule is consecutive, and* Ψ_c & ${\Psi_c}^{-1}$ *are differentiable, then a discrete system will converge to a local minimum.*

A similar analysis can be done for continuous dynamics. Interested readers are referred to [83] for detailed analysis. The above theorems can be easily extended to $\{0, 1\}$ systems. Continuous dynamics are usually hard to simulate on the digital computers and are mostly used in designing the analog circuits. The typical way of implementing continuous dynamics on the digital computer is to discretize the time into small steps. Nevertheless, the objective of presenting the proofs for discrete state systems is to highlight the reader that the convergence of ANNs is a critical issue. Moreover, the convergence of ANNs depends not only on the energy function but also on the systems dynamics. However, it can also be seen from the above proofs that a careful design of energy function (based on various properties of the weight matrix, W) will converge the system to a local minimum, irrespective of the dynamics.

Although Lyapunov function analysis is enough to understand that only for specific scenarios ANNs will converge to local optima. Lyapunov analysis is useful for those optimization problems whose constraints and objective function can be converted into a quadratic form. Moreover, the constraints are to be incorporated into the Lyapunov function using an appropriate penalty function. In general, it is hard to obtain a Lyapunov energy function for a given system. This does not mean that ANNs cannot be applied to these systems. Therefore, for such systems, different types of energy functions are designed.

The primary approach that replaced the usage of Lyapunov energy function analysis is the usage of a general penalty function analysis (by incorporating constraints of a given optimization problem [77]). In the following subsections, the convergence and stability criteria for penalty- and Lagrange-based ANNs will be presented.

3.3 Penalty-Based Energy Functions

Consider the following constrained optimization problem:

$$\text{minimize :} \quad f(\mathbf{x})$$

$$\begin{aligned} &\text{subject to :} \\ &g_i(\mathbf{x}) \geq 0 \quad \forall i = 1, \ldots, m \\ &\mathbf{x} \in \mathbb{R}^n \end{aligned} \tag{68}$$

where, $f, g_i : \mathbb{R}^n \mapsto \mathbb{R}$. It is assumed that both f, g_i are continuously differentiable functions. The above problem can be converted to an unconstrained optimization problem as

$$\begin{aligned} &\text{minimize :} \\ &E_{\text{pen}}(\mathbf{x}) \\ &\text{subject to :} \\ &\mathbf{x} \in \mathbf{R}^n \end{aligned} \tag{69}$$

where

$$E_{\text{pen}}(\mathbf{x}) = f(\mathbf{x}) + \sum_{i=1}^{m} c_i P_i(\mathbf{x}) \tag{70}$$

and $P_i(\mathbf{x})$ is the penalty function. Now, the convergence of ANNs to solve the above problem depends upon the structure of the penalty function. In this work, convergence analysis based on the penalty function of the form $P_i(\mathbf{x}) = \frac{1}{2}[\min\{0, g_i(\mathbf{x})\}]^2 \; \forall \, i = 1, \ldots, m$ will be presented. One of the properties of this penalty function is that it can be converted to a continuous differentiable penalty function [77]. Consider the following transformation:

Let

$$v_i = \begin{cases} c_i \, g_i(\mathbf{x}) & \text{if } g_i(\mathbf{x}) < 0 \\ 0 & \text{if } g_i(\mathbf{x}) \geq 0 \end{cases} \quad \forall \, i = 1, \ldots, m \tag{71}$$

then,

$$c_i \, P_i(\mathbf{x}) = \frac{c_i}{2}[\min\{0, g_i(\mathbf{x})\}]^2 = \frac{1}{2} v_i \, g_i(\mathbf{x}) \quad \forall \, i = 1, \ldots, m \tag{72}$$

since $g_i, \; \forall \, i = 1, \ldots, m$, is assumed to be differentiable, the above penalty function is differentiable.

The dynamics of this system[12] is defined as

$$\frac{dx_i}{dt} = -\frac{1}{C_i} \frac{\partial E(\mathbf{x})}{\partial x_i} \qquad \forall \, i = 1, \ldots, n \tag{73}$$

Thus, these systems are also called as *gradient-based systems*. These systems [77] have some of the useful properties that can be used to solve the COPs. These properties are described in the following theorems:

[12] Using Kirchoffs law, see [77]

Theorem 9 *Let the system be described by Eq. (73). The isolated equilibrium point of the system is asymptotically stable if and only if it is a strict local minimizer of function* E_{pen}.

Theorem 10 *Let the system be described by Eq. (73). If the isolated equilibrium point of the system is stable, then it is a local minimizer of function* E_{pen}.

For the gradient-based systems, assuming Lipschitz continuity, Han et al. [51] proposed many other convergence properties.

One of the difficulties in applying the penalty based method in the ANNs, is the selection of various penalty parameters. Due to the physical limits on the analog circuits, c_i cannot be set to a large value. In order to overcome the difficulties of the penalty method, and to generalize the use of ANNs, Lagrange based energy function was proposed [139]. Following subsection discuss some of the results for the Lagrange-based ANNs.

3.4 Lagrange Energy Functions

Lagrange based neural networks are inspired by the saddle point theory and the duality analysis. The focus of these networks is to search for a point that satisfies K.K.T's first order necessary conditions of optimality [12]. Therefore, analogous to the gradient descent systems, these network are also known as *projective gradient systems*.

In the following subsections, necessary and sufficient conditions related to Lagrange type energy functions will be presented. Consider the following optimization problem:

$$\begin{aligned} &\text{minimize :} \\ &\qquad f(\mathbf{x}) \\ &\text{subject to :} \\ &\qquad h_i(\mathbf{x}) = 0 \quad \forall i = 1, \ldots, m \\ &\qquad \mathbf{x} \in \mathbb{R}^n \end{aligned} \tag{74}$$

where, $f, h_i : \mathbb{R}^n \mapsto \mathbb{R}$. It is assumed that both, f, h_i are continuously differentiable functions. The above problem can be converted to an unconstrained optimization problem using the Lagrange function, as:

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T h(\mathbf{x}) \tag{75}$$

In this subsection, some of the basic optimality condition in the form of theorems are stated without proof. Interested readers may refer to [12] for the complete discussion, with proofs.

Theorem 11 *Let us assume that* $\mathbf{x}^*$ *is a regular point (one that satisfies constraint qualification). If* $\mathbf{x}^*$ *is the local minimum of problem Eq. (74), then there exists* $\boldsymbol{\lambda}^* \in \mathbb{R}^m$ *such that:*

$$\nabla_x L(\mathbf{x}^*, \boldsymbol{\lambda}^*) = 0 \tag{76}$$

and

$$\mathbf{d}^T \nabla^2_{xx} L(\mathbf{x}^*, \boldsymbol{\lambda}^*)\mathbf{d} \geq 0, \quad \forall\ \mathbf{d}\ \in \{\mathbf{d} : \nabla_x h(\mathbf{x}^*)^T \mathbf{d} = 0\} \tag{77}$$

Theorem 12 *Let us assume that* $h(\mathbf{x}^*) = 0$. *If* $\exists\ \boldsymbol{\lambda}^* \in \mathbb{R}^m$ *such that:*

$$\nabla_x L(\mathbf{x}^*, \boldsymbol{\lambda}^*) = 0 \tag{78}$$

and

$$\mathbf{d}^T \nabla^2_{xx} L(\mathbf{x}^*, \boldsymbol{\lambda}^*)\mathbf{d} > 0, \quad \forall\ \mathbf{d}\ \in \{\mathbf{d} : \mathbf{d} \neq 0; \nabla_x h(\mathbf{x}^*)^T \mathbf{d} = 0\} \tag{79}$$

then $\mathbf{x}^*$ *is the local minimum of problem Eq. (74).*

Similar to the case of gradient-based dynamics, Lagrange systems has the following dynamic equations:

$$\nabla_t \mathbf{x} = -\nabla_x L(\mathbf{x}, \boldsymbol{\lambda}) \tag{80}$$

$$\nabla_t \boldsymbol{\lambda} = -\nabla_\lambda L(\mathbf{x}, \boldsymbol{\lambda}) \tag{81}$$

Although, the above equations are similar to the gradient-based systems, due to the difference in type of variables (ordinary variables and Lagrange variables), the system is named as *projective gradient systems*. In the presence of different types of the variables, the Lagrange will decrease w.r.t. one type of the variables and increase w.r.t. the other. This leads to the saddle point theory. For example, it is very easy to see that

$$\frac{dL(\mathbf{x}, \boldsymbol{\lambda})}{dt}\Big|_{\boldsymbol{\lambda}=\text{constant}} = \sum_{i=1}^{n} \frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial x_i} \frac{dx_i}{dt} = -\sum_{i=1}^{n} \left(\frac{dx_i}{dt}\right)^2 \leq 0 \tag{82}$$

$$\frac{dL(\mathbf{x}, \boldsymbol{\lambda})}{dt}\Big|_{\mathbf{x}=\text{constant}} = \sum_{i=1}^{m} \frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial \lambda_i} \frac{d\lambda_i}{dt} = \sum_{i=1}^{m} \left(\frac{d\lambda_i}{dt}\right)^2 \geq 0 \tag{83}$$

Thus, $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ act as a saddle point of the system.

Theorem 13 *If* $\nabla^2_{xx} L(\mathbf{x}, \boldsymbol{\lambda})$ *is positive definite* $\forall\ \mathbf{x}\ \in\ \mathbb{R}^n$ *and* $\forall\ \boldsymbol{\lambda}\ \in\ \mathbb{R}^m$ *, then the proposed ANN is Lyapunov stable.*

The proof of this theorem is presented by Zhang and Constantinides in [139]. From the above theorem, following corollaries can be obtained:

Corollary 1 $\mathbf{x}$ *tends to a limit* $\mathbf{x}^*$ *at which* $\nabla_x L(\mathbf{x}^*, \boldsymbol{\lambda}) = 0$.

Corollary 2 *If* $\nabla h_i(\mathbf{x}^*)\ \ \forall\ \ i\ \ =\ \ 1, \ldots, m$ *are linearly independent, then* $\boldsymbol{\lambda}$ *converges to* $\boldsymbol{\lambda}^*$.

Theorem 14 *Let the network be described by Eqs. (75), (80), and (81). If the network is physically stable, then the ANN converges to a local minimum. The point of convergence is a Lagrange solution to problem Eq. (74).*

The above restriction of the global convexity can be relaxed to a local convexity, and the following theorem can be stated:

Theorem 15 *Let the network be described by Eqs. (75), (80), and (81). If following conditions hold:*
1. $\exists(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ *such that* $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ *is a stationary point*
2. $\nabla^2_{xx} L(\mathbf{x}^*, \boldsymbol{\lambda}^*) > 0$
3. $\mathbf{x}^*$ *is a regular point of problem Eq. (74)*
the initial point is in proximity of $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$*, then the ANN will asymptotically converge to* $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$.

In this section, different ways to study convergence and stability performance of ANNs were illustrated. Various theorems were presented to understand the critical role of the energy functions in the convergence behavior of ANNs. However, the convergence will guarantee local optimality provided suitable energy function, and dynamics are used to update the states of neurons. For obtaining global optimality, there are numerous extensions proposed in the literature. In the following section, a discussion on the methods that can be used to escape from local minima, in the hope of finding the global minimum, is presented.

4 Escaping Local Minima

The very thought of the gradient descent method will kindle the question regarding global optimality. Classical ANNs perform gradient descent method and will hardly converge to the global minimum. In order to overcome such difficulties, various extensions have been proposed. The extension methods that may converge ANNs to global minima by escaping local minima can be broadly classified as:

- Stochastic extensions
- Chaotic extensions
- Convexification
- Hybridization

Clearly, any extension that can be applied to H-T models can be easily applied to the general penalty or the Lagrange-based ANNs. Thus, in most of the cases, the extensions for the H-T's model will be reviewed. In the following part of this section, a brief discussion of the above extension methods will be presented.

4.1 Stochastic Extensions

There are basically four ways to include stochasticity in the H-T's model [107]. They are:

1. Introducing stochasticity in the transfer function
2. Introducing noise in the synaptic weights

3. Introducing noise in the input bias
4. Any combination of the above three

The prominent stochastic extensions that can be applied to the classical ANNs are discussed in the following parts of this subsection.

4.1.1 Boltzmann Machines

This is the first extension of H-T's model, which proposes a way to escape from the local minima [1, 2, 54, 55]. This is done by replacing the deterministic transfer function with a probabilistic transfer function. Let the change in energy of the ith neuron of the H-T's model be represented by ∇E_i. Then, irrespective of the previous state, the ith neuron will fire with probability p_i, which is defined as

$$p_i = \frac{1}{(1 + \exp -\nabla E_i / T)} \tag{84}$$

and

$$x_i = \Psi_{\text{Blz}}(\mathcal{I}_i) = \begin{cases} 1 & \text{with probability } p_i \\ 0 & \text{with probability } (1 - p_i) \end{cases} \tag{85}$$

where T is the parameter similar to the concept of temperature in simulated annealing.

4.1.2 Gaussian Machines

These are the extensions to the Boltzmann machines, where noise is introduced in the inputs [7]. The updating dynamics of an ith neuron in the H-T's model will be modified as

$$\mathcal{I}_i = \sum_{j=1}^{n} w_{ij} x_j + \theta_i + \varepsilon_i \tag{86}$$

where ε_i is the term representing the noise, which follows Gaussian distribution with mean zero and variance σ^2. The deviation of σ depends upon the parameter T, the temperature of the Gaussian network. In addition to the noise, activation level a_i is being assigned to every neuron, which has the following dynamics:

$$\frac{da_i}{dt} = -\frac{a_i}{\tau} + \mathcal{I}_i \tag{87}$$

The external state of the ith neuron is determined by the following equation:

$$x_i = \Psi_{\text{Gss}}(a_i) = \frac{1}{2}\left(\tanh\left(\frac{a_i}{a_0}\right) + 1\right) \tag{88}$$

where a_0 is the additional parameter added to the Gaussian network.

4.1.3 Cauchy Machines

This system is an extension of the Gaussian machines, where instead of a Gaussian noise, a Cauchy noise is added to the system [114, 116]. The reason of replacing the Gaussian noise with the Cauchy noise is based on the observation that the Cauchy noise produces both global random flights and local random flights, whereas Gaussian noise produces only local random noise. Thus, the system has a better chance of convergence to the global minimum. The probability that a neuron will fire is given as

$$p_i = \frac{1}{2} + \frac{1}{\pi}\left(\arctan(\frac{\mathcal{I}_i}{T})\right) \tag{89}$$

and

$$x_i = \Psi_{\text{Cau}}(\mathcal{I}_i) = \begin{cases} 1 & \text{with probability } p_i \\ 0 & \text{with probability } (1 - p_i) \end{cases} \tag{90}$$

where T is the temperature of the system which has the similar meaning as in the Gaussian or Boltzmann machines. In [63], comparison of lower bounds for Boltzmann and Cauchy machines is presented.

4.2 Chaotic Extensions

Let E_{Hop} represent the general H-T's energy function. Then chaotic extensions for the H-T's model can be obtained by adding an additional function to E_{Hop} [73, 111], which can be described as

$$E_{\text{CNN}} = E_{\text{Hop}} + H(\mathbf{x}, W, \boldsymbol{\theta}) \tag{91}$$

Different definitions for the function $H()$ lead to different chaotic extensions. Some of the chaotic extension methods for the H-T's model will be presented in the following subsection.

4.2.1 Chen and Ahira's Model

Chen and Ahira [20, 21] proposed a chaotic addition to the energy function, which is given as

$$H_{\text{CA}} = \frac{\lambda(t)}{2} \sum_i x_i (x_i - 1) \tag{92}$$

where $\lambda(t)$ is the decay parameter. If this term is added to H-T's energy function, then the system's dynamics is defined by the following equation:

$$\frac{d\mathcal{I}_i}{dt} = -\frac{\partial E_{\text{CNN}}}{\partial x_i} \tag{93}$$

The update rule for the internal energy of this model is given as

$$\mathcal{I}_i(t+1) = k\mathcal{I}_i(t) + \alpha\left(\sum_{j=1, j\neq i}^{n} w_{ij}x_j(t) + \theta_i\right) - z(t)(x_i(t) - \theta_0) \tag{94}$$

$$z(t+1) = (1-\beta)z(t) \tag{95}$$

where $\alpha = \nabla t$, $z(t)$ is the self-feedback connection weight $z(t) = \lambda(t)\nabla t$, β is the damping factor, $\theta_0 = 1/2$, and $k = 1 - \nabla t/\tau$.

Initially, z is very high to create strong self-coupling, thereby leading to the chaotic dynamics. Chaotic dynamics acts like an exploration stage in the simulated annealing, searching for a global minima. In the later stages, z is decayed so that the system starts to converge to a stable fixed point (similar to exploitation stage in the simulated annealing).

4.2.2 Wang and Smith's Model

Wand and Smith [124] proposed the following chaotic addition to the H-T's energy function, which is given as

$$H_{\mathrm{WS}} = (\beta^T - 1)E_c \tag{96}$$

where E_c is the original H-T's energy function and β is the parameter which takes the values between $[0, 1]$. If this term is added to H-T's energy function, then the system is defined as

$$\frac{d\mathcal{I}_i}{dt} = -\frac{\partial E_{\mathrm{CNN}}}{\partial x_i} \tag{97}$$

The update rule of the internal energy for this model is given as

$$\mathcal{I}_i(t+1) = \left(1 - \frac{\beta^T \nabla t(0)}{\tau}\right)\mathcal{I}_i(t) + \beta^T \nabla t(0)\left(\sum_{j=1, j\neq i}^{n} w_{ij}x_j(t) + \theta_i\right) \tag{98}$$

$$\beta^T \nabla t(0) = \nabla t(t) \tag{99}$$

Initially, ∇t is sufficiently large to trigger chaotic search. As $t \to \infty$, ∇t gradually decreases.

4.2.3 Chaotic Noise Model

Recently, Wang et al. [125] proposed the following chaotic addition to the H-T's energy function, which is given as

$$H_\eta = -\sum_i \eta_i(t)x_i(t) \tag{100}$$

where η_i is the noise term. In general, η_i is a normalized time series. Initially, the neuron is set with the chaotic time series. If this term is added to H-T's energy function, then the system is defined as

$$\frac{d\mathcal{I}_i}{dt} = -\frac{\partial E_{\text{CNN}}}{\partial x_i} \tag{101}$$

The update rule of the internal state for this model is given as

$$\mathcal{I}_i(t+1) = \left(1 - \frac{\nabla t}{\tau}\right)\mathcal{I}_i(t) + \nabla t\left(\sum_{j=1}^{n} w_{ij}\, x_j(t) + \theta_i + \eta_i(t)\right) \tag{102}$$

This model has a linear form of function $H()$, when compared to the quadratic forms presented in the earlier models. This may be the reason for its superior optimization performance among the chaotic extension models.

4.3 Convexification

One of the important properties of a convex optimization problem is that the local minimum is the global minimum. Thus, convex reformulation of the problem and finding a local minimum of the reformulation will give the global minimum. If the actual model is nonconvex, then convexification leads to different approximations (or relaxations), which in turn may be useful in studying various bounds (lower and/or upper bound) of a given COP. Both penalty-based ANNs and Lagrange-based ANNs can be convexified. Convexification of COPs using penalty function depends upon the type of penalty function. Thus, it will not be discussed in this subsection. Nevertheless, convexification of Lagrange-based ANNs will be discussed. From the Theorem 15, it can be seen that the local convexity of Lagrange systems is hard to obtain in practice. However, an augmented Lagrange method can be used to convexify any Lagrange-based ANN. In this subsection, one of the convexification methods that can be used for Lagrange-based ANN is presented. Let us assume that problem Eq. (74) is nonconvex, then its augmented Lagrange can be written as

$$L_{\text{Aug}}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T h(\mathbf{x}) + \frac{1}{2} c |h(\mathbf{x})|^2 \tag{103}$$

Using Eq. (103), different ANNs can be designed which are known as Augmented Lagrange Neural Networks (ALNN). The state equations can be obtained as

$$\frac{dx_i}{dt} = -\frac{\partial f}{\partial x_i} - \sum_{j=1}^{m}(\lambda_j + c h_j)\frac{\partial h_j}{\partial x_i} \quad \forall\, i = 1, \ldots, n \tag{104}$$

$$\frac{d\lambda_j}{dt} = h_j, \quad \forall\, j = 1, \ldots, m \tag{105}$$

By selecting a sufficiently large value of c, the local convexity for any Lagrange neural networks can be obtained under some restrictions (see [13]). However, due to analog circuit and implementing restrictions, the selection of c will become a critical issue [139].

4.4 Hybridiziation

ANNs have been coupled with metaheuristics like simulated annealing [103, 123–125], tabu search [43], and evolutionary algorithm [102, 126] for solving COPs. The main purpose of these hybridizations is to find the global optimal solution, by adding flexibility to the classical H-T's model. The usual method of coupling H-T's model with a metaheuristic is to incorporate the constraint handling technique of the ANNs in the global search technique of the metaheuristic.

Moreover, there has been research to hybridize H-T's model with SONNs (see [108]). One of the successful implementations is illustrated by Smith et al. [109]. The experimental result from the illustration indicates that a careful and appropriate hybridization of H-T's model with SONNs will result in a better solution methodology, which can be competitive to the state-of-the art optimization solvers.

5 General Optimization Problems

In this section, a discussion on applying ANNs to some of general optimization problems will be presented. For the sake of simplicity and convenience, following general optimization problem will be considered:

- Linear programming problems
- Convex programming problems
- Quadratic programming problems
- Nonlinear programming problems
- Complementarity problems
- Mixed integer programming problem

The objective of selecting mixed integer programming problem is to highlight the reader, the method of transformation between discrete and continuous variables, when applying ANNs.

5.1 Linear Programming

Linear programming (LP) is the most widely used optimization technique for solving numerous optimization problems. Dantzig [30], the father of linear programming, proposed the famous and widely used simplex method to solve LP problems. However, simplex method has an exponential complexity. A polynomial time algorithm (ellipsoid method) for solving LP problems was presented by Khachiyan [68]. Although the ellipsoid method is polynomial time algorithm, in practice simplex

outperforms ellipsoid method. Later, in 1984, Karmarkar [66] proposed a more efficient algorithm than ellipsoid method, and it outperforms simplex method in few complicated problems like scheduling, routing, and planning. Based on the solution methodology, the simplex method is categorized as exterior point method, whereas Karmarkar's algorithm is categorized as interior point method. Further studies [122] on the classification of these two fundamentally different ideas lead to a conclusion that LP problems are finite in nature. Moreover, simplex method is a finite algorithm suitable to solve LP problems. However, it was also noted that sometimes, infinite algorithms like interior point method may outperform exterior point algorithms. Thus, there has been always a curiosity among researchers to investigate the infinite algorithms. ANNs approach in solving COPs can be seen as an infinite approach.

From the demonstration of Hopfield and Tank [58], ANNs were seen as an alternative solution approaches to solve the COPs. Many studies has been conducted to solve LP problems using ANNs [25–28, 35, 67, 100, 136]. Most of these models were extensions of H-T's continuous model. As noted in Sect. 2.7, these models will have an energy function-based model for minimization. Before presenting the models, consider the following notations for LP problem in canonical LP_c and standard LP_s form:

LP_c

minimize :

$$c^T x$$

subject to :

$$g_i(x) \geq 0 \quad \forall i = 1, \ldots, m$$

LP_s

minimize :

$$\widetilde{c}^T x$$

subject to :

$$\widetilde{g}_i(x) = 0 \quad \forall i = 1, \ldots, m$$

Following are the prominent models from the literature.

5.1.1 Kennedy and Chua's Model

Kennedy and Chua's [67] model has neurons similar to the H-T's continuous model. The external states of this model are represented by x. The equations of updates for states and energy function are given as

$$\nabla \mathbf{x} = C^{-1}(-\mathbf{c} - \nabla \mathbf{g}(\mathbf{x})\, \phi(\mathbf{g}(\mathbf{x})))$$

$$E(\mathbf{x}(\mathbf{t})) = \mathbf{c}^T \mathbf{x} + \sum_{j=1}^{m} \int_0^{g_j(\mathbf{x})} \phi_j(s)\, ds$$

where C is a matrix of capacitance and is usually taken as I for most of the cases. Kennedy and Chua showed that Tank and Hopfield's model is a special case of the proposed model. Furthermore, they proved that their model will converge to a steady state, under the assumption that E is bounded from below. Selection of $\phi()$ is similar to that of a penalty function. In [80], it was shown that $\phi()$ is indeed

an implicit form of the penalty function. The most widely used representation of $\phi()$ is $s\ g_i^-(\mathbf{x})$, where $s > 0$ is the penalty weight. Moreover, $g_i^-(\mathbf{x})$ is same as $-\min\{g_i(\mathbf{x}), 0\}$. Based on the idea of applying penalty function, there were many extension of neural networks for LP. Some of the prominent extensions can be seen in [100, 138]. These prominent extensions will be presented in the following part of this section.

5.1.2 Rodriquez-Vazquez et al. Model

Rodriquez-Vazquez et al. [100] proposed another model for LP. This model is described as

$$\nabla \mathbf{x} = -u(\mathbf{x})\mathbf{c} - s(1 - u(\mathbf{x}))\nabla \mathbf{g}(\mathbf{x})\,\mathbf{v}(\mathbf{x})$$

where $\mathbf{v}(\mathbf{x}) = [v_1, \ldots v_m]^T$,

$$u(\mathbf{x}) = \begin{cases} 1 & \text{if } g_i(\mathbf{x}) \geq 0 \quad \forall\, j = 1, \ldots, m, \\ 0 & \text{otherwise} \end{cases}$$

$$v_j(\mathbf{x}) = \begin{cases} 1 & \text{if } g_j(\mathbf{x}) \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

and $s > 0$. This model can be viewed as a dynamic system, which can start from any arbitrary point. Initially, the model will move from an infeasible point to a feasible point. Once a feasible point is obtained, the model will try to move to an optimal point. In [45], the convergence analysis of Rodriquez-Vazquez et al. [100] model was performed. It was shown that if the problem is convex, network will converge. However, Zak et al. [138] proposed that discrete time implementation of Rodriquez-Vazquez et al. model with an adaptive step size may have problem in reaching feasible region. In the following paragraph, the model proposed by Zak et al. will be presented.

5.1.3 Zak et al. Model

This model differs from the earlier ANNs for the LP, as Zak et al. used the exact penalty function (non-differentiable) to formulate the energy function. This selection of penalty function eliminates the selection of very high penalty weight. Apart from that, all the earlier models were proposed for the canonical representation of LP. However, this model is applicable for both standard and canonical representation of LP. Although, in the theory, there is no difference in solving the problem in any representation. But in the case of neural networks, changing from one representation to other will increase in the number of neurons (due to slack or artificial variables) and may increase in the complexity of the ANN model. The proposed model can be described as

$$\nabla \mathbf{x} = -\nabla \mathbf{E}_{p,\rho,\gamma}(\mathbf{x})$$

$$\nabla \mathbf{E}_{p,\rho,\gamma}(\mathbf{x}) = \widetilde{\mathbf{c}}^{T}\mathbf{x} + \rho\nabla\|\widetilde{\mathbf{g}}(\mathbf{x})\|_p + \gamma\nabla\left(\sum_{i=1}^{m} x_i^{-}\right)$$

where $x_i^{-} = -\min\{x_i, 0\},\ \rho > 0,\ \gamma > 0,\ 1 \leq p \leq \infty$. Although the model uses exact penalty function, this penalty function is also non-differentiable and hence has few drawbacks. In [137], convergence analysis of Zak et al.'s model is performed. It was shown that the system trajectory will converge to the solution of LP under some positivity restriction on $\rho,\ \gamma,\ p$. In fact, it is shown that equilibrium points of the ANN are solutions to the corresponding LP problem [137].

These are some of the prominent models of ANNs applied to solve LP problems. Similar to the duality theory in LP, there were attempts to solve LP using dual ANNs. Interested reader may refer to [29, 81].

5.2 Convex Programming

Consider the following convex problem:

$$\begin{aligned} &\text{minimize :} \\ &\qquad f(\mathbf{x}) \\ &\text{subject to :} \\ &\qquad g_i(\mathbf{x}) = 0 \quad \forall i = 1 \ldots m \\ &\qquad \mathbf{x} \in \mathbf{X} \end{aligned} \tag{106}$$

where $f(\mathbf{x}), g_i(\mathbf{x}) : \mathbb{R}^n \mapsto \mathbb{R}$ are any convex functions of $\mathbf{x}$. In order to apply ANNs to solve this problem, the problem is reformulated into unconstrained optimization problem using penalty function. Let $\Phi()$ be the penalty function which has the following properties:

$$\Phi(\mathbf{x}) \begin{cases} = 0 & \text{if } \mathbf{x} \text{ is feasible} \\ > 0 & \text{otherwise} \end{cases} \tag{107}$$

Then, consider the following transformed convex problem:

$$\begin{aligned} &\text{minimize :} \\ &\qquad \mathbf{E}(\mathbf{x}, s) \\ &\text{subject to :} \\ &\qquad \mathbf{x} \in \mathbf{X} \end{aligned}$$

where $E(\mathbf{x}, s) = f(\mathbf{x}) + s\sum_{i=1}^{m} \Phi(g_i(\mathbf{x}))$.

The update equation is

$$\nabla \mathbf{x} = -\eta \nabla \mathbf{E}(\mathbf{x}, s) \tag{108}$$

$$\nabla \mathbf{E}(\mathbf{x}, s) = \nabla f(\mathbf{x}) + s \sum_{i=1}^{m} \phi(g_i(\mathbf{x})) \nabla g_i(\mathbf{x})$$

The stability and convergence analysis of this model has been performed in [22]. It has been shown that dynamics given by Eq. (108) leads to an equilibrium point.

5.3 Quadratic Programming

Similar to the linear programming, quadratic programming problems can be modeled using ANNs. Consider the following quadratic programming problem:

$$\begin{aligned} &\text{minimize :} \\ &\qquad \mathbf{x}^T H \mathbf{x} + c^T \mathbf{x} \\ &\text{subject to :} \\ &\qquad A\mathbf{x} = \mathbf{b} \\ &\qquad \mathbf{x} \in \mathbf{X} \end{aligned} \tag{109}$$

where $\mathbf{x} \in \mathbb{R}^n$ $A, H \in \mathbb{R}^{n \times n}$. This program can be converted to an unconstrained problem using a penalty function method or a Lagrange method. A Lagrange method will be presented in the following part of this section. The Lagrange of problem Eq. (109) is written as

$$L(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{x}^T H \mathbf{x} + c^T \mathbf{x} + \boldsymbol{\lambda}^T (A\mathbf{x} - \mathbf{b}) \tag{110}$$

The equations of motion that describe this model are

$$\nabla_t \mathbf{x} = -\nabla_x L(\mathbf{x}, \boldsymbol{\lambda}) \tag{111}$$

$$\nabla_t \boldsymbol{\lambda} = -\nabla_\lambda L(\mathbf{x}, \boldsymbol{\lambda}) \tag{112}$$

where

$$\nabla_t \mathbf{x} = \left(\frac{dx_1}{dt}, \frac{dx_2}{dt}, \ldots, \frac{dx_n}{dt} \right)^T, \quad x \in \mathbb{R}^n$$

$$\nabla_T \boldsymbol{\lambda} = \left(\frac{d\lambda_1}{dt}, \frac{d\lambda_2}{dt}, \ldots, \frac{d\lambda_m}{dt} \right)^T, \quad \lambda \in \mathbb{R}^m$$

$$\nabla_x L(\mathbf{x}, \boldsymbol{\lambda}) = \left(\frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial x_1}, \frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial x_2}, \ldots, \frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial x_n} \right)^T$$

and

$$\nabla_\lambda L(\mathbf{x}, \boldsymbol{\lambda}) = \left(\frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial \lambda_1}, \frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial \lambda_2}, \ldots, \frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial \lambda_m} \right)^T$$

The above model is iterative Lagrangian neural network. The convergence analysis of this model is presented in Sect. 3 and in [139]. Apart from that, quadratic programming problems with ANNs have been exhaustively studied in [39, 80, 134, 135].

5.4 Nonlinear Programming

Similar to the quadratic and convex programming, general nonlinear programming problems can be solved using ANNs. One of the ways is to use a penalty method approach [67]. Another approach is based on the Lagrange method, presented by Zhang and Constantinides [139]. The main differences with penalty function method and the advantages of Lagrange methods are discussed in Sect. 3 of this chapter. In the following part of this subsection, the method of writing a Lagrange for any generalized nonlinear problem will be presented.

$$\begin{aligned} &\text{minimize :} \\ &\qquad f(\mathbf{x}) \\ &\text{subject to :} \\ &\qquad h_i(\mathbf{x}) = 0 \quad \forall i = 1 \ldots m \\ &\qquad \mathbf{x} \in \mathbb{R}^n \end{aligned} \tag{113}$$

where $f(\mathbf{x}), h_i(\mathbf{x}) : \mathbb{R}^n \mapsto \mathbb{R}$ are any continuous and twice differentiable functions of $\mathbf{x}$. The Lagrange of the above COP is written as

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T h(\mathbf{x}) \tag{114}$$

The equations of motion that describes this model are

$$\nabla_t \mathbf{x} = -\nabla_x L(\mathbf{x}, \boldsymbol{\lambda}) \tag{115}$$

$$\nabla_t \boldsymbol{\lambda} = -\nabla_\lambda L(\mathbf{x}, \boldsymbol{\lambda}) \tag{116}$$

where

$$\nabla_t \mathbf{x} = \left(\frac{dx_1}{dt}, \frac{dx_2}{dt}, \ldots, \frac{dx_n}{dt} \right)^T, x \in \mathbb{R}^n$$

$$\nabla_t \boldsymbol{\lambda} = \left(\frac{d\lambda_1}{dt}, \frac{d\lambda_2}{dt}, \ldots, \frac{d\lambda_m}{dt} \right)^T, \lambda \in \mathbb{R}^m$$

$$\nabla_x L(\mathbf{x}, \boldsymbol{\lambda}) = \left(\frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial x_1}, \frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial x_2}, \ldots, \frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial x_n} \right)^T$$

and

$$\nabla_\lambda L(\mathbf{x}, \boldsymbol{\lambda}) = \left(\frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial \lambda_1}, \frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial \lambda_2}, \ldots, \frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial \lambda_m} \right)^T$$

The above model is iterative, similar to that of H-T's continuous model. Both the types of neuron are decision variables. Lagrangian variables tend to take the solution to a feasible space, whereas the ordinary variables will take the solution to an optimal value. The mathematical properties and proofs related to convergence of this model have been discussed in Sect. 3.

5.5 Complementarity Problem

5.5.1 Linear Complementarity Problem (LCP)

Given a matrix $M \in \mathbb{R}^{n \times N}$ and a vector $\mathbf{q} \in \mathbb{R}^n$, the LCP can be stated as:

$$\begin{aligned} &\text{find :} \\ &\mathbf{x} \\ &\text{subject to :} \\ &M\mathbf{x} + \mathbf{q} \geq 0 \\ &x^T(M\mathbf{x} + \mathbf{q}) = 0 \\ &\mathbf{x} \geq 0 \end{aligned} \tag{117}$$

where $x \in \mathbb{R}^n$. This problem can be solved by ANNs, by converting it to a nonlinear problem by using Fischer function. Let $\phi(a, b)$ be the Fischer function, defined as $\phi(a, b) = \sqrt{a^2 + b^2} - a - b$. The purpose of using this function is that it has an important property, which can be stated as

$$\phi(a, b) = 0 \Leftrightarrow a \geq 0,\ b \geq 0 \text{ and } ab = 0 \tag{118}$$

Thus, this property is exploited while solving LCP using ANN. The above property mimics the LCP, which can be represented as

$$\phi_i(x_i, F_i(\mathbf{x})) = 0 \quad \forall\ i \tag{119}$$

where $F_i(\mathbf{x}) = x_i(M\mathbf{x} + \mathbf{q})_i$. Let $\Phi(\mathbf{x})$ represent a vector, where the ith element is $\phi_i(x_i, F_i(\mathbf{x}))$. Then, solving LCP is same as solving the following problem:

$$\text{minimize :}$$
$$F(\mathbf{x}) \tag{120}$$

where

$$F(\mathbf{x}) = \frac{1}{2}||\Phi(\mathbf{x})||^2 \tag{121}$$

Now, the above problem can be solved by ANNs as a quadratic programming problem, using either a H-T's model or Lagrange model.

5.5.2 Nonlinear Complementarity Problem (NCP)

Given a set of functions $F_i : \mathbb{R}^n \mapsto \mathbb{R}^n, \quad \forall\ i\ =\ 1, \ldots, n$, the NCP can be stated as

$$\begin{aligned} &\text{find :} \\ &\qquad \mathbf{x} \\ &\text{subject to :} \\ &F_i(\mathbf{x}) \geq 0 \\ &x_i F(\mathbf{x}) = 0 \\ &\qquad \mathbf{x} \geq 0 \end{aligned} \tag{122}$$

where $x \in \mathbb{R}^n$ and F_i is assumed to be continuously differentiable function. ANNs can be used to solve NCP by using the idea of Fischer function [75]. Similar to the LCP case, define a vector $\Phi(\mathbf{x})$, where each ith element is represented by

$$\phi_i(x_i, F_i(\mathbf{x})) = 0 \quad \forall\ i \tag{123}$$

Thus, solving NCP is same as solving the following problem:

$$\text{minimize :}$$
$$E(\mathbf{x}) \tag{124}$$

where

$$E(\mathbf{x}) = \frac{1}{2}||\Phi(\mathbf{x})||^2 \tag{125}$$

Now, this above problem can be solved by ANNs using a Lagrange model.

5.6 Mixed Integer Programming Problems (MIPs)

The difficulty in solving any optimization problems is not due to the nonlinear (or discrete) representation of the problem. The criteria that separate

difficult and easy problems are the convexity of the problem. A convex problem in a linear or nonlinear representation is tractable, whereas a nonconvex problem in any representation is difficult to solve. By presenting the example of a general MIPs, we would like to highlight the thin line that separates a continuous optimization problem from a discrete optimization problem while applying ANNs.

Consider the following MIPs defined as

$$\begin{aligned}
&\text{minimize :}\\
&\qquad f(\mathbf{x},\mathbf{y})\\
&\text{subject to :}\\
&\qquad g_i(\mathbf{x},\mathbf{y}) \le 0 \quad \forall i = 1,\ldots,m\\
&\qquad h_j(\mathbf{x},\mathbf{y}) = 0 \quad \forall j = 1,\ldots,p\\
&\qquad l_k \le x_k \le u_k \quad \forall k = 1,\ldots,n\\
&\qquad y_r \in \{0,1\} \quad \forall r = 1,\ldots,q
\end{aligned} \tag{126}$$

Similar to any previous approaches, this problem can be converted to unconstrained optimization problem using either a penalty function or a Lagrange function. For the sake of completeness, both approaches are illustrated in the following part of this subsection.

5.6.1 Penalty Function Approach

The modified (penalized) energy function for problem Eq. (126) can be written as

$$\begin{aligned}
E_{\text{mip}}(\mathbf{x},\mathbf{y}) = C_1\, f(\mathbf{x},\mathbf{y}) + C_2 \sum_{i=1}^{m} \Phi[g_i(\mathbf{x},\mathbf{y})]\\
+C_3 \sum_{j=1}^{p} h_j^2(\mathbf{x},\mathbf{y}) + C_4 \sum_{r=1}^{q} y_r(1-y_r)
\end{aligned} \tag{127}$$

where C_1, C_2, C_3, and C_4 are scaling constants, used as the penalty parameters. Φ is the penalty function. The dynamics of the network defined by Eq. (127) can be described as

$$\frac{d\mathcal{I}_k^x}{dt} = -\eta_x \frac{\partial E_{\text{mip}}}{\partial x_k} \quad \forall k = 1,\ldots,n \tag{128}$$

$$\frac{d\mathcal{I}_r^y}{dt} = -\eta_y \frac{\partial E_{\text{mip}}}{\partial y_r} \quad \forall r = 1,\ldots,q \tag{129}$$

$$x_k = \Psi_c(\mathcal{I}_k^x) \quad \forall k = 1,\ldots,n \tag{130}$$

$$y_r = \Psi_d(\mathcal{I}_r^y) \quad \forall r = 1,\ldots,q \tag{131}$$

where η_x and η_y are positive scaling coefficients for the dynamics of the system. In addition to that, Ψ_d can be replaced by the tuned Ψ_c that results in the discrete values of y_r.

5.6.2 Lagrange Function Approach

The Lagrange of problem Eq. (126) can be written as

$$L_{\text{mip}}(x, y, \mu, \nu^1, \nu^2) = f(x, y) + \sum_{i=1}^{m} \mu_i g_i(x, y) + \sum_{j=1}^{p} \nu_j^1 h_j(x, y) + \sum_{r=1}^{q} \nu_r^2 y_r (1 - y_r) \tag{132}$$

where $\mu_i \geq 0 \quad \forall \; i = 1 \ldots m$ and $\nu_j^1, \nu_r^2 \in \mathbb{R} \quad \forall \; j = 1 \ldots p$ and $\forall \; r = 1 \ldots q$. Similar to penalty function approach, the system dynamics can be described as

$$\nabla_t \mathbf{x} = -\nabla_x L_{\text{mip}}(\mathbf{x}, \mathbf{y}, \boldsymbol{\mu}, \boldsymbol{\nu^1}, \boldsymbol{\nu^2}) \tag{133}$$

$$\nabla_t \mathbf{y} = -\nabla_y L_{\text{mip}}(\mathbf{x}, \mathbf{y}, \boldsymbol{\mu}, \boldsymbol{\nu^1}, \boldsymbol{\nu^2}) \tag{134}$$

$$\nabla_t \nu^1 = \nabla_{\nu^1} L_{\text{mip}}(\mathbf{x}, \mathbf{y}, \boldsymbol{\mu}, \boldsymbol{\nu^1}, \boldsymbol{\nu^2}) \tag{135}$$

$$\nabla_t \nu^2 = \nabla_{\nu^2} L_{\text{mip}}(\mathbf{x}, \mathbf{y}, \boldsymbol{\mu}, \boldsymbol{\nu^1}, \boldsymbol{\nu^2}) \tag{136}$$

$$\frac{d\mu_i}{dt} = \begin{cases} 0 & \text{for } \mu_i = 0 \;\&\; \frac{\partial L_{\text{mip}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\mu}, \boldsymbol{\nu^1}, \boldsymbol{\nu^2})}{\partial \mu_i} < 0 \\ \frac{\partial L_{\text{mip}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\mu}, \boldsymbol{\nu^1}, \boldsymbol{\nu^2})}{\partial \mu_i} & \text{otherwise} \end{cases} \quad \forall i \tag{137}$$

In [127], penalty-based approach is applied to solve factory allocation problem, and to solve the unit commitment problem. Direct use of inequality constraints in Lagrange programming ANNs have been introduced in [61, 79]. In Eq. (137), a simple method of update for inequality constrained is illustrated from [79]. A primitive way to deal with the inequality constrains is to transform them into equality constraints. Another novel approach for direct usage of inequality constraints is to use the square of the Lagrange multiplier for the inequality constraints [61].

From this section, it should be clear that ANNs are not limited for solving either a linear or a quadratic programming problem. With development of penalty- and Lagrange-based methods for the ANNs, they can be applied to any typical optimization problem. With the ease of transformation from continuous to discrete variable space, they can be appropriately designed for any COPs. In the following section, some of the applications of ANNs in solving discrete optimization problems will be surveyed.

6 Discrete Optimization Problems

Discrete optimization problems find their application in the problems related to transportation, scheduling, sorting, networking, etc. In this section, various methods of mapping the discrete optimization problems to the ANNs will be presented.

6.1 Graph Problems

Graph problems are one of the extensively studied and applied problems in operations research. The goal of this subsection is to discuss these problems and their mapping onto ANNs [96].

6.1.1 Graph Partitioning Problem (GPP)

Problem Statement: Given a graph $G = (V, E)$, where V represents the vertices, $|V| = n$ and E represents the edges, $|E| = m$. Let w_i be the weight on each vertex. Let e_{ij} be the weight on the link connecting ith and jth vertices. The problem is to partition the graph into two partitions of nearly equal weights, such that the cut-size is minimized. In other words, the problem is to bisect the graph, such that the sum of weights of the vertices assigned to the partitions is nearly equal, while the sum of weights on the links connecting the partitions is minimum.

Solution Methodology: A continuous H-T's model was used to solve this Problem [9]. Let x_i represent the external state of the neuron, which can take any values between 0 and 1. All the neurons that have the value of 0 are assigned to the first partition. The others neurons, which have the value of 1, were assigned to the second partition. The objective function for GPP is given as

$$E_{\text{GPP}} = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} e_{ij}(x_i + x_j - 2x_i x_j) + \sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j (1 - x_i - x_j + 2x_i x_j) \tag{138}$$

The first term of the objective function aims at minimizing the weighted sum of the edges which belong to the cut, whereas the second term of the objective function aims at balancing the weights in both the partitions. When the E_{GPP} is compared with E_c, the following values for the synaptic weights and the input bias are obtained:

$$W_{ij} = 2e_{ij} - 4w_i w_j \quad \text{and} \quad \theta_i = -\sum_{j=1}^{n} e_{ij} + 2w_i \sum_{j=1}^{n} w_j \tag{139}$$

Thus, the H-T's model is complete. The above system is solved to get the solution of GPP.

6.1.2 Graph K-Partitioning Problem (GKP)

Problem Statement: Given a graph $G = (V, E)$, where V represents the vertices, $|V| = n$ and E represents the edges, $|E| = m$. Let w_i be the weight on each vertex. Let e_{ij} be the weight on the link connecting ith and jth vertices. This problem is a generalized version of GPP, where the objective is to partition given graph with similar description as GPP into K partitions. In general, K is not the multiple of 2.

Solution Methodology: This problem can be formulated as an assignment problem, where each node is assigned to only one partition. Since there are K partitions and n nodes, there will be $N = nK$ number of variables. Since, in H-T's model, each variable represents a neuron, there will be altogether N neurons. Let x_{ij} be the external state of the neuron, which can take value between 0 and 1.

The objective function with penalty parameters is written as

$$\begin{aligned} E_{\text{GKP}} = & \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{K}\sum_{k=1,k\neq j}^{K} x_{ij}x_{jk} + \frac{1}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{K} x_{ij} - n\right)^2 \\ & + \frac{1}{2}\sum_{i,j=1}^{n}\sum_{k,l=1}^{K} e_{ij}(x_{ik} + x_{jl} - 2x_{ik}x_{jl}) \\ & + \sum_{k=1}^{K}\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ik}x_{jk}w_i w_j \end{aligned} \tag{140}$$

When the E_{GKP} is compared with E_d, the following values for the weights and input bias are obtained:

$$W_{ij,kl} = -\delta_{ij}(1 - \delta_{ij}) - 1 + 2e_{ij} - 2w_i w_j \quad \text{and} \quad \theta_i = +n - \sum_{j=1}^{n} e_{ij} \tag{141}$$

where $W_{ij,kl}$ is the synaptic weight between neuron ij[13] and neuron kl. Thus, with the above mapping, the H-T's model is complete. The above system is solved by using Algorithm 1 to get the solution of GKP. A higher order ANNs for solving GKP was presented in [40].

6.1.3 Vertex Cover Problem (VCP)

Problem Definition: Given a graph $G = (V, E)$, where V represents the vertices, $|V| = n$ and E represents the edges, $|E| = m$. Let A be the adjacency matrix, where $a_{ij} = 1$ if there is an edge between ith and jth vertices, otherwise $a_{ij} = 0$. A subset C of the vertices V of the graph G is called the *vertex cover* if all the edges

[13] neuron ij corresponds to variable x_{ij}

of G are adjacent to at least one vertex in the set C. The VCP is to find the *vertex cover*, such that the cardinality of set C is minimum.

Solution Methodology: Let

$$x_i = \begin{cases} 1 & \text{if vertex } i \in C \\ 0 & \text{otherwise} \end{cases} \tag{142}$$

Since there are $|V| = n$ vertices in G, there will be n neurons used to design the model. The modified objective function designed for H-T's model will be

$$E_{\text{VCP}} = C_1 \left(\sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j \right) - \frac{C_2}{2} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} (1 - x_i)(1 - x_j) a_{ij} \right) \tag{143}$$

where C_1 and C_2 are penalty terms whose values decide the relative importance of the constraints and the objective function. The first term in E_{VCP} ensures the minimality of the cover. The second term (is the penalty term) will be zero if the nodes which are not in the *vertex cover* have no edges between them. By comparing the coefficients of E_{VCP} and E_d, the synaptic weights and the input bias are obtained as

$$W_{ij} = -2C_1 - C_2 a_{ij} \quad \text{and} \quad \theta_i = C_2 \sum_{j=1}^{n} a_{ij} \tag{144}$$

Thus, with the above mapping, the H-T's model is complete. The above system is solved by using Algorithm 1 to get the solution of GKP.

6.1.4 Maximum Independent Set Problem (MISP)

Problem Definition: Given a graph $G = (V, E)$, where V represents the vertices, $|V| = n$ and E represents the edges, $|E| = m$. Let A represent the adjacency matrix, where $a_{ij} = 1$ if there is an edge between ith and jth vertices, otherwise $a_{ij} = 0$. A subset I of the vertices V of the graph G is called the *independent set* if there is no edge between any pair of vertices which belong to I. The MISP is to find an *independent set* with the maximum cardinality.

Solution Methodology: Let

$$x_i = \begin{cases} 1 & \text{if vertex } i \in I \\ 0 & \text{otherwise} \end{cases} \tag{145}$$

Since there are $|V| = n$ vertices in G, there will be n neurons used to design the model. The modified objective function designed for H-T's model will be

$$E_{\text{MISP}} = -C_1 \left(\sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j \right) + \frac{C_2}{2} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j a_{ij} \right) \tag{146}$$

where C_1 and C_2 are penalty terms whose values decide the relative importance of the constraints and the objective function. The first term in E_{MISP} minimizes cardinality of the set I. The second term will ensure that the set I forms an *independent set*. By comparing the coefficients of E_{MISP} and E_d, the synaptic weights and the input bias are obtained as

$$W_{ij} = 2C_1 - C_2 a_{ij} \quad \text{and} \quad \theta_i = 0 \tag{147}$$

Thus, with the above mapping, the H-T's model is complete. The above system is solved by using Algorithm 1 to get the solution of MISP.

6.1.5 Maximum Clique Set Problem (MCSP)

Problem Definition: Given a graph $G = (V, E)$, where V represents the vertices, $|V| = n$ and E represents the edges, $|E| = m$. Let A represent the adjacency matrix, where $a_{ij} = 1$ if there is an edge between ith and jth vertices, otherwise $a_{ij} = 0$. A subset of the vertices V of the graph G is called *clique* if every pair of vertices in the subset is connected by an edge. The MSCP is to find a *clique* with the maximum cardinality.

Solution Methodology: Let

$$x_i = \begin{cases} 1 & \text{if vertex } i \in clique \\ 0 & \text{otherwise} \end{cases} \tag{148}$$

Since there are $|V| = n$ vertices in G, there will be n neurons used to design the model. Let $a^c_{ij} = 1 - a_{ij}$, then with this modification, MSCP will be same as MISP. The modified objective function designed for H-T's model will be

$$E_{\text{MCSP}} = -C_1 \left(\sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j \right) + \frac{C_2}{2} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j a^c_{ij} \right) \tag{149}$$

where C_1 and C_2 are penalty terms whose values decide the relative importance of the constraints and the objective function. The first term in E_{MCSP} minimizes cardinality of the clique set. The second term will ensure that the selected set forms a clique. By comparing the coefficients of E_{MCSP} and E_d, the synaptic weights and the input bias are obtained as

$$W_{ij} = 2C_1 - C_2 a^c_{ij} \quad \text{and} \quad \theta_i = 0 \tag{150}$$

Thus, with the above mapping, the H-T's model is complete. The above system is solved by using Algorithm 1 to get the solution of MCSP.

6.1.6 Graph Coloring Problem (GCP)

Problem Definition: Given a graph $G = (V, E)$, where V represents the vertices, $|V| = n$ and E represents the edges, $|E| = m$. Let A represent the adjacency matrix, where $a_{ij} = 1$ if there is an edge between ith and jth vertices, otherwise $a_{ij} = 0$. The GCP is to find the minimum number of subsets of V such that no two adjacent vertices (two vertices connected by an edge) belong to the same subset. In other words, find the minimum number of different colors needed to color the graph such that no two adjacent vertices are of the same color.

Solution Methodology: Let

$$x_{ij} = \begin{cases} 1 & \text{if vertex } i \text{ is colored in the color } j \\ 0 & \text{otherwise} \end{cases} \tag{151}$$

Since there are $|V| = n$ vertices in G, from the graph theory, the maximum number of different colors needed to color the graph will be one plus the maximum degree of any vertex. Let v represent the maximum number of different colors. Thus, there will be nv neurons used to design the model. The modified objective function designed for H-T's model will be

$$E_{\text{GCP}} = \frac{1}{2}\left(\sum_{k=1}^{v}\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ik}x_{jk}a_{ij}\right) - \left(\sum_{k=1}^{v}\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ik}x_{jk}\right) \tag{152}$$

By comparing the coefficients of E_{GCP} and E_d, the synaptic weights and the input bias are obtained as

$$W_{ij} = 1 - a_{ij}/2 \quad \text{and} \quad \theta_i = 0 \tag{153}$$

Thus, with the above mapping, the H-T's model is complete. The above system is solved by using Algorithm 1 to get the solution of GCP.

6.1.7 Graph Matching Problem (GMP)

Problem Definition: Given a graph $G = (V, E)$, where V represents the vertices, $|V| = n$ and E represents the edges, $|E| = m$. Let B be the incidence matrix, where $b_{ij} = 1$ if jth edge is incident on ith vertex, otherwise $b_{ij} = 0$. A subset M of the edges E of the graph G is called *matching* if there are no two edges in M that share a common node. The GMP is to find a *matching* with the maximum cardinality.

Solution Methodology: Let

$$x_i = \begin{cases} 1 & \text{if edge } i \in M \\ 0 & \text{otherwise} \end{cases} \tag{154}$$

Since there are $|E| = m$ vertices in G, there will be m neurons used to design the model. The modified objective function designed for H-T's model will be

$$E_{\text{GMP}} = -\left(\sum_{i=1}^{m} x_i\right) + \left(\sum_{i=1}^{m}\sum_{j=1}^{m} x_i x_j \sum_{k=1}^{n} b_{ki} b_{kj}\right) \tag{155}$$

By comparing the coefficients of E_{GMP} and E_d, the synaptic weights and the input bias are obtained as

$$W_{ij} = -2\sum_{k=1}^{n} b_{ki} b_{kj} \quad \text{and} \quad \theta_i = 1 \tag{156}$$

Thus, with the above mapping, the H-T's model is complete. The above system is solved by using Algorithm 1 to get the solution of GMP.

6.2 Shortest Path Problems

Problem Description: Given a network $G = (V, E)$, where V represents the vertices or cities, $|V| = n$ and E represents the edges or paths, $|E| = m$. Let the distance between the cities be represented by a distance matrix D, where d_{ij} represents the distance between the ith city and the jth city. The problem is to find the shortest path between two given pair of cities. Let s be the origin and e be the destination.

Solution Methodology: Let

$$x_{ij} = \begin{cases} 1 & \text{if edge } (i, j) \text{ is in the path} \\ 0 & \text{otherwise} \end{cases} \tag{157}$$

Since there are n cities, there will be n neurons used to design the model. The modified objective function designed for H-T's model will be

$$E_{\text{SPP}}(t) = C_1\left(\sum_{i=1}^{n}\sum_{j\neq i} d_{ij}\exp^{-t/\tau} x_{ij}\right) + \frac{C_2}{2}\sum_{i=1}^{n}\left(\sum_{k\neq i} x_{ik} - \sum_{l\neq i} x_{li} - \delta_{is} + \delta_{ie}\right) \tag{158}$$

By comparing the coefficients of E_{SPP} and E_c, the synaptic weights and input bias are obtained as

$$W_{ij} = 2 * C_2 \quad \text{and} \quad \theta_i = -2 * C_2 \tag{159}$$

Thus, with the above mapping, the H-T's model is complete. The above system is solved by using Algorithm 2 to get the solution of SPP.

6.3 Number Partitioning Problems (NPP)

Problem Definition: Given a set of n numbers $a_1, a_2, \ldots, a_n \in S$, partition the set S into two sets S_1 and S_2 such that the following cost function is minimized:

$$C(S_1, S_2) = \left| \sum_{i \in S_1} a_i - \sum_{i \in S_2} a_i \right| \tag{160}$$

Solution Methodology: Let

$$x_i = \begin{cases} 1 & \text{if edge } i \in S_2 \\ 0 & \text{if edge } i \in S_1 \end{cases} \tag{161}$$

Since there are n numbers, there will be n neurons used to design the model. The modified objective function designed for H-T's model will be

$$E_{\text{NPP}} = \left(\sum_{i=1}^{n} \sum_{j=1}^{n} (1 + 2x_i x_j - x_i - x_j) a_i a_j \right) \tag{162}$$

The term $(1 + 2x_i x_j - x_i - x_j)$ ensures that only numbers which belong to the same partition will contribute to the sum. By comparing the coefficients of E_{GMP} and E_d, the synaptic weights and input bias are obtained as

$$W_{ij} = -4a_i a_j \quad \text{and} \quad \theta_i = \sum_{i=1}^{n} a_i \tag{163}$$

Thus, with the above mapping, the Hopfield model is complete. The above system is solved by using Algorithm 1 to get the solution of GMP.

6.4 Assignment Problems

Problem Definition: Given n entities, n cells, and the cost c_{ij} of assigning ith entity to jth cell. The linear assignment problem (LAP) is to assign each entity to one cell, such that the assignment cost is minimized.

Solution Methodology: Since there are n entities and n cells, there will be altogether n^2 neurons used for designing ANN. Let x_{ij} represent the external state of H-T's model. Let $\mathcal{I}_{ij}$ represent the internal state of H-T's neuron, such that

$$x_{ij} = \begin{cases} 1 & \text{if entity } i \text{ is assigned to cell } j \\ 0 & \text{otherwise} \end{cases} \tag{164}$$

The dynamics of this system is given as

$$\frac{d\mathcal{I}_{ij}(t)}{dt} = -C_1 \sum_{k=1}^{n} v_{kj}(t) - C_1 \sum_{k=1}^{n} v_{ik}(t) + 2C_1 - C_2 c_{ij} \exp(-t/\tau) \tag{165}$$

$$x_{ij}(t) = f_{ij}\left(\mathcal{I}_{ij}(t)\right) \tag{166}$$

From the above equations, the weights and input bias of H-T's model are defined as

$$w_{ij} = \delta_{ij} 2C_1 + (1 - \delta_{ij})C_1 \quad \text{and} \quad \theta_{ij} = 2C_1 \tag{167}$$

Thus, with the above mapping, the H-T's model is complete. The above system is solved by using Algorithm 2 to get the solution of LAP.

6.5 Sorting Problems

Problem Definition: Given n real numbers $a_1, a_2, \ldots, a_n$, the problem is to order them in ascending, descending, or biotonic order (where both the ends have higher numbers).

This problem can be formulated as an assignment problem, that is, given n numbers that are to be assigned to n cells. The cost c_{ij} of assigning ith number to jth cell is given by $c_{ij} = a_i c_j$, where c_j is an arbitrary sequence of real numbers, which are selected based on the sense of sorting (ascending, descending, or biotonic).

Solution Methodology: Since there are n numbers and n cells, there will be altogether n^2 neurons used for designing ANN. Let x_{ij} represent the external state of H-T's continuous model. Let $\mathcal{I}_{ij}$ represent the internal state of H-T's model such that

$$x_{ij} = \begin{cases} 1 & \text{if entity } i \text{ is assigned to cell } j \\ 0 & \text{otherwise} \end{cases} \tag{168}$$

The energy function and dynamics of this system are given as

$$E_{\text{SP}}(x(t), t) = C_1 \left(\sum_{i=1}^{n} \sum_{j=1}^{n} a_i c_j \exp(-t/\tau) x_{ij} \right) + \sum_{i=1}^{n} \frac{C_2^i}{2} \left(\sum_{j=1}^{n} x_{ij} - 1 \right)^2$$

$$+\sum_{j=1}^{n}\frac{C_2^j}{2}\left(\sum_{i=1}^{n}x_{ij}-1\right)^2 \tag{169}$$

$$\frac{d\mathcal{I}_{ij}(t)}{dt}=-C_2^j\sum_{k=1}^{n}v_{kj}(t)-C_2^i\sum_{k=1}^{n}v_{ik}(t)$$
$$+C_2^j+C_2^i-C_1a_ic_j\exp(-t/\tau) \tag{170}$$

$$x_{ij}(t)=f_{ij}\left(\mathcal{I}_{ij}(t)\right) \tag{171}$$

From the above equations, the weights of H-T's model can be obtained as

$$w_{ij}=-\delta_{ij}(C_2^j+C_2^i)-(1-\delta_{ij})C_2^i \quad \text{and} \quad \theta_{ij}=C_2^j+C_2^i \tag{172}$$

Thus, with the above mapping, the H-T's model is complete. The above system is solved by using Algorithm 2 to get the solution of SPP.

6.6 Traveling Salesman Problems (TSP)

Problem Definition: Given n cities with the Euclidean distance between them, d_{ij}. A path that starts from a given city, passing through all the cities (without visiting a city twice), and returning to the first city is called a *tour*. The TSP problem can be defined as finding the shortest tour for the given cities.

Solution Methodology: In Sect. 2, a detailed mapping of TSP onto H-T's model was shown. However, in this subsection, one of the improved TSP mappings [119] will be presented. This is the modification of H-T's approach, which can be described as

$$E_{\text{tsp}}^{\text{mod}}=E_{\text{tsp}}+E_{\text{cns}}+E_{\text{drv}} \tag{173}$$

where E_{tsp} is same as defined in Eq. (33),

$$E_{\text{cns}}=\frac{C1}{2}\sum_i\left(\sum_j x_{ij}-1\right)^2+\frac{C2}{2}\sum_j\left(\sum_i x_{ij}-1\right)^2 \tag{174}$$

$$E_{\text{drv}}=\frac{C3}{2}\sum_i\sum_j x_{ij}(1-x_{ij})+\frac{C4}{2}\left(\frac{N^2}{4}-\sum_j\sum_i\left(x_{ij}-\frac{1}{2}\right)^2\right) \tag{175}$$

and $C1$, $C2$, $C3$, and $C4$ are new penalty parameters that are used for scaling the respective penalty terms. E_{cns} represents a better way of mapping the TSP constraints, whereas E_{drv} represents the penalty that deflects the external state of the neurons to take the value of 0 or 1.

There are numerous COPs in the literature, and presenting ANNs model for each of them will be a cumbersome task. However, a similar approach (like the above models) can be followed for any discrete optimization problem. Moreover, the above mappings were based on H-T's model with penalty approach. As seen in Sect. 3, the capability of ANNs to solve COPs can be extended by using the Lagrange approach. Thus, any discrete optimization problem (linear or nonlinear) can be modeled using the techniques given in Sect. 5. In the following part of this section, a survey of some of the recent models in ANNs is cited.

A recent survey that presents the usage of ANNs in mathematical programming problems is conducted in [128]. Apart from the theoretical COPs, there has been many practical problems that have been mapped onto ANNs. For example, some of the recent applications of ANNs within the past 2 years are

- Resource allocation in wireless communication [17, 78]
- Scheduling in wireless sensors [105]
- Scheduling in packet radio networks [113]
- Scheduling via H-T's model [31]
- Scheduling in water pumps [49]
- Scheduling to minimize earliness and tardiness [8]
- Small-world nature of H-T's model [141]
- Constrained least absolute deviation problem [60]
- Hybrid approach with genetic algorithms [5]
- Hybrid approach with particle swarm optimization [33]
- Hybrid approach with ant colony optimization [129]
- Parameter setting for the GCP [118]
- Placement of electronic circuit problem [36]
- Edge detection in wood logs [95]
- Economic load dispatching problem [48]
- Hybrid routing algorithm [11]
- Data envelopment analysis [59]
- Water system management problem [120]
- Graphs isomorphism discernment problem [140]
- Nonlinear vector encoding problem [41]

7 Criticism

After the eye-catching implementation of applying ANNs in solving TSP, H-T's model caught the attention of many researchers [92]. However, this method had two drawbacks at that time. It has a lot of parameters to select, and it performs a gradient descent method while solving COPs. In [132], it was illustrated that H-T's model does not perform well. In fact, the results in [132] raised serious questions about the validity of this method. Thus, it drifted many researchers away from the use of ANNs in solving COPs. However, some of the researchers were inclined to improve the method proposed by Hopfield and Tank. One of the direction of improvements was to develop a method of optimally selecting the parameters in Hopfield and

Tank's energy function [37, 52, 65, 74, 117]. Recently, in [119], a systematic way of selecting optimal parameters for various TSP is presented. Apart from that, an enhanced method of mapping TSP onto an ANN is illustrated, and convergence and stability analysis is presented. From the results shown in [119], it can be concluded that by proper construction of energy function and by proper selection of parameters (penalty parameters), there can be a significant difference in the performance of ANNs while solving COPs.

Another direction of improvement in Hopfield and Tank's ANN was to free the algorithm from gradient descend method, that is, to avoid falling into the local minima. One of the methods, that was adopted by many researchers, was to modify Hopfield and Tank's objective function [4, 16]. However, some of the successful methods were to use hybrid techniques like convexification, improve choice of penalty function, and use hill-climbing techniques [123]. In [109], a novel approach that combines self-organizing neural networks and H-T's model is presented. Convergence of such networks is analyzed. The algorithm was tested on car-sequencing problem. From the results, it was concluded that hybrid method of ANN is better than MINOS[14] solver. The results from this study showed that proper application of ANNs to solve COPs will prove better than the conventional solution methods of COPs.

Moreover, there were many stochastic and metaheuristic methods that were incorporated in H-T's model. Some of the studies showed that global convergence can be obtained with ANNs if they are properly coupled with metaheuristic methods like simulated annealing [91]. Apart from that, sophisticated penalty methods and Lagrange methods made the use of ANNs more general (i.e., not only applied to linear and quadratic but can be applied to any nonlinear optimization problems). Among the use of penalty methods, in [6, 42], subspace and hyperplane approaches were used. These were the pioneer approaches that modified the use of penalty function, by incorporating the feasibility issues of the system into a single penalty parameter. In [43], the capability of ANNs were demonstrated based on polyhedral combinatorics. It was one of the novel methods in ANNs that proposed the use of memory (like tabu search) to overcome the entrapment of algorithm at the polyhedral vertices. However, not many studies have actually tried to compare performance of ANNs with other conventional or heuristic methods for solving COPs. Thus, it is very hard to conclude if ANNs perform better or worse than other methods. Apart from that, most of the implementations were simulations of ANNs on the digital computers. Thus, the computing limitations of digital computer may hinder the performance of ANNs. Nevertheless, it should be noted that ANNs can overcome the limitations of digital computers since they can be implemented on the analog circuits.

[14] A standard nonlinear programming problem solver

8 Conclusion

In this chapter, a simple and brief introduction of ANNs is presented, which is followed by addressing the usage of ANNs in solving COPs. Our focus throughout this chapter was not only to highlight the application of ANNs in solving COPs but also to present the theoretical aspects of ANNs in solving COPs. At the beginning of this chapter, some of the classical ANN models that were used in solving COPs were illustrated. Along with these illustration, a methodology of implementing an LP to analog circuit is depicted. Moreover, a detail method of mapping TSP on one of the classical ANNs is presented. In addition to that, some of the stability and convergence analysis of ANNs is studied. Lastly, examples from literature on solving general optimization problems and discrete optimization problems using ANNs were presented.

It can be seen that unlike other metaheuristics, based on appropriate conditions, ANNs do converge to local and/or global optima. Simple methods that are used to define such optimality conditions were shown with proofs in Sect. 3 of this chapter. The aim in presenting these proofs was to highlight the importance of selecting proper energy function and the type of dynamics (i.e., consecutive, parallel, or synchronous). Apart from that, methods to extend the classical models (which can solve only linear or quadratic problems) by using penalty functions, Lagrange functions, or hybrid approaches were presented. Through various illustration, it was shown that a continuous model can be used to solve discrete problem very easily. It was underlined that the only change in solving a discrete problem and continuous problem is the adjustment in the transfer function and the adjustment in the energy function for the discrete variables.

Since the first implementation of ANN in 1980s until the present, there are handful analog implementations of ANNs in solving COPs. Due to the limitations of performance comparison of ANNs with other solution procedures, a concrete conclusion regarding the advantage or disadvantage of ANNs in solving COPs cannot be presented at this point. However, proper implementation of the analog circuit and systematic study in deployment of these circuits to solve real world problems will demonstrate the true potential of ANN.

Cross-References

▸Binary Unconstrained Quadratic Optimization Problem
▸Graph Searching and Related Problems
▸Quadratic Assignment Problems

Recommended Reading

1. E. Aarts, J. Korst, *Simulated Annealing and Boltzmann Machines* (Wiley, Chichester, 1988)

2. E.H.L. Aarts, J. Korst, *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing* (Wiley, New York, 1989)
3. E.H.L. Aarts, P.H.P. Stehouwer, J. Wessels, P.J. Zwietering, Neural networks for combinatorial optimization. 1994
4. S. Abe, Global convergence and suppression of spurious states of the Hopfield neural networks. IEEE Trans Circuits Syst.-1 Fundam. Theory Appl. **40**(4), 246–257 (1993)
5. A. Agarwal, S. Colak, J. Deane, NeuroGenetic approach for combinatorial optimization: an exploratory analysis. Ann. Oper. Res. **174**(1), 185–199 (2010)
6. S.V.B. Aiyer, M. Niranjan, F. Fallside, A theoretical investigation into the performance of the Hopfield model. IEEE Trans. Neural Netw. **1**(2), 204–215 (1990)
7. Y. Akiyama, A. Yamashita, M. Kajiura, H. Aiso, Combinatorial optimization with Gaussian machines, in *International Joint Conference on Neural Networks, 1989. IJCNN*, Washington, DC (IEEE, New York, 2002), pp. 533–540
8. D.E. Akyol, G.M. Bayhan, Multi-machine earliness and tardiness scheduling problem: an interconnected neural network approach. Int. J. Adv. Manuf. Technol. **37**(5), 576–588 (2008)
9. J.R. Anderson, Neural networks and NP-complete optimization problems; a performance study on the graph bisection problem. Complex Syst. **2**, 59–89 (1988)
10. J.A. Anderson, J. Davis, *An Introduction to Neural Networks* (MIT, Cambridge, 1995)
11. C.J.A. Bastos-Filho, W.H. Schuler, A.L.I. Oliveira, A fast and reliable routing algorithm based on Hopfield neural networks optimized by particle swarm optimization, in *IEEE International Joint Conference on Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence)* (IEEE, Piscataway, 2008), pp. 2777–2783
12. M.S. Bazaraa, H.D. Sherali, C.M. Shetty, *Nonlinear Programming: Theory and Algorithms* (Wiley, Hoboken, 2006)
13. D.P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods* (Athena Scientific, Belmont, 1996)
14. C.M. Bishop, *Pattern Recognition and Machine Learning*, vol. 4 (Springer, New York, 2006)
15. C. Blum, A. Roli, Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput. Surv. (CSUR) **35**(3), 268–308 (2003)
16. R.D. Brandt, W. Yao, A.J. Laub, S.K. Mitra, Alternative networks for solving the traveling salesman problem and the list-matching problem, in *IEEE International Conference on Neural Networks*, San Diego (IEEE, Piscataway, 1988), pp. 333–340
17. D. Calabuig, J.F. Monserrat, D. Gómez-Barquero, N. Cardona, A delay-centric dynamic resource allocation algorithm for wireless communication systems based on HNN. IEEE Trans. Veh. Technol. **57**(6), 3653–3665 (2008)
18. G.A. Carpenter, Neural network models for pattern recognition and associative memory. Neural Netw. **2**(4), 243–257 (1989)
19. G.A. Carpenter, S. Grossberg, *Pattern Recognition by Self-organizing Neural Networks* (MIT, Cambridge, 1991)
20. L. Chen, K. Aihara, Chaotic simulated annealing by a neural network model with transient chaos. Neural Netw. **8**(6), 915–930 (1995)
21. L. Chen, K. Aihara, Global searching ability of chaotic neural networks. IEEE Trans. Circuits and Syst.-1 Fundam. Theory Appl. **46**(8), 974–993 (1999)
22. Y.H. Chen, S.C. Fang, Solving convex programming problems with equality constraints by neural networks. Comput. Math. Appl. **36**(7), 41–68 (1998)
23. L.O. Chua, G.N. Lin, Non-linear optimization with constraints: a cook-book approach. Int. J. Circuit Theory Appl. **11**(2), 141–159 (1983)
24. L.O. Chua, G.N. Lin, Nonlinear programming without computation. IEEE Trans. Circuits Syst. **31**(2), 182–188 (1984)
25. A. Cichocki, R. Unbehauen, Neural networks for solving systems of linear equations and related problems. IEEE Trans. Circuits Syst. **39**(2), 124–138 (1992)
26. A. Cichocki, R. Unbehauen, Neural networks for solving systems of linear equations and related problems. IEEE Trans. Circuits Syst. 1 Fundam. Theory Appl. **39**(2), 124–138 (1992)

27. A. Cichocki, R. Unbehauen, K. Weinzierl, R. Hölzel, A new neural network for solving linear programming problems. Eur. J. Oper. Res. **93**(2), 244–256 (1996)
28. J.C. Culioli, V. Protopopescu, C. Britton, N. Ericson, Neural network models for linear programming. Technical report, Oak Ridge National Lab., TN (1989)
29. T. Da-Gang, Duality and neural networks for solving linear programming. Acta Autom. Sin. **29**(2), 219-226 (2003)
30. G.B. Dantzig, M.N. Thapa, *Linear Programming: Theory and Extensions* (Springer, New York, 2003)
31. V. David, S. Rajasekaran, Retracted chapter: solving scheduling problems with competitive Hopfield neural networks, in *Pattern Recognition Using Neural and Functional Networks* (Springer, Berlin, 2009), pp. 115–122
32. J.B. Dennis, *Mathematical Programming and Electrical Networks*, (MIT, Cambridge, 1959)
33. Y.H. Duan, Y.L. Gao, PSO algorithm connected with neural network for solving a class of 0/1 optimization problems. Jisuanji Yingyong/J. Comput. Appl. **28**(6), 1559–1562 (2008)
34. R. Durbin, D. Willshaw, An analogue approach to the travelling salesman problem using an elastic net method. Nature, **326**(6114), 689–691 (1987)
35. S. Effati, A.R. Nazemi, Neural network models and its application for solving linear and quadratic programming problems. Appl. Math. Comput. **172**(1), 305–331 (2006)
36. M. Ettaouil, K. Elmoutaouakil, Y. Ghanou, The continuous Hopfield networks (CHN) for the placement of the electronic circuits problem. WSEAS Trans. Comput. **8**(12), 1865–1874 (2009)
37. G. Feng, C. Douligeris, Using Hopfield networks to solve traveling salesman problems based on stable state analysis technique, in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, 2000. IJCNN 2000*, Como, vol. 6 (IEEE, New York, 2002), pp. 521–526
38. T.A. Feo, M.G.C. Resende, Greedy randomized adaptive search procedures. J. Glob. Optim. **6**(2), 109–133 (1995)
39. M. Forti, A. Tesi, New conditions for global stability of neural networks with application to linear and quadratic programming problems. IEEE Trans. Circuits Syst. I Fundam. Theory Appl. **42**(7), 354–366 (2002)
40. G.C. Fox, W. Furmanski, Load balancing loosely synchronous problems with a neural network, in *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications: Architecture, Software, Computer systems, and General Issues-Volume 1* (ACM, New York, 1988), pp. 241–278
41. V. Gardaševic, R.R. Muller, G.E. Øien, Hopfield neural networks for vector precoding, in *International Zurich Seminar on Communications*, Zurich, 2010, p. 66
42. A.H. Gee, Problem solving with optimization networks (1993)
43. A.H. Gee, R.W. Prager, Polyhedral combinatorics and neural networks. Neural Comput. **6**(1), 161–180 (1994)
44. M. Gendreau, J.Y. Potvin, Metaheuristics in combinatorial optimization. Ann. Oper. Res. **140**(1), 189–213 (2005)
45. M.P. Glazos, S. Hui, S.H. Zak, On solving constrained optimization problems with neural networks, in *IEEE International Conference on Neural Networks, 1994. IEEE World Congress on Computational Intelligence, 1994*, Orlando, vol. 7 (IEEE, New York, 2002), pp. 4547–4552
46. F. Glover, G.A. Kochenberger, *Handbook of Metaheuristics* (Springer, 2003)
47. R.E. Gomory, Outline of an algorithm for integer solutions to linear programs. Bull. Am. Math. Soc. **64**(5), 275–278 (1958)
48. D. Gupta, S.K. Jain, Solving combinatorial optimization problem of economic load dispatch using modified Hopfield neural network, in *Joint International Conference on Power System Technology and IEEE Power India Conference, 2008. POWERCON 2008* (IEEE, Piscataway, 2009), pp. 1–6
49. M. Hajji, A. Fares, F. Glover, O. Driss, Water pump scheduling system using scatter search, tabu search and neural networks the case of bouregreg water system in morocco. ASCE (2010)

50. A. Haken, M. Luby, Steepest descent can take exponential time for symmetric connection networks. Complex Syst. **2**(2), 191–196 (1988)
51. Q. Han, L.Z. Liao, H. Qi, L. Qi, Stability analysis of gradient-based neural networks for optimization problems. J. Glob. Optim. **19**(4), 363–381 (2001)
52. S.U. Hedge, J.L. Sweet, W.B. Levy, Determination of parameters in a Hopfield/Tank computational network, in *IEEE International Conference on Neural Networks, 1988*, San Diego (IEEE, San Diego, 2002), pp. 291–298
53. A. Hertz, M. Widmer, Guidelines for the use of meta-heuristics in combinatorial optimization. Eur. J. Oper. Res. **151**(2), 247–252 (2003)
54. G.E. Hinton, T.J. Sejnowski, Learning and relearning in Boltzmann machines, in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*, ed. by D.E. Rumelhart, J.L. McClelland MIT, Cambridge, 1986, pp. 283–317
55. G.E. Hinton, T.J. Sejnowski, D.H. Ackley, *Boltzmann Machines: Constraint Satisfaction Networks that Learn* (Carnegie-Mellon University, Department of Computer Science, Pittsburgh, 1984)
56. J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities. Proc. Natl. Acad. Sci. USA **79**(8), 2554 (1982)
57. J.J. Hopfield, Neurons with graded response have collective computational properties like those of two-state neurons. Proc. Natl. Acad. Sci. USA **81**(10), 3088 (1984)
58. J.J. Hopfield, D.W. Tank, Neural computation of decisions in optimization problems. Biol. Cybern. **52**(3), 141–152 (1985)
59. S.C. Hu, Y.K. Chung, Y.S. Chen, Using Hopfield neural networks to solve DEA problems, in *IEEE Conference on Cybernetics and Intelligent Systems, 2008* (IEEE, Piscataway, 2008), pp. 606–611
60. X. Hu, C. Sun, B. Zhang, Design of recurrent neural networks for solving constrained least absolute deviation problems. IEEE Trans. Neural Netw. **21**(7), 1073–1086 (2010)
61. Y. Huang, Lagrange-type neural networks for nonlinear programming problems with inequality constraints, in *44th IEEE Conference on Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05*, Sevilla (IEEE, Sevilla, 2006), pp. 4129–4133
62. J.L. Huertas, A. Rueda, A. Rodriguez-Vazquez, L.O. Chua, Canonical non-linear programming circuits. Int. J. Circuit Theory Appl. **15**(1), 71–77 (1987)
63. H. Jeong, J.H. Park, Lower bounds of annealing schedule for Boltzmann and Cauchy machines, in *International Joint Conference on Neural Networks, 1989. IJCNN*, Washington, DC (IEEE, 2002), pp. 581–586
64. G. Joya, M.A. Atencia, F. Sandoval, Hopfield neural networks for optimization: study of the different dynamics. Neurocomputing **43**(1–4), 219–237 (2002)
65. B. Kamgar-Parsi, Dynamical stability and parameter selection in neural optimization, in *International Joint Conference on Neural Networks, 1992. IJCNN*, Baltimore, vol. 4 (IEEE, 2002), pp. 566–571
66. N. Karmarkar, A new polynomial-time algorithm for linear programming, in *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, Washington, DC (ACM, New York, 1984), p. 311
67. M.P. Kennedy, L.O. Chua, Neural networks for nonlinear programming. IEEE Trans. Circuits Syst **35**(5), 554–562 (1988)
68. L.G. Khachiian, Polynomial algorithms in linear programming. Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki **20**, 51–68 (1980)
69. V. Klee, G.J. Minty, How good is the simplex algorithm (1970)
70. T. Kohonen, *Self-Organization and Associative Memories* (Springer, Heidelberger, 1982)
71. T. Kohonen, Self-organized formation of topologically correct feature maps. Biol. Cybern. **43**(1), 59–69 (1982)
72. T. Kohonen, *Self-Organization and Associative Memory* (Springer, Berlin/New York, 1989); S. Ridella, S. Rovetta, R. Zunino, Plastic algorithm for adaptive vector quantization. Neural Comput. Appl. **1**, 152–159 (1998)

73. T. Kwok, K.A. Smith, A unified framework for chaotic neural-network approaches to combinatorial optimization. IEEE Trans. Neural Netw. **10**(4), 978–981 (2002)
74. W.K. Lai, G.G. Coghill Genetic breeding of control parameters for the Hopfield/Tank neural net, in *International Joint Conference on Neural Networks, 1992. IJCNN*, Baltimore, vol. 4 (IEEE, 2002), pp. 618–623
75. L.Z. Liao, H. Qi, L. Qi, Solving nonlinear complementarity problems with neural networks: a reformulation method approach* 1. J. Comput. Appl. Math. **131**(1–2), 343–359 (2001)
76. A.M. Liapounoff, *Probleme Général de la Stabilité du Mouvement* (Princeton University Press, Princeton, 1947)
77. W.E. Lillo, M.H. Loh, S. Hui, S.H. Zak, On solving constrained optimization problems with neural networks: a penalty method approach. IEEE Trans. Neural Netw. **4**(6), 931–940 (2002)
78. Y. Liu, M. Jiang, D. Yuan, Cross-layer resource allocation optimization by Hopfield neural networks in OFDMA-based wireless mesh networks, in 2009 Fifth International Conference on Natural Computation (IEEE, Los Alamitos, 2009), pp. 119–123
79. P.B. Luh, X. Zhao, Y. Wang, L.S. Thakur, Lagrangian relaxation neural networks for job shop scheduling. IEEE Trans. Robot. Autom. **16**(1), 78–88 (2002)
80. C.Y. Maa, M.A. Shanblatt, Linear and quadratic programming neural network analysis. IEEE Trans. Neural Netw./A Publication of the IEEE Neural Networks Council **3**(4), 580 (1992)
81. A. Malek, A. Yari, Primal-dual solution for the linear programming problems using neural networks. Appl. Math. Comput. **167**(1), 198–211 (2005)
82. W.S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biol. **5**(4), 115–133 (1943)
83. I.I. Melamed, Neural networks and combinatorial optimization. Autom. Remote Control **55**(11), 1553 (1994) Translated from Avtomatika i Telemekhanika, No. 11, pp. 3–40, 1994.
84. A.N. Michel, J.A. Farrell, Associative memories via artificial neural networks. IEEE Control Syst. Mag. **10**(3), 6–17 (2002)
85. M.L. Minsky, S.A. Papert, *Perceptrons*, Expanded edn. (MIT, Cambridge, 1988)
86. K.S. Narendra, K. Parthasarathy, Gradient methods for the optimization of dynamical systems containing neural networks. IEEE Trans. Neural Netw. **2**(2), 252–262 (2002)
87. I.H. Osman, J.P. Kelly, Meta-heuristics: an overview, in *Meta-Heuristics: Theory and Applications* (Kluwer, Boston, 1996)
88. A.S. Pandya, R.B. Macy, *Pattern Recognition with Neural Networks in C++* (CRC, Boca Raton, 1996)
89. Y. Pao, *Adaptive Pattern Recognition and Neural Networks* (Addison-Wesley, Reading, 1989)
90. M. Papadrakakis, N.D. Lagaros, Reliability-based structural optimization using neural networks and Monte Carlo simulation. Comput. Methods Appl. Mech. Eng. **191**(32), 3491–3507 (2002)
91. M. Peng, N.K. Gupta, A.F. Armitage, An investigation into the improvement of local minima of the Hopfield network. Neural Netw. **9**(7), 1241–1253 (1996)
92. J.Y. Potvin, *The Traveling Salesman Problem: A Neural Network Perspective* (Centre for Research on Transportation, Montréal, 1992)
93. D. Psaltis, N. Farhat, Optical information processing based on an associative-memory model of neural nets with thresholding and feedback. Opt. Lett. **10**(2), 98–100 (1985)
94. J. Puchinger, G.R. Raidl, Combining metaheuristics and exact algorithms in combinatorial optimization: a survey and classification, in *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, ed. by J. Mira (Springer, Berlin, 2005), pp. 41–53
95. D. Qi, P. Zhang, X. Jin, X. Zhang, Study on wood image edge detection based on Hopfield neural network, in *IEEE International Conference on Information and Automation (ICIA), 2010* (IEEE, Piscataway, 2010), pp. 1942–1946
96. J. Ramanujam, P. Sadayappan, Mapping combinatorial optimization problems onto neural networks. Inf. Sci. **82**(3), 239–255 (1995)
97. J. Ramanujam, P. Sadayappan, Optimization by neural networks, in *IEEE International Conference on Neural Networks, 1988*, San Diego (IEEE, San Diego, 2002), pp. 325–332

98. M. Resendel, C. Ribeiro, GRASP with path-relinking: recent advances and applications, in *Metaheuristics: Progress as Real Problem Solvers*, ed. by T. Ibaraki, K. Nonobe, M. Yagiura (Springer, New York, 2005), pp. 29–63
99. M.G.C. Resende, K.G. Ramakrishnan, Z. Drezner, Computing lower bounds for the quadratic assignment problem with an interior point algorithm for linear programming. Oper. Res. **43**(5), 781–791 (1995)
100. A. Rodrigues-vazqquez, R. Dominguez-castro, A. Rueda, J.L. Huertas, E. Sanchez-sinencio, Nonlinear switched-capacitor Neural networks for optimization problems. IEEE Trans. Circuits Syst. **37**(3), 384–398 (1990)
101. F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain. Psychol. Rev. **65**(6), 386–408 (1958)
102. S. Salcedo-Sanz, C. Bousoño-Calzón, A.R. Figueiras-Vidal, A mixed neural-genetic algorithm for the broadcast scheduling problem. IEEE Trans. Wirel. Commun. **2**(2), 277–283 (2003)
103. S. Salcedo-Sanz, R. Santiago-Mozos, C. Bousono-Calzon, A hybrid Hopfield network-simulated annealing approach for frequency assignment in satellite communications systems. IEEE Trans. Syst. Man, Cybern. Part B Cybern. **34**(2), 1108–1116 (2004)
104. R. Sharda, Neural networks for the MS/OR analyst: an application bibliography. Interfaces **24**(2), 116–130 (1994)
105. Y.J. Shen, M.S. Wang, Broadcast scheduling in wireless sensor networks using fuzzy Hopfield neural network. Expert Syst. Appl. **34**(2), 900–907 (2008)
106. Y. Shrivastava, S. Dasgupta, S.M. Reddy, Guaranteed convergence in a class of Hopfield networks. IEEE Trans. Neural Netw. **3**(6), 951–961 (2002)
107. K.A. Smith, Neural networks for combinatorial optimization: a review of more than a decade of research. INFORMS J. Comput. **11**, 15–34 (1999)
108. K. Smith, Solving the generalised quadratic assignment problem using a self-organising process, in *IEEE International Conference on Neural Networks, 1995. Proceedings*, Perth, vol. 4 (IEEE, Piscataway, 2002), pp. 1876–1879
109. K. Smith, M. Palaniswami, M. Krishnamoorthy, A hybrid neural approach to combinatorial optimization. Comput. Oper. Res. **23**(6), 597–610 (1996)
110. K. Smith, M. Palaniswami, M. Krishnamoorthy, Neural techniques for combinatorial optimization with applications. IEEE Trans. Neural Netw. **9**(6), 1301–1318 (2002)
111. K.A. Smith, J.Y. Potvin, T. Kwok, Neural network models for combinatorial optimization: a survey of deterministic, stochastic and chaotic approaches. Control Cybern. **31**(2), 183–216 (2002)
112. T.E. Stern, *Theory of Nonlinear Networks and Systems* (Addison-Wesley, Reading, 1965)
113. M. Sun, L. Zhao, W. Cao, Y. Xu, X. Dai, X. Wang, Novel hysteretic noisy chaotic neural network for broadcast scheduling problems in packet radio networks. IEEE Trans. Neural Netw. **21**(9), 1422–1433 (2010)
114. H. Szu, R. Hartley, Fast simulated annealing* 1. Phys. Lett. A **122**(3–4), 157–162 (1987)
115. E.D. Taillard, L.M. Gambardella, M. Gendreau, J.Y. Potvin, Adaptive memory programming: A unified view of metaheuristics. Eur. J. Oper. Res. **135**(1), 1–16 (2001)
116. Y. Takefuji, H. Szu, Design of parallel distributed cauchy machines, in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, 1989. IJCNN*, Washington, DC, vol. 1 (IEEE, New York, 1989), pp. 529–532
117. P.M. Talaván, J. Yáñez, Parameter setting of the Hopfield network applied to TSP. Neural Netw. **15**(3), 363–373 (2002)
118. P.M. Talaván, J. Yáñez, The graph coloring problem: a neuronal network approach. Eur. J. Oper. Res. **191**(1), 100–111 (2008)
119. K.C. Tan, H. Tang, S.S. Ge, On parameter settings of Hopfield networks applied to traveling salesman problems. IEEE Trans. Circuits Syst. I Regul. Pap. **52**(5), 994–1002 (2005)
120. B.B. Tan, S.L. Shen, Z.H. Wu, L.P. Wang, Application of neural networks in project optimization of water system management in mining subsidence. J. Anhui Univ. Sci. Technology (Nat. Sci.) **29**(2), 13–16 (2009)

121. D.W. Tank, J.J. Hopfield, Simple neural optimization networks: an A/D converter, signal decision circuit, and a linear programming circuit. IEEE Trans. Circuits Syst. **33**(5), 533–541 (1986)
122. L.N. Trefethen, *The Definition of Numerical Analysis* (Cornell University, Ithaca, 1992)
123. D.E. Van den Bout, T.K. Miller, Improving the performance of the Hopfield-Tank neural network through normalization and annealing. Biol. Cybern. **62**(2), 129–139 (1989)
124. L. Wang, K. Smith, On chaotic simulated annealing. IEEE Trans. Neural Netw. **9**(4), 716–718 (2002)
125. L. Wang, S. Li, F. Tian, X. Fu, A noisy chaotic neural network for solving combinatorial optimization problems: Stochastic chaotic simulated annealing. IEEE Trans. Syst. Man Cybern. Part B Cybern. **34**(5), 2119–2125 (2004)
126. Y. Watanabe, N. Mizuguchi, Y. Fujii, Solving optimization problems by using a Hopfield neural network and genetic algorithm combination. Syst. Comput. Jpn **29**(10), 68–74 (1998)
127. P.B. Watta, M.H. Hassoun, A coupled gradient network approach for static and temporal mixed-integer optimization. IEEE Trans. Neural Netw. **7**(3), 578–593 (2002)
128. U.P. Wen, K.M. Lan, H.S. Shih, A review of Hopfield neural networks for solving mathematical programming problems. Eur. J. Oper. Res. **198**(3), 675–687 (2009)
129. Y.L. Weng, C.Y. Lee, Z.J. Lee, Incorporating Hopfield neural networks into ant colony system for traveling salesman problem, in *New Advances in Intelligent Decision Technologies* (Springer, Berlin/Heidelberg), pp. 287–294 (2009)
130. D.J. Willshaw, C. Von Der Malsburg, A marker induction mechanism for the establishment of ordered neural mappings: its application to the retinotectal problem. Philos. Trans. R. Soc. Lond. Ser. B Biol. Sci. **287**(1021), 203–243 (1979)
131. G. Wilson, Quadratic programming analogs. IEEE Trans. Circuits Syst. **33**(9), 907–911 (1986)
132. G.V. Wilson, G.S. Pawley, On the stability of the travelling salesman problem algorithm of Hopfield and Tank. Biol. Cybern. **58**(1), 63–70 (1988)
133. S.J. Wright, *Primal-Dual Interior-Point Methods* (Society for Industrial Mathematics, Philadelphia, 1997)
134. X.Y. Wu, Y.S. Xia, J. Li, W.K. Chen, A high-performance neural network for solving linear and quadratic programming problems. IEEE Trans. Neural Netw. **7**(3), 643–651 (2002)
135. Y. Xia, A new neural network for solving linear and quadratic programming problems. IEEE Trans. Neural Netw. **7**(6), 1544–1548 (2002)
136. Y. Xia, J. Wang, A recurrent neural network for solving linear projection equations. Neural Netw. **13**(3), 337–350 (2000)
137. S.H. Zak, V. Upatising, W.E. Lillo, S. Hui, A dynamical systems approach to solving linear programming problems. Lect. Notes Pure Appl. Math. **152**, 913–913 (1993)
138. S.H. Zak, V. Upatising, S. Hui, Solving linear programming problems with neural networks: a comparative study. IEEE Trans. Neural Netw. **6**(1), 94–104 (1995)
139. S. Zhang, A.G. Constantinides, Lagrange programming neural networks. IEEE Trans. Circuits Syst. II Analog Digit. Signal Process. **39**(7), 441–452 (1992)
140. M. Zhang, N.B. Liao, C. Zhou, Neural networks model for optimization an study on isomorphism discernment. Adv. Mater. Res. **156**, 492–495 (2011)
141. P. Zheng, W. Tang, J. Zhang, A simple method for designing efficient small-world neural networks. Neural Netw. **23**(2), 155–159 (2010)