MODIFIED HOPFIELD NETWORKS FOR INTEGRATED PROCESS PLANNING
AND SCHEDULING

A Thesis
Submitted to the Faculty of Graduate Studies and Research
In Partial Fulfillment of the Requirements
For the Degree of

Doctor of Philosophy
in
Industrial Systems Engineering
University of Regina

by
Suriya Jirasatitsin

Regina, Saskatchewan
February, 2018

Copyright 2018: S. Jirasatitsin
Suriya Jirasatitsin, candidate for the degree of Doctor of Philosophy in Industrial Systems Engineering, has presented a thesis titled, *Modified Hopfield Networks for Integrated Process Planning and Scheduling*, in an oral examination held on October 5, 2017. The following committee members have found the thesis acceptable in form and content, and that the candidate demonstrated satisfactory knowledge of the subject material.

External Examiner:  
*Dr. Simon X. Yang, University of Guelph*

Supervisor:  
Dr. Rene Mayorga, Industrial Systems Engineering

Committee Member:  
Dr. Wei Peng, Industrial Systems Engineering

Committee Member:  
Dr. Amr Henni, Industrial Systems Engineering

Committee Member:  
**Dr. Malek Mouhoub, Department of Computer Science**

Chair of Defense:  
Dr. Karen Meagher, Faculty of Graduate Studies & Research

*via SKYPE
**Not present at defense
ABSTRACT

In production systems, process planning and scheduling are the two core functions that must be implemented before starting a production. The process planning generates a suitable process plan for each part that indicates the sequences of machining operations of the part, and the suitable machines and tools for that operations. This plan depends on the available manufacturing resources and their operation time. The scheduling, then, specifies the timetable of each operation from the process plans’ information. A significant conflict during the scheduling process is that some determined resources are not valid or overloaded because the process planning intentionally specifies the machines that make the least manufacturing cost. Therefore, it is time-consuming to re-schedule or search for a feasible solution that agree to the constraints of both process planning and scheduling.

Integrated process planning and scheduling is a trend of modern manufacturing systems that would improve production in more flexible ways and overcome the disadvantages of conventional procedures of process planning and scheduling. It is mostly solved the integrated approach by using various metaheuristics algorithms. Hopfield networks, in literature, show effective results in scheduling and can be implemented in hardware level that has the computation powers and speed over the digital computations. Otherwise, the analog computation has no iteration and obtain a real-time solution. However, the Hopfield network is a local minimization method with slow convergence by its gradient descent algorithm. Some modifications of Hopfield network should lead to a fast convergence and global optimization problem.

This thesis presents two original and novel Hopfield neural networks architectures. One of the proposed Hopfield neural networks modifies the standard Hopfield Network by
properly incorporating the Broyden-Fletcher-Goldfarb-Shanno (BFGS) unconstrained optimization algorithm. The second proposed Hopfield network is based on properly incorporating the Nazareth unconstrained optimization algorithm in the standard Hopfield network. One important issue to be investigated is the convergence speed of these two novel Artificial Networks with respect to the convergence of the standard Artificial Hopfield Network. The proposed model uses analog integrators to follow the traditional algorithms of BFGS and Nazareth algorithms. The algorithms can be modeled by using simple analog circuits, such as integrators, multipliers, and adders. The proposed algorithms show a significant increase in the convergence speed of modified Hopfield networks compared to the standard one. Also noise and adaptive learning parameter technique are integrated to the modified Hopfield networks for global optimization. To test and compare the effectiveness of the proposed Networks, they are implemented on the solution of an integrated process planning and scheduling (IPPS) problem, which is a non-deterministic polynomial time problem (NP-hard). Moreover, here, it is treated under a novel formulation and to represent a modern manufacturing approach.
ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my advisor, Professor Rene V. Mayorga, for his excellent guidance and suggestions in the thesis developments. Without his meaningful encouragement and supervision, this thesis would not have been possible.

I would like to extend my deepest thanks to my external examiner and committee members for their insightful suggestions in improving the dissertation.

I gratefully acknowledge the Faculty of Graduate Studies and Research and the Faculty of Engineering and Applied Science at the University of Regina for providing research awards, and teaching assistantships, during my study.

I would like to express my appreciation to the Ministry of Science and Technology in Thailand for the scholarships, and to the Office of Educational Affairs, Royal Thai Embassy, Washington DC, USA, and Royal Thai Embassy, Ottawa, Canada, for all supports and official documentation. I would also like to acknowledge Prince of Songkla University in Thailand for granting me study leave and the financial supports. Many thanks also go to all faculty and staff at the Faculty of Engineering, Prince of Songkla University, Thailand, who encouraged and gave me counsel to study aboard.

Finally, I would like to thank my parents, elder brothers, and friends for their encouragement and moral support which made my studies in Regina, Saskatchewan, Canada more pleasurable.
TABLE OF CONTENTS

ABSTRACT..................................................................................................................i

ACKNOWLEDGEMENTS ............................................................................................iii

TABLE OF CONTENTS ...............................................................................................iv

LIST OF TABLES ..........................................................................................................vii

LIST OF FIGURES .......................................................................................................viii

CHAPTER 1 INTRODUCTION .......................................................................................1

1.1 Motivation .............................................................................................................1

1.2 Objective of the Research .....................................................................................4

1.3 Organization of the Thesis ....................................................................................4

CHAPTER 2 LITERATURE REVIEW ..........................................................................6

2.1 Hopfield Neural Networks ....................................................................................6

2.2 Integration of Process Planning and Scheduling ................................................10

2.2.1 Nonlinear process planning (NLPP) ...............................................................11

2.2.2 Closed loop process planning .......................................................................11

2.2.3 Distributed process planning ........................................................................13

2.3 Process Planning and Scheduling with Artificial Neural Networks .................16

2.4 Global Optimization by Hopfield Neural Networks ...........................................22

CHAPTER 3 BFGS HOPFIELD NEURAL NETWORK .............................................29

3.1 The BFGS Approach ..........................................................................................29

3.2 Modified Hopfield Neural Networks by BFGS methods ...................................32

3.3 Simulation Results ..............................................................................................41

3.3.1 Two-dimensional function ..........................................................................41
LIST OF TABLES

Table 5.1 The solutions and convergence times for the modified HNN for a simple linear programming ..................................................94

Table 5.2 The solutions and convergence times of the modified Hopfield Neural Networks for The Rosen-Suzuki problem .................................................99

Table 7.1 Order related-information of problem 1 (Lee & Dicesare, 1992) ..............150

Table 7.2 The results of starting time variables for problem 1 .................................157

Table 7.3 The results of resource selection variables for problem 1 .........................158

Table 7.4 Job related-information and alternative operation sequences of example 2 .........................................................................................162

Table 7.5 The results of starting time variables for problem 2 .................................172

Table 7.6 The results of resource selection variables for problem 1 .........................173
LIST OF FIGURES

Figure 2.1 The structure of the Hopfield Neural Network …………………….7

Figure 3.1 An architecture of the general dynamic equation ………………….36

Figure 3.2 The block diagram for the proposed BFGS Hopfield neural network …37

Figure 3.3 A block diagram of inverse Hessian matrix updating ………………….38

Figure 3.4 A block diagram of \( y^* \) ………………………………………………….39

Figure 3.5 A block diagram for \( t_k \) …………………………………………………..40

Figure 3.6. The Simulink model of HNN ………………………………………….43

Figure 3.7 The Simulink model of the modified HNN ………………………….44

Figure 3.8 The Simulink model of H_updating block ……………………………45

Figure 3.9 The trajectories \( x(t) \) of HNN for 2-dimensional Rosenbrock function …46

Figure 3.10 The trajectories \( x(t) \) of the modified HNN for 2-dimensional Rosenbrock function ………………………………………………………….47

Figure 3.11 The values of BFGS Hessian matrix for 2-dimensional Rosenbrock function ………………………………………………………….48

Figure 3.12 The trajectories \( x(t) \) of HNN for Powell function ……………………50

Figure 3.13 The trajectories \( x(t) \) of BFGS HNN for Powell function ………….51

Figure 3.14 The values of BFGS Hessian matrix for Powell Function ………….52

Figure 3.15 The trajectories \( x(t) \) of HNN for 10-dimensional Rosenbrock function …54

Figure 3.16 The trajectories \( x(t) \) of BFGS HNN for 10-dimensional Rosenbrock function ………………………………………………………….55

Figure 3.17 The values of BFGS Hessian matrix for 10-dimensional Rosenbrock function ……………………………………………………...56
Figure 4.1 A block diagram of the modified HNN by Nazareth direction ...............69
Figure 4.2 A block diagram of the Equation (4.34b) ........................................70
Figure 4.3 Simulink model of the Nazareth HNN ...........................................72
Figure 4.4 The Simulink block diagram for updating $\mu_{CG}$ ................................73
Figure 4.5 The trajectory of $x(t)$ when applying Nazareth HNN .........................75
Figure 4.6 The $\mu_{CG}$ updating .................................................................76
Figure 4.7 The trajectory $x(t)$ of Nazareth HNN for Powell function .................78
Figure 4.8 The updating matrix of Nazareth HNN for Powell function .................79
Figure 4.9 The trajectory $x(t)$ of Nazareth HNN for Extended Rosenbrock function ........................................................................................................81
Figure 4.10 The values of updating matrix of Nazareth HNN for Extended Rosennbrook function ........................................................................................................82
Figure 5.1 Simulation result from HNN for a simple linear programming ..........91
Figure 5.2 Simulation result from BFGS HNN for a simple linear programming ....92
Figure 5.3 Simulation result from Nazareth direction for a simple linear programming ........................................................................................................93
Figure 5.4 Simulation result from HNN for The Rosen-Suzuki problem ..........96
Figure 5.5 Simulation result from BFGS HNN for The Rosen-Suzuki problem ....97
Figure 5.6 Simulation result from Nazareth direction for The Rosen-Suzuki problem ..98
Figure 6.1 Peaks function ......................................................................................98
Figure 6.2 The path of neuron’s states for the initial condition $x_1(0) = 0$ and $x_2(0) = 2$ .................................................................................................110
Figure 6.3 The trajectory of neuron’s states from HNN for Peaks function with $x_1(0) = 0$ and $x_2(0) = 2$ .........................................................................................111
Figure 6.4 The trajectory of neuron’s states from BFGS HNN for Peaks function with 
\( x_1(0) = 0 \) and \( x_2(0) = 2 \) .................................................................112

Figure 6.5 The trajectory of neuron’s states from Nazareth HNN for Peaks function with 
\( x_1(0) = 0 \) and \( x_2(0) = 2 \) .................................................................113

Figure 6.6 The graph of Griewank function .............................................115

Figure 6.7 The path of neuron’s states along the Griewank function ..........116

Figure 6.8 The trajectory of neuron’s states from HNN for Griewank function with 
\( x_1(0) = 50 \) and \( x_2(0) = 50 \) .................................................................117

Figure 6.9 The trajectory of neuron’s states from BFGS HNN for Griewank function with 
\( x_1(0) = 50 \) and \( x_2(0) = 50 \) .................................................................118

Figure 6.10 The trajectory of neuron’s states from Nazareth HNN for Griewank function with \( x_1(0) = 50 \) and \( x_2(0) = 50 \) .................................................................119

Figure 7.1 The MILP network for job-shop scheduling (Foo et al., 1994) .............124

Figure 7.2 The trajectories of starting time neurons for problem 1 using HNN ....151

Figure 7.3 The trajectories of starting time neurons for problem 1 using BFGS HNN ....152

Figure 7.4 The trajectories of starting time neurons for problem 1 using Nazareth HNN ........................................................................153

Figure 7.5 The trajectories of resource selection variables for problems 1 using HNN ........................................................................154

Figure 7.6 The trajectories of resource selection variables for problem 1 using BFGS HNN ........................................................................155

Figure 7.7 The trajectories of resource selection variables for problem 1 using Nazareth HNN ........................................................................156

Figure 7.8 Gantt Chart of problem 1 ..........................................................159
Figure 7.9 The trajectories of starting time neurons for problem 2 using HNN ........163
Figure 7.10 The trajectories of starting time neurons for problem 2 using BFGS HNN...164
Figure 7.11 The trajectories of starting time neurons for problem 2 using Nazareth
HNN.................................................................................................................165
Figure 7.12 The trajectories of resource selection variables for problems 2 using
HNN.................................................................................................................166
Figure 7.13 The trajectories of resource selection variables for problem 2 using BFGS
HNN .................................................................................................................167
Figure 7.14 The trajectories of resource selection variables for problem 2 using Nazareth
HNN .................................................................................................................168
Figure 7.15 The trajectories of alternative process plan selection variables of problem 2
using HNN .............................................................................................................169
Figure 7.16 The trajectories of alternative process plan selection variables of problem 2
using BFGS HNN ............................................................................................170
Figure 7.17 The trajectories of alternative process plan selection variables of problem 2
using Nazareth HNN ........................................................................................171
Figure 7.18 Gantt Chart of problem 2 using HNN.................................................174
Figure 7.19 Gantt Chart of problem 2 using BFGS HNN.......................................175
Figure 7.20 Gantt Chart of problem 2 using Nazareth HNN.................................176
Figure 7.21 The trajectories of starting time variables of problem 2 using BFGS NNN and
the global optimization ....................................................................................178
Figure 7.22 The trajectories of starting time variables of problem 2 using Nazareth NNN
and the global optimization ............................................................................179
Figure 7.23  The Gantt Chart of the global BFGS HNN for Example 2…………………180
Figure 7.24  The Gantt Chart of the global Nazareth HNN for Example 2……………181
CHAPTER 1
INTRODUCTION

1.1 Motivation

Artificial Neural Network (ANN) has been developed and applied to many science and engineering areas. One important application is an efficient method for solving non-deterministic polynomial-time hard (NP-hard) optimization problems (Akyol & Bayhan, 2007). This ANN-based method can predict and classify a complicated relationship of the system with less time consumption. It can be implemented in dynamic systems and compared to other mathematical modeling techniques suitable for static systems. The Hopfield Neural Networks (HNN), a concept of ANN, have also demonstrated a practical ability to solve optimization problems and their mathematical models. HNN can optimize problems with or without constraints. Moreover, HNN can be simulated efficiently on a digital computer and even be constructed by an analog circuit with a parallel distributed process. This characteristic leads to a significantly rapid computation (Galan-Marin & Munoz-Perez, 2001).

In general, HNNs implement a gradient descent technique in the form of a dynamic system. The technique is the primary approach for solving an optimization problem. Some studies modify the conventional HNN to construct a conjugate-gradient Hopfield Network (Zhang, 2000). The modified Hopfield network has better performance compared to the standard HNN. However, it is well known in the optimization field that the conjugate gradient method takes more computation time than the Newton’s or the quasi-Newton methods. Currently, the most popular and efficient quasi-Newton method is Broyden-
Fletcher-Goldfarb-Shanno (BFGS) method. Therefore, a Hopfield network based on the BFGS method would give the best performance in computation speed. An important issue to note is that a BFGS requires more parameters than a conjugate gradient method. Hence, although very effective, BFGS consumes more memory in computation storage than a conjugate gradient method. Nazareth algorithm can be implemented to overcome the computation memories and computation speed (Nazareth, 1979). Therefore, HNN based on the Nazareth algorithm expects to improve computation speed and computation memory utilization on optimization problems.

Process planning and scheduling are the principal manufacturing activities taken into consideration before starting any production. A process planning focuses on how a product would be produced following the product design specifications; while the main objective of scheduling is to allocate available manufacturing resources into the operations. Conventionally, these two activities are performed sequentially. First, the process plan must be prepared, and then scheduling will assign the resources according to the operations described in the process plan. Process plans and schedules may become incompatible and infeasible to perform since the real-time status of the manufacturing resources is not considered in either process planning or scheduling. Consequently, the initial process plan must change its operations or machine tools because the resources in the shop floor are unavailable. Additionally, the goal of these two processes is usually different in several ways. For process planning, process planners consider the operation cost and performance as basic. On the other hand, production schedulers plan the production by minimizing the makespan and production cost. That is, the selected machines or resources for process plans may be not desired in scheduling.
Combining both the process planning and scheduling seems to increase the feasibility and optimality of production. Recent studies propose that making use of alternative process plans as inputs of scheduling is then a possible way to select operation sequences of the parts according to the availability of the manufacturing resources and the current status of the manufacturing system. That is, the scheduling objectives such as minimizing the makespan, minimizing the tardiness, maximizing equipment utilization, or minimizing production cost may be improved in most cases. Many researchers have reported that common scheduling objectives can be significantly enhanced when properly integrate process planning and scheduling functions. However, there are some limitations of their approaches (Tan & Khoshnevis, 2000). For example, the time window method cannot estimate the effect of each assignment to the overall results but is hard to find the appropriate window-size. The agent-based approach and simulated annealing approach take time to search if the search space is enormous. The decision matrix is tough to assign the interactions between objects within criteria. The petri-net method gives a weak result when the process planning is non-linear. The hierarchical mathematical programming does not guarantee the global optimization because it uses two decision makings. Since both process planning and scheduling are NP-hard problems, the integrated process planning and scheduling (IPPS) is also an NP-hard problem or even strongly NP-hard problem in some cases. Then, recent researches trend to develop hybrid algorithm combining heuristics and meta-heuristics techniques to optimize IPPS in large scale problems.

This thesis proposes two novel Artificial Neural Networks: the BFGS Hopfield Network and Nazareth Hopfield Network. These proposed networks will be tested on the solution of a novel formulation of the IPPS problem. Particular attention will be dedicated
to their computation speed and performances on the proposed novel formulation of the IPPS.

### 1.2 Objective of the Research

These main aims of the thesis are to modify the traditional Hopfield neural network with the BFGS method and Nazareth algorithm and to implement the modified Hopfield neural network to the integrated process planning and scheduling problem. The performance of integrated process planning and scheduling is the total production time (makespan). The optimization of integrated process planning and scheduling is to minimize the makespan. Therefore, the objectives of the research are:

1.2.1 To construct two novel Artificial Neural Networks by using Hopfield neural network, BFGS, and Nazareth algorithm.

1.2.2 To model an integrated Process Planning and Scheduling for job-shop production by constrained optimization that suitable for the modified Hopfield neural networks.

1.2.3 To optimize the integrated process planning and scheduling model based on the makespan minimization using the modified Hopfield neural networks.

1.2.4 To compare performance and computation speed among modified Hopfield neural network and traditional Hopfield neural network.

### 1.3 Organization of the Thesis

The dissertation is organized into eight chapters: introduction, literature review, BFGS Hopfield neural network, Nazareth Hopfield neural network, constrained
optimization, global optimization, integrated process planning and scheduling, and conclusions. In Chapter 2, a comprehensive literature review provides a fundamental concept and previous studies of HNN, IPPS, process planning, and scheduling optimization by using artificial neural network techniques, and Hopfield neural network in global optimization. Chapter 3 will introduce the BFGS methods and describe how to modify the conventional Hopfield neural network with the BFGS updating. Then, the modified Hopfield neural network by using Nazareth algorithm is proposed in Chapter 4. Optimization with constraints is presented in Chapter 5. This Chapter explains a penalty function method for solving a constrained optimization problem. The method has been combined with the BFGS and Nazareth Hopfield neural networks. In Chapter 6, the modified Hopfield neural networks combined with Gaussian noise are used to solve global optimization problems. Chapter 7 focus on the integrated process planning and scheduling modeling and implementation with the modified Hopfield neural networks. Finally, the conclusions and future research recommendations are proposed in Chapter 8.
2.1 Hopfield Neural Networks

Hopfield Neural Network (HNN) is a highly-interconnected network of nonlinear analog processing elements, called neurons. HNN is one of the Artificial Neural Networks (ANNs) that mimics the human brain’s neurons by using electronic devices such as operational amplifiers as human neurons, or resistors as human synapses, and capacitors as human memories. HNN circuit consists of amplifiers, resistors, and capacitors to perform feedback neural networks. The structure of HNN is shown in Figure 2.1. The triangles represent neurons constructing by sigmoid amplifier circuits. The neurons have input voltage $u_1, u_2, \ldots, u_n$, output voltage $V_1, V_2, \ldots, V_n$, and external input current $I_1, I_2, \ldots, I_n$. The external inputs can be constant biases for shifting the input-output relation or specific input currents representing data in the problem. Each neuron is connected by conductance $T_{ij}$, performing as synapses in human neuron system. Input resistor $\rho_i$ and input capacitor $C_i$ provide the integrative analog summation of all input currents. The amplifiers have sigmoid monotonic input-output relations with the function $V_i = g_i(u_i)$. By using Kirchhoff’s law, the dynamic equation of input $u_i$ is

$$C_i \left( \frac{du_i}{dt} \right) = \sum_{j=1}^{N} T_{ij} V_j - \frac{u_i}{R_i} + I_i$$

(2.1)

where $R_i$ is a parallel combination of $\rho_i$ and $1/|T_{ij}|$. 


Figure 2.1 The structure of the Hopfield Neural Network.
For simplicity, the dynamic equation of input $u_i$ can be expressed as
\[
\frac{du_i}{dt} = -\frac{u_i}{\tau} + \sum_{j=1}^{N} T_{ij} V_j + I_i
\] (2.2)

where $\tau = R_i C_i$ or $RC$ if all $R_i = R$ and $C_i = C$, and redefine $\frac{T_{ij}}{C} = T_{ij}$, and $\frac{I_{ij}}{C} = I_{ij}$.

The energy function of the network, which is a Lyapunov function for the system, is defined as
\[
E = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} T_{ij} V_i V_j - \sum_{i=1}^{N} I_i V_i + \frac{1}{\tau} \sum_{i=1}^{N} \int_{0}^{V_i} g_i^{-1}(v)dv
\] (2.3)

where $g_i^{-1}(V)$ is the inverse of output $V$. The time derivative of $E$ when $T$ is symmetric is
\[
\frac{dE}{dt} = \sum_{i=1}^{N} \frac{\partial E}{\partial V_i} \frac{dV_i}{dt} = -\sum_{i=1}^{N} \left( \sum_{j=1}^{N} T_{ij} V_j + I_i - \frac{u_i}{\tau} \right) \frac{dV_i}{dt}
\] (2.4)

By substituting Equation (2.2) into Equation (2.4), we define the time derivative of $E$ as
\[
\frac{dE}{dt} = -\sum_{i=1}^{N} \frac{du_i}{dt} \frac{dV_i}{dt}
\] (2.5)

Since $u_i = g_i^{-1}(V_i)$, $\frac{du_i}{dt} = g_i^{-1}(V_i) \frac{dV_i}{dt}$

Thus,
\[
\frac{dE}{dt} = -\sum_{i=1}^{N} g_i^{-1}(V_i) \frac{dV_i}{dt} \frac{dV_i}{dt} = -\sum_{i=1}^{N} g_i^{-1}(V_i) \left( \frac{dV_i}{dt} \right)^2
\] (2.6)

and since $g_i^{-1}(V_i)$ is a monotone increasing function.

\[
\frac{dE}{dt} \leq 0, \quad \frac{dE}{dt} = 0 \rightarrow \frac{dV_i}{dt} = 0 \text{ for all } i
\] (2.7)

Equation (2.7) states that the energy function $E$ always becomes smaller with time and remain at a certain value when the system reaches a local minimum.

From Equation (2.3), we can determine $\frac{\partial E}{\partial V_i}$ as
\[
\frac{\partial E}{\partial V_i} = -\sum_{j=1}^{N} T_{ij} V_j - I_i + \frac{u_i}{\tau}
\]  

(2.8)

Substituting Equation (2.8) into Equation (2.2), we have

\[
\frac{du_i}{dt} = -\frac{\partial E}{\partial V_i}
\]  

(2.9)

Equation (2.9) shows that the time evolution of the input \( u_i \) is the negative gradient of the energy function of the system. Note that HNN is a special type of gradient-based optimization. To solve Equation (2.2), one can use an integration method to simulate the behavior of the system.

With the advanced microelectronic technology, ANNs can be implemented in chips, known as neurochips, neuromorphic, or optical neural networks, with up to \( 10^6 \) neurons and 40,000 synapses (Goser, 1996; Dias, Antunes & Mota, 2004; Misra & Saha, 2010). The neurons will continuously receive data from the massive number of sensors and provide the parallel processing as the human nervous systems do. The main advantages of this analog neural network are the computation powers and speed. The digital computers would use excessive computation time even for low-dimensional problems or not converge for some nonlinear problems (Hopfield & Tank, 1985; Kennedy & Chua, 1987). Additionally, the analog computation has no iteration and obtain a real-time solution. HNN, firstly, performs as a content-addressable memory (Hopfield, 1982; Hopfield, 1984). It is, then, implemented in areas of optimization problems, such as Traveling-Salesman Problem (Hopfield & Tank, 1985), and linear programming problem (Tank & Hopfield, 1986). HNN can, also, be unified as a nonlinear programming circuit by correcting nonlinear activation function (Kennedy & Chua, 1987). It shows a high computation power in nonlinear problems.
After HNN for Traveling-Salesman Problem has been proposed, HNN is implemented to the combinatorial problems, for example, the factory allocation problem (Watta & Hassoun, 1996), a car sequencing problem (Smith, Palaniswami & Krishnamoorthy, 1996), the economic load dispatch problem (Lee, Sode-Yome & Park, 1998), and the task assignment problem (Mohamed, Chakir, Youssef & Khalid, 2012). The modified HNNs can be solved various pattern recognition problems, such as human eye movement mechanisms (Washizawa, 1993), medical image segmentation (Cheng, Lin & Mao, 1996), face recognition (Ricanek, Lebby & Haywood, 1999) and cancer attractors (Szedlak, Paternostro & Piermarocchi, 2014).

For solving the optimization problems by HNN, the weights and external inputs of HNN are determined appropriately to represent the function to be minimized. The energy (Lyapunov) function of a system is the objective function including the penalty terms for the constraints of the optimization problem (Smith, 1999; Shih & Yang, 2002; Wen, Lan & Shih, 2008). HNN, in general, is a linear programming network. If the optimization problems contain high-order terms, the terms can be added as entropy. Then, the derivative of entropy term is added directly to the dynamic equation of the neurons (Jeffrey & Rosner, 1986).

### 2.2 Integration of Process Planning and Scheduling

Many researchers have studied the effective methods to integrate process planning and scheduling (IPPS). IPPS mainly focuses on how to make a process plan with flexibility and feasibility to the manufacturing resources. Then, the process plan is implemented an optimization algorithm to find the most suitable solution depending on the constraints of
production. Three basic process planning model for an IPPS are nonlinear, closed loop, and distributed process planning.

2.2.1 Nonlinear process planning (NLPP)

NLPP generates all alternative process plans with a priority index. Scheduling will choose the highest priority to assigning the available resources at the current time. If this plan is not suitable, scheduling will choose the next priority consecutively.

Leung, Wong, Mak, and Fung (2010) minimized the makespan of integrated process planning and shop-floor scheduling by implementing an ant colony optimization algorithm in an agent-based system. The alternative process plans were generated by an AND/OR graph structure. Their proposed method was less competitive than the negotiation-based multi-agent system because of suitable parameters were not found. A method of design of experiment such as analysis of variance may be applied to conduct parameter controlling.

Wong, Zhang, Wang, and Zhang (2012) developed a two-stage ant colony optimization algorithm. The alternative production processes and sequences are generated by an AND/OR graph. Their optimization was divided into two stages; process selection and process sequencing. The updating strategy of pheromones was modified by adding incremented pheromone values. The modified two-stage ant colony optimization was efficient and effective to IPPS with moderate complexity.

2.2.2 Closed loop process planning

This model of IPPS uses a dynamic process planning system with a feedback mechanism. First, the process plans are created based on available resources.
Next, production scheduling sends the status of available machines on the shop floor to the process plans. Hence, scheduling is conducted by feasible process plans.

Wang et al. (2008) proposed an approach to reduce the tardy jobs. Two heuristic rules are used: an iterative manner and a planning solution space intuitiveness. After a process plan and schedule had been generated, the integrator module modified the schedule by measuring schedule performances. A CAPP module received feedback of generated constraints from the integrator module. A satisfactory schedule was reached when all performance measures were met. JAVA based browser/server technology was used to commute data between the production database and the system. A web-based collaborative system simulated the rules showed that schedule performance levels were satisfied in a fair number of iterations.

Lv, Xu, Zhang, and Xu (2009) implemented the ant system optimization and dynamic scheduling system to minimize the makespan of direct hot charge rolling production (DHCR). The IPPS was formulated as the relationship of material specification, priority of order, due date of order, prize of node, and state of anode. The dynamic scheduling system used a rule-based reasoning engine which received the status of processes from production monitoring system. The proposed system showed that the actual rate was greater than 85% which is feasible and efficient enough for DHCR.

Sormaz, Arumugam, Harihara, Patel, and Neerukonda (2010) used XML to represent data required for scheduling and FMS control. First, features of apart were created by Unigraphics CAD software and saved into an XML file. Next, a rule-based system was used to generate a process selection. Finally, scheduling was performed by
linear programming optimization. The model was powerful in direct distributed applications which XML was main data form of real-time on-line communication.

2.2.3 Distributed process planning

A hierarchical approach is used to perform the process planning and production scheduling simultaneously. First, the primary process plans are determined along with evaluated process resources. Next, the process plans are adjusted to the current status of the shop floor. Therefore, the detailed process plans and scheduling plans are obtained simultaneously.

Zhang, Saravanan, and Fuh (2002) presented a back-propagation approach IPPS for batch manufacturing. The approach was depended on a flexible computer-aided process planning system. The knowledge-based module generated a schedule which is adjusted the available resources by a try-and-adjust manner. They showed that the approach would produce a decent schedule within 5 – 10 iterations.

Chua, Wang, Cai, and Yin (2006) modeled an advanced planning and scheduling (APS) system. The APS system consisted of two layers of heuristic algorithms. Job prioritization (JP) heuristics rules were to prioritize operation orders. Machine selection (MS) algorithm designed the best-fit machines and resources for dispatching the list. For a production dealing with a bottleneck operation, they proposed a bottleneck scheduling engine which performed forward and backward order tree. This model improved machine utilization and cycle time of target product in the discrete manufacturing industry.

Li and McMahon (2007) proposed a simulated annealing-based optimization approach (SA) for IPPS. IPPS was represented as the functions of the
makespan, the manufacturing cost, the balanced level of machine utilization, and the part
tardiness. The SA was performed as 1) Select a satisfied process plan, 2) Set an initial
schedule, 3) Assign the start and end temperatures, i.e. numbers of iteration, 4) If the
current temperatures is greater than the end temperatures, the shift, adjacent swapping, and
mutation strategies are applied to the current schedule and generated a temporary schedule,
5) Adjust the temporary schedule according to the constraints, 6) Determine the difference
between current performance and temporary performance, 7) If the difference is less than
or equal zero, replace the current schedule by the temporary schedule. Otherwise, lower
the temperature and repeat step four again, and 8) If the current temperature is less than or
equal to the end temperature, finish the optimization. The proposed SA showed that it took
a shorter time to find a solution, but it was hard to define the parameters. Comparing to the
genetic algorithm and particle swamp optimization, the SA was faster but less robust.
Mohammadi, Karampourhaghghi, and Samaei (2012) used a mixed integer linear
programming to generate schedules and select the best performance schedule by SA. This
approach gave a better search structure for multi-objective IPPS.

Baykasoglu and Ozbakir (2009) presented a grammatical representation of
general process plans. The representation composed of preparatory tasks, formative tasks,
transitional tasks, terminating tasks, set of non-value adding tasks, and set of value-adding
tasks. No actual resources were not assigned in the generic process. Tabu search algorithm
optimized the generic process plans. Li, Gao, and Li (2012) presented a model for the
multi-objective IPPS. The objectives of this IPPS were minimizing the makespan, the
maximal machine workload, and the total workload of machines. The model used the Nash
equilibrium of game theory to optimize the objectives. The research combined genetic
algorithm and tabu search into a hybrid algorithm. The modified IPPS performed well with the multiple objectives problem and found optimal or near optimal solutions efficiently.

Shao, Li, Gao, and Zhang (2009) applied both alternative process plans and hierarchical approach to take advantages of both methods. The IPPS is optimized by a genetic algorithm based on the current shop floor status. The results showed that the proposed method considered all the conditions synthetically and got better results than other previous research.

Guo, Li, Mileham, and Owen (2009) proposed the development of particle swarm optimization (PSO) for the process planning and scheduling integration. PSO algorithm used to encode initial solutions. The process plan consists of operations defined as particles. The process plans are alternative when there are more than one machining for each operation. They modified the traditional PSO by mutation, crossover and shift steps to overcome the difficulties of computation and the hard to reach a global solution. The results from their case studies showed that PSO is faster in optimization speed and more robust than the genetic algorithm and simulated annealing algorithm when the complexity of problems increases.

Lian, Zhang, Gao, and Li (2012) proposed a new algorithm called imperialist competitive algorithm (ICA) for IPPS. The ICA names the problem parameters as country compare to the chromosome in the GA. Each country has its fitness called power which is the makespan of IPPS. The differences between ICA and GA are 1) all solutions in ICA are kept throughout the optimization process, 2) the crossover in the ICA assign the chromosome of children to mix with parents, 3) the mutation in the ICA applies to children only, and 4) the ICA is a socio-political motivated global search algorithm. Otherwise, the
ICA converges when all countries reach the same position. The researchers showed that the ICA reduced the chances to find local optima and was able to optimize the IPPS more deeply than the GA.

Mohapatra, Benyoucef, and Tiwari (2013) presented adaptive characteristics of a process plan with a cross-machine setup planning. They applied non-dominated sorting algorithm (NSGA-II) to construct the process plans. A fuzzy set was developed to extract one of the Pareto-optimal solutions. The proposed model showed a robust solution to develop optimal adaptive setup plan and multi-objective IPPS. Seker, Erol, and Botsali (2013) optimized the IPPS using the GA and Fuzzy Neural Network (FNN). GA represented the model of IPPS with a clustered chromosome structure and generated schedules. Artificial neural network (ANN) predicted a schedule from obtained schedule when the production conditions were changed. In this model, multilayer perceptron and radial basis functions network were implemented. Nevertheless, the values of the new operation sequences generated by ANN have not been an integer. Then, a fuzzy membership function was applied to classify the input data of ANN. The proposed model generated schedules quickly and accurately.

2.3 Process Planning and Scheduling with Artificial Neural Networks

In recent decades, artificial neural networks (ANNs), a technique in soft computing, plays a major role in optimization problems. Some features of ANNs dominate the production scheduling are listed (Akyol & Bayhan, 2007).
- ANNs can simplify to predict or classify a variety of the complex relationship between the input and output variables that may not set as a mathematical model.

- ANNs use less time consuming than simulation approaches in some cases of manufacturing systems.

- Backpropagation networks generate a schedule from a given set of input parameters for a schedule retrieval system. They can also estimate the system performance measures.

- ANNs can be employed in dynamic scheduling environments unlike the mathematical modeling, dynamic programming, or branch and bound that do well optimization for static scheduling environments.

- Optimizing networks such as Hopfield network and its extended models deal with the optimization. The networks can perform an optimization with or without constraints.

ANNs performs the scheduling into two aspects; schedule retrieving, and schedule optimization. Schedule retrieving uses concepts of supervised ANNs, that is ANNs will be trained with data sets. Schedule optimization has applied some algorithms or unsupervised ANNs, such as Hopfield networks.

Lee and Shaw (2000) performed a neural-net approach to real-time flow-shop sequencing. The neural-net approach utilized the job sequencing knowledge to make real-time decisions. The network consisted of two layer; preceding jobs, and succeeding jobs layer. The weights held desirability of the sequencing between two different jobs. The
networks showed a better quality and less computational time than the deterministic greedy heuristics and the NEH heuristic.

Yang and Wang (2001) proposed a constraint satisfaction adaptive neural network (CSANN). The CSANN combined a heuristic algorithm to produce non-delay schedules for job-shop scheduling problems. CSANN consisted of SC-units and RC-units layer which fed back sequence and resource constraints information to ST-units layer corresponding to the starting times of all operations. The feedback adjusted ST-units to satisfy the constraints. A feasible schedule was shortened each starting time of each operation to the completion time of the previous operation. This approach obtained a non-delay schedule.

Yu and Liang (2001) used GA to optimize machine sequencing and neural network to optimize the operation start times with a fixed sequence. New neurons represented processing restrictions and resolve constraint conflicts. The neural network performed as a gradient search algorithm.

EXTS (Solimanpur, Vrat & Shankar, 2004) was a neuro-tabu heuristic for flow shop scheduling system. Neural networks represented the tabu search processes. The networks provided a facility of tabu moves. Hence, the heuristic reduced the tabu effect of a move exponentially. The EXTS reduced the computation time and improved the makespan.

Wechman, Ganduri, and Koonce (2008) proposed a neural network job-shop scheduler. The scheduler performed in two steps: GA optimal schedules, and ANNs schedule generation. GA optimized job-shop schedules based on machine and processing time. The optimal schedules, generated by GA, were used as the training data for a
backpropagation neural network. The NN scheduler performed well with large problem sizes and provided a close approximation to the performance of a GA scheduler.

El-Bouri, Balakrishnan, and Popplewell (2000) presented a neural network approach for single machine job sequencing. The procedures of sequencing composed of two steps: problem classification, and job sequencing. A classifier network with competitive learning algorithm assorted the problems into ten categories of sequencing. The input for the classifier network was the range of production due date. The sequencing networks were specified based on the classes of problem. The neural sequencers used an adjacent pairwise interchange strategy. 5,000-10,000 data patterns trained the classifier network and the neural sequencers. The model showed effective results with retrieving a schedule but useless for scheduling optimization.

Vidal, Mucientes, Bugarin, and Lama (2011) modified NSGA-II (Non-dominated Sorting Genetic Algorithm II) for optimizing machine scheduling. The modified NSGA-II combined evolutionary computing and neural networks to reduce the huge search space and computational times. The evolutionary computing optimized the makespan and total tardiness of schedules. Otherwise, a multilayer perceptron neural network estimated processing time of each operation. The neural network showed a much better accuracy than the other regression techniques.

Foo and Takefuji (1988b) presented a job-shop scheduling formulation based on Hopfield network. They used a matrix representation corresponding to three constraints; 1) no more than $m$ number of jobs can start at time 0, 2) self-dependency on each operation is not allowed, and 3) precedence relationships between operations must be obeyed. The formulation was optimized by a simulated annealing search to avoid local minima
solutions. The simulated annealing was added a random component to decrease the time-consuming. The approach generated near-optimal solutions and took an infinite amount of solution searching time when the size of the problem was large.

Vaithyanathan and Ignizio (1992) proposed a modified Hopfield network for resource constrained scheduling. First, the constrained problems were decomposed into the multidimensional knapsack mathematic model. Next, the mathematic model was formed as the energy function of Hopfield networks. Their Hopfield networks were added new neurons and two new layers. The layers determined incomplete-consumed resources and adjusted stochastically the solution to avoid local optima. The two-thirds of the generated schedules were optimal, and the rests were near-optimal.

Foo, Takefuji, and Szu (1994) presented an analog circuit for job shop scheduling which is modified from Hopfield and Tank linear programming network. The proposed network solved mixed integer linear programming problem. The researchers appended nonlinear amplifiers representing the zero-one variables to the traditional network. They illustrated the circuit diagram for solving a 2-job 3-machine problem. They concluded that their network had a very simple activation function for each neuron and not required any extensive calculations.

Satake, Morikawa, and Nakamura (1994) modified the Hopfield networks for minimizing the makespan of the job-shop scheduling. They made a non-delay schedule by changing the threshold values at each transition of neurons. They also applied the Boltzmann mechanism to get rid of non-optimal solutions. The proposed model generate all optimal schedules within the specified number of tests.
Willems and Rooda (1994) formulated the job-shop scheduling into an integer linear programming and then translated the formula to a neural network. The integer linear programming represented the scheduling constraints. The neural network was divided into three units connected each other. This model found feasible solutions with less time than compared algorithms and no need of integer adjustments.

Willems and Brandts (1995) used the rules of thumb heuristics as a local optimization criterion for job-shop scheduling. The schedule was modeled in the form of an integer linear programming. The model was separated as units and connected each other to form a neural network. They concluded that the model showed equal performance, but better calculation speed compared to the advanced heuristic sequential schedulers.

Sabuncuoglu and Gurgun (1996) solved the single machine mean tardiness scheduling problem and the minimum makespan of a job shop scheduling problem. The proposed model modified the Hopfield network by adding monitor and control processor. The processor drove the network to converge a final solution. The results showed that the proposed network generated better solution than Wilkerson and Irwin algorithm for the single machine problem.

A Hopfield network was implemented for solving real-time scheduling problems (Silva, Cardeira & Mammeri, 1997). A matrix represented task schedules with a task as row and a unit of time as a column. A mapping method for the proposed networks ensured valid convergence. The researchers found that the objective function, minimized by the networks, was not the same as the Lyapunov function, but it satisfied the problem constraints. They confirmed that the network could find a solution within microseconds when an analog hardware implemented it.
Liansheng, Gang, and Shuchun (2000) implemented a Hopfield neural network for intelligent scheduling model. They derived energy computation functions for many schedule modes including job-shop static scheduling, scheduling with due-date constraint or priority constraint, dynamic scheduling, and Just-In-Time scheduling (JIT). They concluded that the neural network schedule model and algorithm gave a useful method in variants of discrete systems, not only the manufacturing systems.

Wang, Xu, and Wu (2003) proposed a modified permutation matrix of Hopfield neural networks for job shop scheduling problems. The permutation matrix constructed based on the constraints of the scheduling. This approach made the initial setting for the Hopfield networks. They also applied simulated annealing algorithm to the Hopfield networks for solving global optimization of the problem. The developed Hopfield networks made neural networks converge to the minimum of the energy function, and feasible solutions for scheduling.

2.4 Global Optimization by Hopfield Neural Networks

The solution of HNN strictly depends on the initial condition of its gradient descent dynamics that the system will always converge to the nearest local stable point (Hopfield, 1982). This feature comes to the main part of HNN for data collecting and retrieving. However, it does not give the best answer for the optimization problems which need only an optimal solution. Then, some heuristic and algorithms are combined with neural networks to overcome this drawback. The concept of these algorithms is to change the system energy (i.e. the value of cost function or objective function) when HNN falls into local extrema. The developed HNNs, for example, are based on stochastic and simulated
annealing algorithms (SSA), tabu search, smoothing algorithms, genetic algorithms, and particle swarm optimization (Duch & Korczak, 1998).

The very first SSA is proposed as a Boltzmann machine (BM) (Hinton, Sejnowski & Ackley, 1984). This model of neural network decides the neurons’ states transition by adopting the Metropolis algorithm. Consequently, the network will reach a global or near-global state following the relative probability of the Boltzmann distribution. In practical, BM converges to a global in relatively slow manner because of its enormous computational complexity (Takefuji & Szu, 1989; Biro, Kornkai & Tron, 1996). Gaussian machine (GM) and Cauchy machine (CM) improve the function of neuron’s outputs followed the Gaussian white noise distribution, and Cauchy color noise distribution, respectively, and introduce the cooling schedule in inversely linear function which is faster than BM’s cooling schedule (Szu, 1986). The machines obtain the hill-climbing capability from BM and increase the convergence speed of HNN. CM preserves the local random walks and global random flights, that occasionally long jump in expediting the state space searching, while GM produces only fast local random walks. The very long jumps of CM overcome the very slow convergence of BM. Instead of using stochastic activation function like BM, diffusion machine (DM) adds noise directly to the HNN dynamic system and possibly implements in an electrical circuit (Wong, 1991; Kesidis, 1995). DM, theoretically, is a combination of the Langevin algorithm and HNN. The Langevin algorithm is based on the dynamic of molecular systems that have effects of viscosity which state that the particles move randomly as Brownian motion, i.e. there is noise in the system perturbing the movement of particles (Gidas, 1985). In DM, the noise is operated as independent Wiener processes or White Gaussian noise which its magnitude or intensity is controlled by a parameter,
called temperature. However, the noise can destroy the behavior of the original dynamic system of HNN. Then an annealing temperature, in the principal of simulated annealing algorithm (SA), is used to control and eliminate the noise when a global minimum is found. SA will gradually decrease the temperature from a sufficiently large to zero. It allows the system to quickly escape from some local minima by large random fluctuations in earlier iterations and finally backs to the original dynamic model when the temperature is in very small value (Geman & Hwang, 1986). With the flourishing results, many researchers have extended the diffusion machine by improving its simulated annealing algorithm to accurate the optimal solution (Biro et al., 1996; Chen, Zhang & Nee, 1998; Liang, R., 2000; Singh & Dixit, 2013). These models define their annealing schedule in the form of inverse logarithmic or linear cooling schedules. However, in term of hardware implementation, linear cooling schedules are easy to build compared to inverse logarithmic ones (Mandziuk, 2000). Regardless of the cooling schedules, SSAs significantly converge to an optimal solution depending on the initial temperature of cooling schedules. If it is not suitable tuned, one cannot find a suitable result or uses more time to reach the optimal point (Ingber, 1993). Until now, the value of the initial temperature is always set by trial and error. One suggests that the starting amplitude of noises should be about 8-10 times larger than the possible largest derivatives of energy function (Biro et al., 1996). One starts with a very large noise intensity to assure that the neuron’s states can escape from a local minimum. Then, the intensity is decreased, if the new states behave as Brownian motion, or increased if the state’s trajectory moves around a point (G. Yin & K. Yin, 2006). The advantages of DM over BM are that DM can perform training, learning, and weight-adjustment simultaneously that BM does not (Wong, 1991). DM uses less computation time compared
to BM, for example, 7 hours of BM but 8 minutes of DM in segmenting magnetic resonance image of the human brain (Sammouda, Niki & Nishitani, 1996). Unlike the DM, Local minima escape (LME) algorithm randomly disturbs the connections and the biases of HNN (Peng, Gupta & Armitage, 1996; Martin-Valdivia, Ruiz-Sepulveda & Triguero-Ruiz, 2000). LME-HNN converge to a new stable state from a local minimum by changing the value of the connections and the biases of HNN. If the new state has a lower energy level than the previous state, the new state will be used as the initial state of LME. If not, the initial state of LME is back to the previous local minimum. This technique uses much less time but a better result than SA. In contrast to stochastic noise, the HNN’s noise can, also, be defined in a deterministic model. This extension of HNN, called chaotic neural network (CNN), is introduced to overcome the disadvantages of SSA. The CNN modifies the activation function of neurons in chaotic dynamics approach (Aihara, Takabe & Toyoda, 1990). The convergent processing of CSA is bifurcation structures. The searching spaces of CSA are only possibly subspaces of neuron’s states, not all possible states when using SSA. Thus, state space of CSA is smaller than SSA. Otherwise, the deterministic dynamics of CSA limit the searching region to contain some optimal or near-optimal solutions (Chen & Aihara, 1995). The chaotic dynamical systems are implemented in parameter controlling of household appliances such as fan heaters, dishwashers, air conditioners, and microwave ovens (Aihara, 2002).

Smoothing algorithms transform the energy function to a simple function that has few extreme and some smoothed functions with fine-tuned parameters may leave only the global extrema. One of the possible methods is to intentionally adjust the weights and thresholds of HNN to increase the energy at a local minimum (Tang, Jin, Murao, Ishizuka
The weights and thresholds are changed in the positive gradient direction respect to that parameters until they go to the largest derivative. This algorithm will gradually increase the energy so that the states of HNN are changed to a local maximum. Consequently, the HNN will repeat the energy minimization from that local maximum and then do the hill-climbing algorithm until it finds a global minimum. In contrast to use positive gradient direction to adjust the weights of HNN, these parameters may change in negative gradient direction to do the energy function decreasing (R. Li, Qiao, & W. Li, 2016). The connection weights are adjusted by a negative learning rate, some adjustment parameters, and output of connected neurons. These two algorithms are suitable for a model with well-defined weight and threshold parameters. In the case of constrained minimization problem, HNN with constraint terms can find a global minimum if the cost term and constraint terms of the energy function are balanced. The coefficients of cost term and constraint terms are defined to eliminate the local minima that appear in the cost or constraint terms by a deterministic function (Wang, Tang & Cao, 2002), Hopfield-type barrier function with a barrier parameter (Dang, Liang & Yang, 2013), or chaotic parameters disturbance annealing (Wei & Zheng-Ou, 2003). The balanced cost and constraint term techniques approximate the solution to a near-optimal or optimal point quickly and efficiently.

Many researchers use some global optimization methods combine with ANN to find a global solution for a specific problem. HNN can be reformulated in term of the parameter \( \lambda \) for finding extrema of functions and implementing to the inversion of remote sensing (Jeffrey & Rosner, 1986). They noted that the differential form of neurons leads to a decrease in the energy function with positive parameter \( \lambda \) or an increase in the energy
function with negative parameter $\lambda$. Therefore, HNN can find minima or maxima when $\lambda$ is adjusted. The algorithm itself initiates the consecutive starting points for finding extrema. If it keeps tracking the energy function, the lowest or highest energy function would be found easily. The reformulated HNN perform faster in computation time comparing to simulated annealing and conjugate gradient methods. Parameterization of decision variables gives effective near optimal solution for nonlinear global optimization (Mayorga & Arriaga, 2007). The technique is based on a damped least square formulation. Its concept is to keep decreasing the value of the objective function, and finding the corresponding decision variables by approximating its inverse function via trained ANNs. There are four parameters concerning in the algorithm: the decreasing ratio, the singularities avoidance constant, the convexity checking parameter, and the null space vector scaling parameter. This algorithm is harder to tune up compared to simulated annealing and genetic algorithms, but consumes less time in computation and can be used for multi-objective optimization. The combination of Genetic Algorithm (GA) and HNN are processed into two folds. HNN is used to store information of problems, such as objective functions, constraints, or patterns, and initially found a feasible solution. Then, GA uses the HNN’s solution for generating optimal solutions. This kind of algorithm is widely implemented in the combinational problem, such as burnable poison placement (Khoshahval & Fadaei, 2012), pattern recalling (Kumar & Singh, 2012), and frequency assignment problem (Yang, Wu, Jin & Xu, 2016). Other techniques, such as tabu search, and particle swarm optimization, are used to initiate the state of HNN neurons or find optimal or near-optimal solutions from the output of HNN (Konishi, Shimba, Toyama, Kudo & Simbo, 1999; Bastos-Filho, Schuler, Oliveira & Vitorino, 2008). These above
algorithms are effective, but dealing with some parameters that may be difficult to implement into hardware.
CHAPTER 3
BFGS HOPFIELD NEURAL NETWORK

This chapter explains the principle concept and how to modify conventional Hopfield neural network with Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithms. The proposed method realizes the approximation of the inverse Hessian matrix updating as a multiplier of the negative gradient search direction. This multiplier changes its values over time. Therefore, the multiplier would be called an adaptive learning rate of Hopfield dynamic system.

3.1 The BFGS Approach

Considering the unconstrained optimization problems,

$$\min f(x)$$

where \( x \in \mathbb{R}^n \), and \( f(x) \) is a nonlinear function and has a Lipschitz continuous gradient.

The iterative descent method can be considered as

$$x_{k+1} = x_k + \alpha_k d_k \quad (3.1)$$

where \( \alpha_k \) is step size that in the descent direction \( d_k \), and \( k \) is the current iteration number.

The \( x_{k+1} \) is determined and must satisfy the inequality:

$$f(x_{k+1}) = f(x_k + \alpha_k d_k) < f(x_k) \quad (3.2)$$

The optimal step size \( \alpha_k \), usually, is computed by any line search (one-dimensional search) methods, which follow

$$\alpha_k^* = \arg \min_{\alpha_k \geq 0} \phi(\alpha_k) \quad (3.3)$$
where

\[ \phi(\alpha_k) = f(x_k + \alpha_k d_k) \]  

(3.4)

The function \( f(x) \) is approximated to be a quadratic function if \( x_k \) is close to a local minimum. The approximated quadratic function is defined as

\[ f(x_{k+1}) \approx f(x_k) + g^T (x_{k+1} - x_k) + \frac{1}{2} (x_{k+1} - x_k)^T H(x_{k+1} - x_k) \]  

(3.5)

where \( g \) is the first order partial derivatives or the gradient of \( f(x) \), and \( H \) is the second order partial derivatives or the Hessian matrix of \( f(x) \). By differentiating Equation (3.5) with respect to \( x_{k+1} \) and setting it to zero, the minimum point \( x^* \) would be found as

\[ g + H(x^* - x_k) = 0 \Rightarrow x^* = x_k - H^{-1} g \]  

(3.6)

From Equation (3.1) and (3.6), a descent direction is defined as \( d_k = -H^{-1} g \), and called the Newton direction. One disadvantage of Newton direction is that it needs Hessian matrix which is hard to define in practical optimization problems. Broyden (1970), Fletcher (1970), Goldfarb (1970), and Shanno (1970) proposed an approximation of inverse Hessian matrix that retains the positive definite property and less depend on any line search method by using the gradient of \( f'(x) \), and values of parameter \( x \). The formula of the approximated inverse Hessian matrix is

\[ H^{-1}_{k+1} = H_k^{-1} + (1 + \frac{y^T_k H_k^{-1} y_k}{s_k^T y_k}) s_k s_k^T - \left( \frac{s_k y^T_k H_k^{-1} y_k}{s_k^T y_k} \right) \]  

(3.7)

where \( s_k = x_{k+1} - x_k \), and \( y_k = g_{k+1} - g_k \).
\( \mathbf{H}_{k+1}^{-1} \) will remain symmetric positive definite if \( \mathbf{H}_k^{-1} \) is symmetric positive definite and \( s_k^T y_k > 0 \) (Shanno, 1978). This condition can always be satisfied if the line search at each iteration is sufficiently accurate (Gill, Murry & Wright, 1977).

However, the Hessian matrix of nonlinear function may not be a positive definite. Equation (3.7) would not be expected to be a positive definite neither. Li and Fukushima (2001) modified \( y_k \) that always keep \( s_k^T y_k > 0 \). The new \( y_k^* \) is defined by

\[
y_k^* = y_k + t_k \| g_k \| s_k
\]

(3.8)

where \( t_k = 1 + \max \left\{ -\frac{s_k^T y_k}{\| s_k \|^2}, 0 \right\} \).

The extension of \( t_k \) in general is

\[
t_k = C + \max \left\{ -\frac{s_k^T y_k}{\| s_k \|^2}, 0 \right\} \| g_k \|^{-1}
\]

(3.9)

with some constant \( C > 0 \), and \( \| \cdot \| \) denotes the Euclidean norm of vectors. Then, the \( s_k^T y_k^* \) always holds that

\[
s_k^T y_k^* = s_k^T \left\{ y_k + \left[ C + \max \left\{ -\frac{s_k^T y_k}{\| s_k \|^2}, 0 \right\} \| g_k \|^{-1} \right] \| g_k \| s_k \right\}
\]

\[
= s_k^T y_k + C \| g_k \| \| s_k \|^2 + \| s_k \|^2 \max \left\{ -\frac{s_k^T y_k}{\| s_k \|^2}, 0 \right\}
\]

\[
= \begin{cases} 
  s_k^T y_k + C \| g_k \| \| s_k \|^2 & \text{if } s_k^T y_k \geq 0 \\
  C \| g_k \| \| s_k \|^2 & \text{if } s_k^T y_k < 0
\end{cases}
\]

\[
\geq C \| g_k \| \| s_k \|^2 > 0
\]

(3.10)
Replacing all the $y_k$ with $y_k^*$ into equation (3.7) ensures that $H_{k+1}^{-1}$ is a symmetric positive definite and independent of the convexity of $f(x)$ and the use of line search (Zhang, 2005; Zhang, 2009; Xiao, Li & Wei, 2013).

### 3.2 Modified Hopfield Neural Networks by BFGS methods

The general dynamic equation for HNN is defined by

\[
\frac{dx_j(t)}{dt} = -\sum_{i=1}^{n} \mu_{ji}(t) \frac{\partial E(x(t))}{\partial x_i(t)}, \quad x_j(0) = x_j^{(0)}, \quad j = 1, 2, ..., n \tag{3.11}
\]

which can be written in the compact matrix form

\[
\frac{dx}{dt} = -\mu \nabla E(x), \quad x(0) = x^{(0)} \tag{3.12}
\]

where $x(t) \in \mathbb{R}^n$, $\mu(x, t)$ is an $n \times n$ symmetric positive definite matrix and called convergence parameter, and $\partial E(x)/\partial x$ or $\nabla E(x)$ is the gradient of energy function $E(x)$ at the point $x$.

The $\mu_{ij}(t)$ must be positive to ensure the system stability, i.e. the system reaches an equilibrium point $x^*$ which is the local minimum or inflection point of the energy function $E(x)$. The $\mu_{ij}(t)$ corresponds to the time constant of the integrators or adjusts the convergence rate of the neural network in the hardware implementation. Small values of $\mu_{ij}(t)$, which is the large time constant of the integrators, make a very slow convergence speed of the system, but a large $\mu_{ij}(t)$ may lead to an unstable behavior of the system (Cichocki & Unbehauen, 1994). Normally and simplicity, the $\mu_{ij}(t)$ is set to a large constant that the integrators permit (Zhang, Mu & Zheng, 2013).

Nevertheless, an appropriate value of the convergence parameter can increase the convergence rate and remain stability of the system. Jacobs (1988) suggests heuristics for
increasing the convergence rate as: (i) every $x_j$ should have its own individual $\mu_{ij}(t)$ and its $\mu_{ij}(t)$ can vary over time during the optimization process, (ii) If the $\partial E(x)/\partial x$ has the same sign for several consecutive time steps, the $\mu_{ij}(t)$ should be increased because $x_j$ is still far from a minimum. Conversely, the $\mu_{ij}(t)$ should be decreased if the $\partial E(x)/\partial x$ has the alternative sign for several consecutive time steps, which indicates that $x_j$ is near a minimum. From these heuristics, some convergence rate adaptations are proposed such as momentum, delta-delta learning rule, delta-bar-delta rule (Jacobs, 1988), and adaptive learning rates (Ku & Lee, 1995).

Two main distinctions of BFGS methods and dynamic system are the line search method and the initial condition of parameters. Let’s consider the general algorithms for BFGS method.

Step 1. Given $x_0$ compute $g_0$.

Step 2. Set $d_0 = -H_0^{-1}g_0$

$H_0^{-1}$ is a symmetric positive definite matrix.

Step 3. Obtain $x_{k+1} = x_k + \alpha_k d_k$ where $\alpha_k$ minimized $f(x_k + \alpha_k d_k)$ (i.e. using line search methods).

Step 4. Compute $g_k$

Step 5. Compute $d_{k+1}$,

$$d_{k+1} = -H_{k+1}^{-1}g_{k+1},$$

where $H_{k+1}^{-1}$ is determined by Equation (3.7),

Step 6. Replace $x_k$ by $x_{k+1}$ and return to step 3

To solve the dynamic equation (3.12), one uses any numerical integration method with a fixed small step-size. The integration method may or may not satisfy the line search methods. Therefore, to implement BFGS method the inverse Hessian matrix is defined as
Equation (3.7) with modified $y_k$ as Equation (3.8) is the best choice because of its less dependent of line search and convexity of $\nabla E(x)$. The initial conditions of $H_0^{-1}$ is set by using integrators. Consequently, the proposed BFGS Hopfield neural network modified the general dynamic equation by changing the right-hand side of Equation (3.12) to the Newton direction and implementing integrators to set initial conditions and compute the inverse of Hessian matrix, without using any heuristics.

The dynamic system for the modified network is defined as

$$\frac{dx}{dt} = -\mu_{BFGS} \nabla E(x), \quad (3.13a)$$

$$\frac{d\mu_{BFGS}}{dt} = H^{-1} \quad (3.13b)$$

where

$$H^{-1} = \mu_{BFGS} + \left(1 + \frac{z^T \mu_{BFGS} z}{s^T z} \right) ss^T - \frac{ss^T \mu_{BFGS}}{s^T z} \frac{ss^T + \mu_{BFGS} zs^T}{s^T z}$$

$$s = x(t) - x(t-\tau), \quad y = g(t) - g(t-\tau), \quad g = \nabla E(x)$$

$$z = y + t \|g\|s, \quad t = C + \max \left\{ -\frac{s^T y}{\|s\|^2}, 0 \right\} \|g\|^{-1}, \quad C > 0$$

The time derivative of the energy function $E(x)$ is in the form

$$\frac{dE}{dt} = \sum_{i=1}^{n} \frac{\partial E}{\partial x_i} \frac{dx_i}{dt} = -[\nabla_x E(x)]^T \mu_{BFGS} \nabla_x E(x) \quad (3.14)$$

From the above Equation, the energy function $E(x)$ decreases in time because the $\mu_{BFGS}$ is symmetric and positive definite. The trajectory $x(t)$ converges to a minimum when the $\nabla E(x)$ goes to zero as $t \to \infty$. 
A general functional diagram for the Equation (3.13a) is depicted in Figure 3.1. The diagram consists of continuous-time integrators, analog multipliers and nonlinear function generators (Cichocki & Unbehauen, 1994). Figure 3.2 shows the block diagram for the proposed HNN. The values of $\boldsymbol{\mu}_{BFGS}$ is determined by continuous-time integrators, analog multipliers/dividers, summing amplifiers, and time-delays. Figure 3.3 illustrates the block diagram of inverse Hessian matrix updating, Equation (3.13b). The value of $\mathbf{z}$ is realized by analog multipliers, summing amplifiers, and time-delays as shown in Figure 3.4. The control variable $t$ implements comparators for finding maximum values, analog multipliers/dividers, summing amplifiers, and time-delays. Its block diagram is depicted in Figure 3.5. Moreover, one may use vector-matrix multipliers in the networks to reduce the complexity of circuit’s connections. The initial condition of $\boldsymbol{\mu}_{BFGS}$ is set to be 1s for $i=j$, and 0s for $i \neq j$, when $i$ is the number of rows and $j$ is the number of column of $\boldsymbol{\mu}_{BFGS}$. The number of integrators for determining $\boldsymbol{\mu}_{BFGS}$ depends on the number of neurons. If the system has $n$ neurons, $n^2$ integrators are used to construct the matrix $\boldsymbol{\mu}_{BFGS}$.

Equation (3.13b) is an ascent method, i.e. the value of $\boldsymbol{\mu}_{BFGS}$ is increasing over the time. This will affect the system by multiplying a large number. Thus, the integrators must be reset to its initial condition after the gradient of $E(\mathbf{x})$ is close to zero, i.e. the values of $\mathbf{x}$ are very near to a minimum. After the resetting, the system will turn back to the original negative gradient method or the traditional HNN model. The integrator resetting is performing as the termination criteria of the iterative methods. Without the resetting, the dynamic system of modified HNNs will cope with the integration by a very large number. Consequently, the error of singularity calculation occurs in the simulation.
Figure 3.1  An architecture of the general dynamic equation
Figure 3.2 The block diagram for the proposed BFGS Hopfield neural network
Figure 3.3 A block diagram of inverse Hessian matrix updating.
Figure 3.4 A block diagram of $y^*$
Figure 3.5 A block diagram for $t_k$. 
3.3 Simulation Results

The model simulations have been shown by MATLAB/Simulink. The Simulink model of traditional HNN is depicted in Figure 3.6. The Gradient Vector block contains the equations of $\nabla E(x)$. The Simulink model diagram of the modified HNN is illustrated in Figure 3.7. The $H_{\text{updating}}$ block performs the inverse Hessian approximation as Equation (3.13b). The model diagram of $H_{\text{updating}}$ block is shown in Figure 3.8. To investigate the effect of $\mu_{\text{BFGS}}$, the convergence rate of traditional HNN is set to 1 for all simulation.

3.3.1 Two-dimensional function

Consider the two-dimensional Rosenbrock function

$$E(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

The Rosenbrock function has a global minimum at the point $x^* = [1,1]^T$. The gradients of $E(x)$ are

$$\frac{\partial E}{\partial x_1} = -400(x_2 - x_1^2)x_1 - 2(1 - x_1)$$

$$\frac{\partial E}{\partial x_2} = 200(x_2 - x_1^2)$$

Applying the traditional HNN, the set of differential equations are

$$\frac{dx_1(t)}{dt} = 400(x_2 - x_1^2)x_1 + 2(1 - x_1) = 400x_1x_2 - 400x_1^3 + 2 - x_1$$

$$\frac{dx_2(t)}{dt} = -200(x_2 - x_1^2) = -200x_2 + 200x_1^2$$

Then $T_{11} = -1, T_{12} = 0, T_{21} = 0, T_{22} = 200,$

$$I_1 = 400x_1x_2 - 400x_1^3 + 2, I_2 = -200x_1^2$$
Applying the modified HNN, as Equation (3.13), we obtain the set of differential equations

\[
\frac{dx_1(t)}{dx} = -\mu_{11} \frac{\partial E}{\partial x_1} - \mu_{12} \frac{\partial E}{\partial x_2},
\]

\[
\frac{dx_2(t)}{dx} = -\mu_{21} \frac{\partial E}{\partial x_1} - \mu_{22} \frac{\partial E}{\partial x_2},
\]

when \( \mu_{11}, \mu_{12}, \mu_{21}, \) and \( \mu_{22} \) are calculated by the integral of Equation (3.13b).

The initial point is \( x_0 = [-10,10]^T \). The numerical integration is Euler method with \( 10^{-5} \) step-size. The constant \( C \) is set to 1 and \( \tau = 10^{-5} \). The trajectories \( x(t) \) of the HNN and modified HNN illustrated in Figure 3.9, and 3.10, respectively. From the simulation results, BFGS HNN converges to the minimum point within 6 seconds, while HNN reaches to the minimum point at the time than 20 seconds. The \( x(t) \) converges to \([0.9235, 0.8524]^T\) at \( t = 20 \) seconds for HNN, and to \([1,1]^T\) at \( t = 6 \) seconds for BFGS HNN. The fast convergence of the modified HNN is caused by the values of the \( \mu_{BFGS} \). The values of \( \mu_{BFGS} \) are shown in Figure 3.11. This means that the convergence rate has been increased to 3 times before it reaches the minimum point. When the system reaches the minimum point, the inverse Hessian matrix is reset to the identity matrix as the initial state. Additionally, BFGS HNN has more accuracy than HNN because it goes to the minimum faster than HNN.
Figure 3.6. The Simulink model of HNN.
Figure 3.7 The Simulink model of the modified HNN.
Figure 3.8 The Simulink model of H_updating block
Figure 3.9 The trajectories $x(t)$ of HNN for 2-dimensional Rosenbrock function.
Figure 3.10 The trajectories $x(t)$ of the modified HNN for 2-dimensional Rosenbrock function.
Figure 3.11 The values of BFGS Hessian matrix for 2-dimensional Rosenbrock function.
3.3.2 *Four-dimensional Function*

Consider the Powell function

\[ E(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4 \]

It has a global minimum at the point \( x^* = [0, 0, 0, 0]^T \) with \( E(x^*) = 0 \). In this simulation, we set the initial condition \( x_0 = [3, -1, 0, 1]^T \). The numerical integration is Runge-Kutta method with \( 10^{-4} \) step-size. The constant \( C \) is set to 1. Figure 3.12 depicts the trajectory of traditional HNN. Figure 3.13 shows the trajectory of BFGS HNN. The values of \( \mu_{BFGS} \) are illustrated in Figure 3.14.

The optimal solution for HNN is \( x^* = [0.1012, -0.0101, 0.05002, 0.05081]^T \), and for BFGS HNN is \( x^* = [0.04882, -0.00488, 0.02417, 0.02426]^T \). The convergence time for BFGS HNN is 4 seconds, while HNN converges to the optimal solution in more than 10 seconds of simulation time.
Figure 3.12  The trajectories $x(t)$ of HNN for Powell function.
Figure 3.13  The trajectories $x(t)$ of BFGS HNN for Powell function.
Figure 3.14  The values of BFGS Hessian matrix for Powell Function.
3.3.3 Ten-dimensional Function

Consider the 10-dimensional Rosenbrock function

\[ E(x) = \sum_{i=1}^{9} 100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2 \]

It has one global minimum at \( x = [1, 1, \ldots, 1]^T \) with \( E(x) = 0 \), and a local minimum at \( x = [-1, 1, \ldots, 1]^T \) with \( E(x) = 4 \). In the simulation, we set the initial condition is \( x_0 = [0, 0, \ldots, 0]^T \). The numerical integration is Euler method with 10^{-5} step-size. The constant \( C \) is set to 1. The trajectory of traditional HNN and BFGS HNN are depicted in Figure 3.15, and 3.16, respectively. The values of \( \mu \) are illustrated in Figure 3.17. The optimal solution for HNN is

\[ x^* = [1, 1, 0.9999, 0.9999, 0.9997, 0.9995, 0.999, 0.9979, 0.9959, 0.9917]^T, \]

and for BFGS HNN is \( x^* = [1, 1, 1, 1, 1, 1, 1, 0.9999, 0.9998]^T \).

The convergence time for BFGS HNN is 4 seconds, while HNN converges to the optimal solution in more than 10 seconds of simulation time.
Figure 3.15  The trajectories $x(t)$ of HNN for 10-dimensional Rosenbrock function.
Figure 3.16 The trajectories $x(t)$ of BFGS HNN for 10-dimensional Rosenbrock function.
Figure 3.17. The values of BFGS Hessian matrix for 10-dimensional Rosenbrock function.
3.4 Summary

The proposed method to modify HNN is to change the negative gradient direction to BFGS direction. This method can be implemented to the HNN quite well because it is less depended on any line search methods. For a general function, the BFGS updating should modify some parameters for preserving the positive definite property.

The HNN can converge to a local minimum faster by updating its convergence rate following the BFGS method. The convergence rate of the proposed HNN is more than six times of the traditional HNN in all simulated examples. It also shows a good quality of solution compared to the traditional one.
CHAPTER 4
NAZARETH HOPFIELD NEURAL NETWORK

4.1 Conjugate Gradient Optimization

Suppose that we are trying to minimize an objective function $E(x)$, i.e. finding a point $x = x^*$ which gives the minimum value of $E(x)$. In iterative descent methods, the next point $x_{k+1}$ is determined by

$$x_{k+1} = x_k + \alpha_k d_k$$

(4.1)

where $k$ denotes the current iteration number, $d_k$ is a direction vector, and $\alpha_k$ is a distant toward to $x_{k+1}$ in that direction vector, called step size. The iterative descent methods determine the point $x_{k+1}$ that satisfy

$$E(x_{k+1}) = E(x_k + \alpha_k d_k) < E(x_k)$$

(4.2)

Conjugate gradient methods suggest a suitable direction vector by using the orthogonal of two vectors. This approach results in more speed of convergence than the tradition optimization, such as the steepest descent methods, or the gradient methods (Jang, Sun & Mizutani, 1997).

Given a symmetric matrix $Q_{n \times n}$, two $n$-dimensional vectors $d_i$ and $d_j$ are mutually conjugate with respect to $Q$ or $Q$-orthogonal if

$$d_i^T Q d_j = 0$$

The $d_i$ and $d_j$ are mutually orthogonal if $Q = I$ (the identity matrix).

A line search method is used to determine the value of $\alpha_k$. For the conjugate direction $d_k$, we implement the Gram-Schmidt orthonormalization. In general, an objective function $E(x)$ is approximated by the Taylor series as
\[ E(x_{k+1}) \approx E(x_k) + g_k^T(x_{k+1} - x_k) + \frac{1}{2}(x_{k+1} - x_k)^T H(x_{k+1} - x_k) \quad (4.3) \]

where \( H \) is the second partial derivatives of \( E(x_k) \), called the Hessian matrix, and \( g \) is the first partial derivatives of \( E(x_k) \), called the gradient of \( E(x_k) \).

Differentiating Equation (4.3) respect to \( x_{k+1} \), we have

\[
\frac{\partial E(x_{k+1})}{\partial x_{k+1}} = g_{k+1} = g_k + H(x_{k+1} - x_k) \quad (4.4)
\]

\[
g_{k+1} - g_k = H(x_{k+1} - x_k) \quad (4.5)
\]

From Equation (4.1), the Equation (4.5) can be rewritten as

\[
g_{k+1} - g_k = \alpha_k H d_k \quad (4.6)
\]

The step size \( \alpha_k \) can be determined by line minimization

\[
\alpha^*_k = \arg \min_{\alpha_k > 0} \phi(\alpha_k) \quad (4.7)
\]

Where

\[
\phi(\alpha_k) = f(x_k + \alpha_k d_k) \quad (4.8)
\]

Differentiating Equation (4.3) respect to \( \alpha_k \) and set it to zero, then the \( \alpha_k \) will be expressed as

\[
E(x_k + \alpha_k d_k) = E(x_k) + g_k^T(\alpha_k d_k) + \frac{1}{2}(\alpha_k d_k)^T H(\alpha_k d_k) \quad (4.9)
\]

\[
\frac{\partial E(x_k + \alpha_k d_k)}{\partial \alpha_k} = g_k^T d_k + (\alpha_k d_k)^T H d_k = 0 \quad (4.10)
\]

\[
\alpha_k = -\frac{g_k^T d_k}{d_k^T H d_k} \quad (4.11)
\]

The conjugate direction \( d_k \) is determined using Gram-Schmidt orthonormalization with the value of
Equation (4.12) implied a direction \( \mathbf{d}_k \) conjugated to all previous descent direction \( \mathbf{d}_j \) (\( j < k \)) with respect to \( \mathbf{H} \). The \( \mathbf{s}_k \) is any arbitrary basis.

By setting \( \mathbf{s}_k \) to the negative gradient \( -\mathbf{g}_k \), Equation (4.12) is known as the conjugate gradient direction and becomes

\[
\mathbf{d}_k = -\mathbf{g}_k + \sum_{j=0}^{k-1} \mathbf{d}_j^T \mathbf{H} \mathbf{s}_k \mathbf{d}_j
\]  

(4.13)

Using Equation (4.6) yields

\[
\mathbf{d}_k = -\mathbf{g}_k + \sum_{j=0}^{k-1} \mathbf{g}_k^T \left( \mathbf{g}_{j+1} - \mathbf{g}_j \right) \mathbf{d}_j
\]  

(4.14)

Because the \( \mathbf{g}_j \mathbf{d}_j = 0 \) for \( j < k \), Equation (4.14) reduces to

\[
\mathbf{d}_k = -\mathbf{g}_k + \beta_k \mathbf{d}_{k-1}
\]  

(4.15)

where

\[
\beta_k = \frac{\mathbf{g}_k^T (\mathbf{g}_k - \mathbf{g}_{k-1})}{\mathbf{d}_{k-1}^T (\mathbf{g}_k - \mathbf{g}_{k-1})}
\]  

(4.16)

This \( \beta_k \) is developed by (Hestenes & Steifel, 1952). The other formulas of \( \beta_k \) are determined by

Polak-Ribiers-Polyak (Polak & Ribiere 1969; Polyak, 1969):

\[
\beta_k = \frac{\mathbf{g}_k^T (\mathbf{g}_k - \mathbf{g}_{k-1})}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}}
\]  

(4.17)

Fletcher-Reeves (Fletcher & Reeves, 1964):

\[
\beta_k = \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}}
\]  

(4.18)
Liu-Storey (Liu & Storey, 1991):

\[
\beta_k = -\frac{\nabla f_k^T (\nabla f_k - \nabla f_{k-1})}{\nabla f_{k-1}^T \nabla f_{k-1}}
\]

(4.19)

Dai-Yuan (Dai & Yuan, 1999):

\[
\beta_k = \frac{\nabla f_k^T \nabla f_k}{\nabla f_{k-1}^T (\nabla f_k - \nabla f_{k-1})}
\]

(4.20)

4.2 Nazareth Conjugate Gradient Optimization

Conjugate gradient and BFGS algorithms show a closed relationship with each other. Nazareth (1979) proposed the connection between these two algorithms and presented an algorithm that accelerates the convergence of the conjugate gradient method.

Recalled Equation (4.15) and (4.16) the conjugate gradient direction with a fixed metric defined by the positive symmetric matrix \( H \) would be

\[
d_{CG}^k = -H \nabla f_k + \frac{\nabla f_k^T H \nabla f_k}{\nabla f_{k-1}^T d_{CG}^{k-1}} d_{CG}^{k-1}
\]

(4.21)

The BFGS updating is defined by

\[
H_{k+1}^{-1} = H_k^{-1} + (1 + \frac{\nabla f_k^T H_k^{-1} \nabla f_k}{\nabla f_{k-1}^T \nabla f_{k-1}}) \left( \frac{s_k^T s_k}{s_k^T \nabla f_k} \right) - \left( \frac{s_k \nabla f_k^T H_k^{-1} + H_k^{-1} \nabla f_k s_k^T}{s_k^T \nabla f_k} \right)
\]

(4.22)

where

\[
s_k = x_{k+1} - x_k, \text{ and } y_k = \nabla f_{k+1} - \nabla f_k.
\]

and the search direction of BFGS method is

\[
d_{BFGS}^k = -H_{BFGS}^k \nabla f_k
\]

(4.23)

From Equation (4.22) and using the exact line searches \( (\nabla f_{k+1}^T d_k = 0) \), the BFGS direction can be expressed as
\[ d_k^{BFGS} = -H_{k-1}^{BFGS} g_k + \frac{y_k^T H_{k-1}^{BFGS} g_k}{y_k^T d_{k-1}^{BFGS}} d_{k-1}^{BFGS} \] (4.24)

If \( E(x_k) \) is a quadratic function and using the same starting point and positive definite symmetric, the BFGS direction is equal to the conjugate gradient direction or \( H_k^{BFGS} g_k = H_{k-1}^{BFGS} g_k = H g_k \) (Nazareth, 1979). Then, the Equation (4.24) is simplified as

\[ d_k^{BFGS} = -H g_k + \frac{y_k^T H g_k}{y_k^T d_{k-1}^{BFGS}} d_{k-1}^{BFGS} \] (4.25)

Otherwise, \( d_k^{CG} \) and \( d_k^{BFGS} \) are linearly dependent. The equation (4.25) can be obtained as

\[ d_k^{BFGS} = -H g_k + \frac{y_k^T H g_k}{y_k^T d_{k-1}^{BFGS}} d_{k-1}^{CG} = d_k^{CG} \] (4.26)

From equation (4.26) it shows \( d_k^{BFGS} = d_k^{CG} \), i.e. for the BFGS update the search vectors \( d_k^{CG} \) and \( d_k^{BFGS} \) are the same in norm as well as direction.

By comparing Equation (4.21) and (4.24), one can observe that the BFGS methods may be implied as a conjugate gradient algorithm for which the metric \( H \) is updated as each step instead of being fixed. That is, the conjugate gradient method can be improved by using limited storage of vectors. Conjugate gradient algorithms require four vectors in storage, but BFGS algorithms need \( n^2 \) storage. Moreover, the conjugate gradient method requires two more factors of iterations for convergence than the BFGS method. Therefore, the convergence of the conjugate gradient method can be accelerated by using more information about the function.

A variable storage generalized conjugate gradient (VSGCG) algorithm can be used to exploit additional storage and form a continuum between the BFGS and conjugate gradient method. The procedures of this algorithm are described as follows.
**On Input.**

- $n$: dimension of the problem.
- $\theta_1$: starting point.
- $\delta$: vector giving diagonal elements of initial diagonal approximation to inverse Hessian $H_0$.

**Step 1: Initialize.**

$$E_1 \leftarrow E(\theta_1), \; g_1 \leftarrow g(\theta_1), \; (g_1 - g_0) \leftarrow 0, \; d_0 \leftarrow 0, \; H_0 \text{ and } H_1 \text{ are diagonal matrices defined by } \delta, \; k \leftarrow 0.$$

**Step 2: Develop search direction.**

$$d_k \leftarrow -H_{k-1}g_k + \frac{y_k^T H_{k-1} g_k}{y_k^T d_{k-1}} d_{k-1}$$

**Step 3: Search.**

- $k \leftarrow k+1$ if $g_k^T d_k > 0$ then restart;
- $x_{k+1} \leftarrow x_k + \alpha_k d_k$, 
- $\alpha_k^* = \arg \min_{\alpha_k > 0} \phi(\alpha_k)$

where

$$\phi(\alpha_k) = f(x_k + \alpha_k d_k)$$

**Step 4: Test for convergence.**

If convergence criterion is met then Stop

else go to Step 5

**Step 5: If** available storage is exceeded.

Step 5A then employ a Reset Option
Step 5B  else  update $H_{k-1}$ to $H_k$ using a member of the Broyden’s $\beta$-class.

**Step 6:** If restart criterion not satisfied then go to Step 2

else  employ suitable restart option

The resetting option is to hold $H_k$ at a fixed $H$. The restart criterion is to set $H_k$ to $H_0$ at every $n$ or $n+1$ iterations.

In conclusion, the VSGCG algorithm uses the variable metric information, especially BFGS update, within the framework of conjugate gradient methods.

### 4.3 Modified Hopfield Neural Networks by Nazareth Algorithm

The conceptual design of modified HNN is similar to the BFGS HNN described in Chapter 3. The dynamic equation for HNN is changed to the direction defined by Equation (4.25). The proposed method is described as follow.

Considering the unconstrained optimization problems,

$$\min f(x)$$

where $x \in \mathbb{R}^n$, and $f(x)$ is a nonlinear function and has Lipschitz continuous gradient. The conjugate gradient method can be considered as

$$x_{k+1} = x_k + \alpha_k d_k \quad (4.27)$$

$$d_{k+1} = -g_{k+1} + \beta_k d_k \quad (4.28)$$

Where $g_k = \nabla f(x_k)$, $\alpha_k$ is to minimize $f(x)$ along the search direction $d_k$, and $\beta_k$ is defined by

$$\beta_k = \frac{y^T_k g_{k+1}}{y^T_k d_k} \quad (4.29)$$
where \( y_k = g_{k+1} - g_k \). The equation (4.28) can be rewritten in the general form (Perry, 1976) as

\[
d_{k+1} = - \left( I - \frac{d_k y_k^T}{y_k^T d_k} \right) g_{k+1}
\]  

(4.30)

Defining

\[
s_k = x_{k+1} - x_k = \alpha_k d_k
\]

Then the value of \( \alpha_k \) is relaxed from the exact line search method (Shanno, 1978), and the equation (4.30) is derived as

\[
d_{k+1} = - \left( I - \frac{s_k y_k^T}{y_k^T s_k} \right) g_{k+1}
\]  

(4.31)

The Nazareth direction, Equation (4.22), can also be relaxed from exact line search by changing \( d_k \) to \( s_k \). Therefore, the relaxed Nazareth direction is

\[
d_{k+1} = - H_k g_{k+1} + \frac{y_k^T H_k g_{k+1} d_k}{y_k^T d_k}
\]

\[
= - \left( H_k - \frac{d_k y_k^T H_k}{y_k^T d_k} \right) g_{k+1}
\]

\[
= - \left( H_k - \frac{s_k y_k^T H_k}{y_k^T s_k} \right) g_{k+1}
\]

\[
= - \left( I - \frac{s_k y_k^T}{y_k^T s_k} \right) H_k g_{k+1}
\]  

(4.32)

where \( H_k \) is updated by a Broyden’s \( \beta \)-class (Broyden, 1970).

Although the equation (4.32) is relaxed from the exact line search method, it is not symmetric, and not positive definite (Shanno, 1978). Hence, Equation (4.32) is not
necessarily descent directions. Some modifications should be performed to agree with symmetric positive definite, and secant equation.

Defined

\[ Q_{k+1} = I - \frac{s_k y_k^T}{y_k^T s_k} \]  

(4.33)

The modified \( Q_{k+1} \) is defined as (Shanno, 1978),

\[ Q_{k+1} = I - \frac{s_k y_k^T + y_k s_k^T}{y_k^T s_k} + (1 + \frac{y_k^T y_k}{y_k^T s_k}) \frac{s_k s_k^T}{y_k^T s_k} \]  

(4.34)

Then, the conjugate gradient methods with this direction are satisfied the secant equation and symmetric positive definite. Additionally, Equation (4.34) is the fixed variable matrix version of BFGS matrix. The conclusion is consistent with the concept of Nazareth algorithm as mentioned above.

The Equation (4.32), thus, can be modified to

\[ d_{k+1} = -(H_k - \frac{s_k y_k^T H_k + H_k y_k s_k^T}{y_k^T s_k} + (1 + \frac{y_k^T y_k}{y_k^T s_k}) \frac{s_k s_k^T}{y_k^T s_k})g_{k+1} \]  

(4.35)

Hence, the modified Nazareth direction is realized as

\[ d_{k+1} = -(I - \frac{s_k y_k^T}{y_k^T s_k} + y_k s_k^T + (1 + \frac{y_k^T y_k}{y_k^T s_k}) \frac{s_k s_k^T}{y_k^T s_k})H_k g_{k+1} \]  

(4.36a)

or \[ d_{k+1} = \mu_{CG} \mu_{\beta} g_{k+1} \]  

(4.37b)

where

\[ \mu_{CG} = (I - \frac{s_k y_k^T + y_k s_k^T}{y_k^T s_k} + (1 + \frac{y_k^T y_k}{y_k^T s_k}) \frac{s_k s_k^T}{y_k^T s_k}), \]

\[ \mu_{\beta} = H_k, \text{ and } H_k \text{ is defined form a member of Broyden’s } \beta\text{-class.} \]
The symmetric rank-1 update is defined by

\[
H_{k+1} = H_k - \frac{(s_k - H_k y_k)(s_k - H_k y_k)^T}{(s_k - H_k y_k)^T y_k}
\]  

(4.38)

This update is the recommended Broyden’s β-class by Nazareth (1979) because of its less memory storage of vectors than other members. One can choose BFGS update with resetting if the memory storage is less concerned.

For nonlinear function \( f(x) \), \( \mu_{CG} \) and \( \mu_{B} \) may or may not hold a positive definite for every iteration. Changing the \( y_k \) to \( y^* \) as Equation (3.8) would remain the property of positive definite for \( \mu_{CG} \) and \( \mu_{B} \) and not depend on the line search method (Zhang, 2009; Xiao et al., 2013; Dai, et al., 2015).

Therefore, the dynamic system for the modified network is defined as

\[
\frac{dx}{dt} = -\mu_{CG} \mu_{BFGS} \nabla E(x),
\]  

(4.39a)

\[
\frac{d\mu_{CG}}{dt} = Q
\]  

(4.39b)

\[
\frac{d\mu_{BFGS}}{dt} = H^{-1}
\]  

(4.39c)

where

\[
Q = I - \frac{sz^T + zs^T}{z^T s} + \left(1 + \frac{z^T z_k}{z^T s}\right) \frac{ss^T}{z^T s}
\]

\[
H^{-1} = \mu_{BFGS} + \left(1 + \frac{z^T \mu_{BFGS} z}{s^T z}\right) \frac{ss^T}{s^T z} - \left(\frac{sz^T \mu_{BFGS} + \mu_{BFGS} zs^T}{s^T z}\right)
\]

\[
s = x(t) - x(t-\tau), \quad y = g(t) - g(t-\tau), \quad g = \nabla E(x)
\]

\[
z = y + t \frac{\|g\|}{s} s, \quad t = C + \max\left\{-\frac{s^T y}{\|s\|^2}, 0\right\} \|g\|^{-1}, \quad C > 0
\]

The time derivative of the energy function \( E(x) \) is in the form
\[
\frac{dE}{dt} = \sum_{i=1}^{n} \frac{\partial E}{\partial x_i} \frac{dx_i}{dt} = -[\nabla_x E(x)]^T \mu_{CG} \mu_{BFGS} \nabla_x E(x)
\]  
(4.40)

From the above Equation, the energy function \(E(x)\) decreases in time because the \(\mu_{CG}\) and \(\mu_{BFGS}\) are symmetric positive definite. The trajectory \(x(t)\) converges to a minimum when the \(\nabla E(x)\) goes to zero as \(t \to \infty\).

Figure 4.1 shows the block diagram for the proposed HNN. The diagram consists of analog integrators, multipliers, and nonlinear function generators. The diagram for Equation (4.39b) is depicted in Figure 4.2. The diagram for Equation (4.39c) is the same as Figure 3.3. The diagram of \(y^*_k\) and \(t_k\) have been illustrated in Figure 3.4, and 3.5, respectively. The initial condition of \(\mu_{CG}\) and \(\mu_{BFGS}\) is set to be 1s for \(i=j\), and 0s for \(i \neq j\), when \(i\) is the number of rows and \(j\) is the number of column of each \(\mu\). The number of integrators for determining \(\mu_{CG}\) and \(\mu_{BFGS}\) depends on the number of neurons. If the system has \(n\) neurons, \(2n^2\) integrators are used to construct the matrix \(\mu_{CG}\) and \(\mu_{BFGS}\) in total. The dynamic equation (4.39b), and (4.39c) are ascent direction, i.e. the values of \(\mu_{CG}\) and \(\mu_{BFGS}\) are increasing over time. Then, the integrators must be reset to avoid the huge values of \(\mu_{CG}\) and \(\mu_{BFGS}\) that affect the instability of the system. When the values of the gradient of \(E(x)\) is close to zero, the integrators would be reset to their initial conditions.
Figure 4.1 A block diagram of the modified HNN by Nazareth direction.
Figure 4.2 A block diagram of the Equation (4.34b)
4.4 Simulation Results

By using MATLAB/Simulink to simulate the minimization of the given functions, the model of the modified HNN is depicted in Figure 4.3. The CG Updating block is similar to the H Updating block in Figure 3.8. Instead of sending a feedback of $\mu_{BFGS}$, the block uses an identity matrix block to perform the update. The updating of $\mu_{CG}$ is illustrated in Figure 4.4. We use the same functions as the simulation section in Chapter 3 to compare the convergence speed of BFGS HNN and Nazareth HNN.

4.4.1 Two-dimensional function

Consider the two-dimensional Rosenbrock function

$$E(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

which has a global minimum at the point $x^* = [1,1]^T$. Applying the modified HNN, the set of differential equations can be in the form of

$$\frac{dx_1(t)}{dt} = -\left(\mu_{CG11}\mu_{BFGS11} + \mu_{CG12}\mu_{BFGS21}\right) \frac{\partial E}{\partial x_1} - \left(\mu_{CG11}\mu_{BFGS12} + \mu_{CG12}\mu_{BFGS22}\right) \frac{\partial E}{\partial x_2},$$

$$\frac{dx_2(t)}{dt} = -\left(\mu_{CG21}\mu_{BFGS11} + \mu_{CG22}\mu_{BFGS21}\right) \frac{\partial E}{\partial x_1} - \left(\mu_{CG21}\mu_{BFGS12} + \mu_{CG22}\mu_{BFGS22}\right) \frac{\partial E}{\partial x_2},$$

where

$$\frac{\partial E}{\partial x_1} = -400(x_2 - x_1^2)x_1 - 2(1 - x_1), \quad \frac{\partial E}{\partial x_2} = 200(x_2 - x_1^2)$$
Figure 4.3 Simulink model of the Nazareth HNN.
Figure 4.4    The Simulink block diagram for updating $\mu_{CG}$
The simulation uses Euler method with $10^{-5}$ step-size. The initial point is $x_0 = [-10,10]^T$. The constant $C$ is set to 1 and $\tau = 10^{-5}$. The trajectory of $x$ is shown in Figure 4.5. The adaptive rate of the system is depicted in Figure 4.6. The $x(t)$ converges to $[0.9235, 0.8524]^T$ at $t = 20$ seconds for HNN, and to $[1,1]^T$.

The Nazareth HNN converges to the minimum point $x^* = [1,1]^T$ within 3 seconds, which is 6 times faster than the original HNN. Comparing the result from Figure 3.10 and 4.5, the Nazareth HNN has a faster speed of convergence than BFGS HNN. The higher convergence rate is because of the multiplication of $\mu_{CG}$ and $\mu_{BFGS}$. 
Figure 4.5 The trajectory of $x(t)$ when applying Nazareth HNN.
Figure 4.6 The $\mu_{CG}$ updating.
4.4.2 **Four-dimensional Function**

Consider the Powell function

\[
E(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4
\]

It has a global minimum at the point \( \mathbf{x}^* = [0, 0, 0, 0]^T \) with \( E(\mathbf{x}^*) = 0 \). In this simulation, we set the initial condition \( \mathbf{x}_0 = [3, -1, 0, 1]^T \). The numerical integration is Runge-Kutta method with \( 10^{-4} \) step-size. The constant \( C \) is set to 1 and \( \tau = 10^{-5} \). Figure 4.7 shows the trajectory of Nazareth HNN. The values of \( \mu \) updating are illustrated in Figure 4.8.

The optimal solution for Nazareth HNN is \( \mathbf{x}^* = [0.06289, -0.006284, 0.03117, 0.03137]^T \). The convergence time for Nazareth HNN is 3 seconds, while HNN converges to the optimal solution in more than 10 seconds of simulation time.
Figure 4.7 The trajectory $x(t)$ of Nazareth HNN for Powell function.
Figure 4.8 The updating matrix of Nazareth HNN for Powell function.
Consider the 10-dimensional Rosenbrock function

\[ E(x) = \sum_{i=1}^{9} 100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2 \]

It has one global minimum at \( x = [1,1,...,1]^T \) with \( E(x) = 0 \), and a local minimum at \( x = [-1,1,...,1]^T \) with \( E(x) = 4 \). In the simulation, we set the initial condition is \( x_0 = [0,0,...,0]^T \). The numerical integration is Euler method with \( 10^{-5} \) step-size. The constant \( C \) is set to 1 and \( \tau = 10^{-5} \). The trajectory of Nazareth HNN is depicted in Figure 4.9. The value of the updating matrix is illustrated in Figure 4.10. The optimal solution for Nazareth HNN is \( x^* = [1,1,1,1,1,1,1,1,0.9999]^T \).

The convergence time for Nazareth HNN is 4 seconds, while HNN converges to the optimal solution in more than 10 seconds of simulation time.
Figure 4.9 The trajectory $x(t)$ of Nazareth HNN for Extended Rosenbrock function.
Figure 4.10 The values of the updating matrix of Nazareth HNN for Extended Rosenbrock function.
4.5 Summary

The modified Hopfield neural network is a combination of conjugate gradient method and the BFGS method. Our proposed method has modified the original conjugate gradient direction in Nazareth algorithm to contain the symmetric, and positive definite followed by the Shanno’s conjugated gradient direction. The convergence speed of the modified HNN is faster than HNN and BFGS HNN in all examples.
CHAPTER 5
CONSTRAINED OPTIMIZATION

This chapter explains the models and methods of the modified HNNs to solve constrained optimization problem. The constraints are transformed to a penalty term and combined with the objective function to make an energy function. Then, the energy function is minimized by the techniques described in chapter 3, and 4.

5.1 Exterior Penalty Method

Consider the general form of optimization problems:

$$\min f(x)$$

Subject to $g(x) \geq 0$

where $x \in \mathbb{R}^n$, $f: \mathbb{R}^n \rightarrow \mathbb{R}$, and $g = [g_1(x), g_2(x), \ldots, g_p(x)]^T: \mathbb{R}^n \rightarrow \mathbb{R}^p$ is a $p$-dimensional vector with the function of $n$ variables. The $f$ and $g$ are continuously differentiable functions. Simplicity, the constrained problems can be changed to the unconstrained problems by using penalty function methods. Then the form of penalty constrained problems is defined as:

$$\min E(x) = f(x) + kP(g(x))$$  \hspace{1cm} (5.1)

where $k$ is a positive constant and $P(g(x))$ is a penalty function defined by

$$P(g(x)) = \begin{cases} 0, & \text{if } g(x) \geq 0 \\ > 0, & \text{if } g(x) < 0 \end{cases}$$  \hspace{1cm} (5.2)

Tank and Hopfield (1986) suggested a nonlinear function in the form of
Their penalty function is used for a linear programming problem and easy to build a corresponding real circuit. However, Kennedy and Chua (1987) found that the Tank and Hopfield’s penalty function might not be suitable in practice. The penalty function in Equation (5.3) is not bounded from below. Then, the original Hopfield network circuit is a minimization problem without lower bound. Kennedy and Chua (1987), then, proposed an appropriate nonlinear function in the form of

$$P(g(x)) = \begin{cases} 0, & \text{if } g(x) \geq 0 \\ -\frac{1}{R}g(x), & \text{if } g(x) < 0 \end{cases} \quad (5.4)$$

where $R > 0$. This function has been made by using an ideal diode with its terminals reversed.

Lillo, Loh, Hui, and Zak (1993) applied an exact penalty function to solve linear and nonlinear programming problems. The penalty function can be formed as

$$P(g(x)) = \frac{1}{q} \sum_{j=1}^{p} \left[ g_{j}(x) \right]^{q} \quad (5.5)$$

where $g_{j}(x) = -\min(0, g_{j}(x))$ and $q > 0$. If $q > 1$, the penalty function can be differentiable on the border of the feasible region. A nonlinear resistor generates the penalty function following the canonical nonlinear programming circuit (Kennedy & Chua, 1988).

Many penalty functions or optimization methods, such as Lagrange multiplier or augmented Lagrange multiplier, can effectively solve the constrained optimization problems. However, these methods give a complexity and expensive computation to the
system which difficult to build a corresponding circuit (Cichocki, Unbehauen, Weinzierl & Holzel, 1996).

When the penalty term combined into the objective function, the problems turn to be unconstrained optimization problems. Using the general HNN to find the minimum of the constrained problems, the dynamic equation is a set of differential equation defined as

\[ \frac{dx}{dt} = -\mu \nabla E(x), \quad x(0) = x^{(0)} \]  \hspace{1cm} (5.6)

\[ \frac{dx_i}{dt} = -\mu_i \left( \frac{\partial f(x)}{\partial x_i} + \sum_{j=1}^{p} k_j \frac{\partial P \frac{\partial g_j(x)}{\partial x_j}}{\partial x_i} \right), \quad x_i(0) = x_i^{(0)} \]  \hspace{1cm} (5.7)

where \( \mu_i > 0 \) is the convergence rate, and \( k_j > 0 \) is the penalty parameter. Based on the Equation (5.5) with \( q = 2 \), we get a quadratic penalty function

\[ P(g_j(x)) = \left[ \min \{0, g_j(x)\} \right]^2 \]  \hspace{1cm} (5.8)

Hence, the energy function for this type of penalty term is

\[ E(x, k) = f(x) + \frac{1}{2} \sum_{j=1}^{p} k_j \left[ \min \{0, g_j(x)\} \right]^2 \]  \hspace{1cm} (5.9)

The dynamic equations for Equation (5.9) is obtained

\[ \frac{dx_i}{dt} = -\mu_i \left( \frac{\partial f(x)}{\partial x_i} + \sum_{j=1}^{p} k_j \left[ \min \{0, g_j(x)\} \right] \frac{\partial g_j(x)}{\partial x_i} \right), \]  \hspace{1cm} (5.10)

The function \( \min \{0, g_j(x)\} \) is realized by a voltage-controlled switch (Cichocki, & Unbehauen, 1994). It performs the step function defined as

\[ S_j = \begin{cases} 0, & \text{if } g_j(x) \geq 0, \\ 1, & \text{if } g_j(x) < 0 \end{cases} \]  \hspace{1cm} (5.11)
If the constraint \( g_j(x) \geq 0 \), the dynamic system will not consider this constraint. However, if the constraint \( g_j(x) < 0 \), it will be multiplied by \( k_i \) and integrated into the system. The complete dynamic equation for the penalty energy function can be formulated as

\[
\frac{dx_i}{dt} = -\mu_i \left( \frac{\partial f(x)}{\partial x_i} + \sum_{j=1}^{p} k_j S_j g_j(x) \frac{\partial g_j(x)}{\partial x_i} \right), \quad i = 1, 2, \ldots, n \tag{5.12}
\]

where,

\[
S_j = \begin{cases} 
0, & \text{if} \quad g_j(x) \geq 0, \\
1, & \text{if} \quad g_j(x) < 0
\end{cases}
\]

The penalty parameters \( k_i \) take effect on the solution accuracy. The exact solution is found or in a feasible solution when the penalty parameters tend to infinity. For large penalty parameters, the minimum point of Equation (5.1) will be in a region where \( P(g_j(x)) \) is small. Thus, the minimum point would be expected in the feasible region (Luenberger, & Ye, 2008). However, the large values of penalty parameters may not be suitable for hardware implementation. The penalty energy function may be changed in the form (Cichocki, & Bargiela, 1997)

\[
E(x,k) = \gamma f(x) + \frac{1}{2} \sum_{j=1}^{p} k_j \left[ \min \left\{ 0, g_j(x) \right\} \right]^2
\tag{5.13}
\]

The parameter \( \gamma \) decreases to zero as time increase to infinity. In practical, the parameter \( \gamma \) is set to a sufficiently small value. Then parameter \( k_j \) can be a small number and will take the role of the weight of each constraint.

87
5.2 Modified Hopfield Neural Networks

From Chapter 3, and 4, the proposed modified HNN can increase the convergence speed by gradually increasing the values of $\mu_{CG}$ and $\mu_{BFGS}$. The matrix form of Equation (5.12) is defined as

$$\frac{dx}{dt} = -\mu \left( \nabla f(x) + J^T k S g \right),$$  \hspace{1cm} (5.14)

where

$\mu$ is a $n\times n$ positive definite matrix,

$\nabla f(x)$ is the derivative of $f(x)$

$J^T$ is the transpose of Jacobian matrix of constraints, defined as

$$J^T = \begin{bmatrix}
\frac{\partial g_1(x)}{\partial x_1} & \frac{\partial g_2(x)}{\partial x_1} & \cdots & \frac{\partial g_p(x)}{\partial x_1} \\
\frac{\partial g_1(x)}{\partial x_2} & \frac{\partial g_2(x)}{\partial x_2} & \cdots & \frac{\partial g_p(x)}{\partial x_2} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial g_1(x)}{\partial x_n} & \frac{\partial g_2(x)}{\partial x_n} & \cdots & \frac{\partial g_p(x)}{\partial x_n}
\end{bmatrix}$$

$k = \text{diag}(k_1, k_2, \ldots, k_p)$

$S = \text{diag}(S_1, S_2, \ldots, S_p)$

$g = \begin{bmatrix} g_1(x), g_2(x), \ldots, g_p(x) \end{bmatrix}^T$

Therefore, the proposed BFGS HNN becomes

$$\frac{dx}{dt} = -\mu_{BFGS} \left( \nabla f(x) + J^T k S g \right),$$  \hspace{1cm} (5.15a)

$$\frac{d\mu_{BFGS}}{dt} = H^{-1}$$  \hspace{1cm} (5.15b)
where

\[ H^{-1} = \mu_{BFGS} + (1 + \frac{s^T \mu_{BFGS} z}{s^T z}) ss^T - \left( \frac{sz^T \mu_{BFGS}}{s^T z} \right) \]

\[ s = x(t) - x(t-\tau), \quad y = g_x(t) - g_x(t-\tau), \quad g_x = \nabla E(x) \]

\[ z = y + t \| g_x \| s, \quad t = C + \max \left\{ -\frac{s^T y}{\|s\|^2}, 0 \right\} \| g_x \|^{-1}, \quad C > 0 \]

and the proposed Nazareth HNN is formulated as

\[ \frac{d\mu_{CG}}{dt} = Q \]

\[ \frac{d\mu_{CG}}{dt} = H^{-1} \]

where

\[ Q = I - \frac{sz^T + zs^T}{s^T s} + (1 + \frac{s^T z}{s^T s}) ss^T \]

\[ H^{-1} = \mu_{BFGS} + (1 + \frac{s^T \mu_{BFGS} z}{s^T z}) ss^T - \left( \frac{sz^T \mu_{BFGS}}{s^T z} \right) \]

\[ s = x(t) - x(t-\tau), \quad y = g_x(t) - g_x(t-\tau), \quad g_x = \nabla E(x) \]

\[ z = y + t \| g_x \| s, \quad t = C + \max \left\{ -\frac{s^T y}{\|s\|^2}, 0 \right\} \| g_x \|^{-1}, \quad C > 0 \]

5.3 Simulation Results

Some examples are solved by the modified HNNs. Simulations have been performed using MATLAB/Simulink. The simulations show the convergence speed of each modified HNNs. All examples use the equation (5.4) as their penalty function with \( R = 1 \).
5.3.1 A simple linear programming

Consider a linear program (Wang, 1992);

\[
\begin{align*}
\text{min} & \quad f(x) = -2x_1 - 3.5x_2 \\
\text{Subject to} & \quad x_1 - 4x_2 + 1 \geq 0 \\
& \quad -2x_1 - 3x_2 + 3.5 \geq 0 \\
& \quad -2x_1 - x_2 + 3 \geq 0 \\
& \quad x_1, x_2 \geq 0
\end{align*}
\]

The optimal solution of this problem is \( x^* = [1.0, 0.5]^T \). In this simulation, the parameter \( \gamma = 1e-2 \), and all \( k = 1 \). The initial \( x_0 = [0.1, 1.5]^T \). The simulation results are given in Figure 5.1, 5.2, and 5.3, for HNN, BFGS HNN, Nazareth HNN, respectively. The HNN converges to the optimal solution at time \( t = 25 \) seconds with \( x^* = [1.0004, 0.5011]^T \). The BFGS HNN reaches the optimal point \( x^* = [1.0004, 0.5011]^T \) at time \( t = 0.7 \) seconds. For Nazareth HNN, the optimal solution is \( x^* = [1.0004, 0.5011]^T \), and convergence time is 0.5 seconds. Table 5.1 concludes the solutions and computational time of each search direction. The BFGS HNN and Nazareth HNN converge to the optimal point faster than HNN around 20 times.
Figure 5.1  Simulation result from HNN for a simple linear programming.
Figure 5.2   Simulation result from BFGS HNN for a simple linear programming.
Figure 5.3  Simulation result from Nazareth direction for a simple linear programming.
Table 5.1 The solutions and convergence times for the modified HNN for a simple linear programming

<table>
<thead>
<tr>
<th>Search Direction</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steepest Descent</td>
<td>1.004</td>
<td>0.5011</td>
<td>25</td>
</tr>
<tr>
<td>BFGS</td>
<td>1.004</td>
<td>0.5011</td>
<td>0.7</td>
</tr>
<tr>
<td>Nazareth</td>
<td>1.004</td>
<td>0.5011</td>
<td>0.5</td>
</tr>
</tbody>
</table>
5.3.2 The Rosen-Suzuki problem (Lian, 2012)

\[
\min f(x) = x_1^2 + x_2^2 + 2x_3^2 + x_4^2 - 5x_1 - 5x_2 - 21x_3 + 7x_4
\]

Subject to

\[
-2x_1^2 - x_2^2 - x_3^2 - 2x_1 + x_2 + x_4 + 5 \geq 0
\]

\[
-x_1^2 - x_2^2 - x_3^2 - x_4^2 - x_1 + x_2 - x_3 + x_4 + 8 \geq 0
\]

\[
-x_1^2 - 2x_1^2 - x_3^2 - 2x_4^2 + x_1 + x_4 + 10 \geq 0
\]

The optimal solution of this problem is \( x^* = [0, 1, 2, -1]^T \). The parameters in this simulation have been set as \( \gamma = 1e-2 \), and all \( k = 10 \). The initial \( x_0 = [5, 1, -1, -5]^T \). The simulation results are given in Figure 5.4, 5.5, and 5.6, respectively. The HNN converges to the optimal solution at time \( t = 100 \) seconds with \( x^* = [0.1361, 1.093, 1.858, -1.139]^T \). The BFGS HNN reaches the optimal point \( x^* = [0.1372, 1.092, 1.857, -1.14]^T \) at time \( t = 4 \) seconds. For Nazareth HNN, the optimal solution is \( x^* = [0.1361, 1.093, 1.858, -1.139]^T \), and convergence time is 3 seconds.

Table 5.2 shows the computation time and results of each search direction. The results are close to the optimal solution. If the value of \( k \) is increased, the results are more closely to the optimal solution. Nazareth HNN, and BFGS HNN show very high rate of convergence.
Figure 5.4 Simulation result from HNN for The Rosen-Suzuki problem.
Figure 5.5  Simulation result from BFGS HNN for The Rosen-Suzuki problem.
Figure 5.6  Simulation result from Nazareth direction for The Rosen-Suzuki problem
Table 5.2    The solutions and convergence times of the modified Hopfield Neural Networks for The Rosen-Suzuki problem

<table>
<thead>
<tr>
<th>Search Direction</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steepest Descent</td>
<td>0.1361</td>
<td>1.093</td>
<td>1.858</td>
<td>-1.139</td>
<td>100</td>
</tr>
<tr>
<td>BFGS</td>
<td>0.1372</td>
<td>1.092</td>
<td>1.857</td>
<td>-1.140</td>
<td>4</td>
</tr>
<tr>
<td>Nazareth</td>
<td>0.1361</td>
<td>1.093</td>
<td>1.858</td>
<td>-1.139</td>
<td>3</td>
</tr>
</tbody>
</table>
5.4 Summary

This chapter explains how to implement the modified HNNs for constrained optimization problem. The penalty functions are used to integrate all constraints into the objective function. Then the problems are solved in the same way as the unconstrained optimization problems. The suggested penalty functions are based on the behavior of diode and can be implemented by an electrical circuit.

The modified HNNs are applied to a linear programming problem and a nonlinear programming problem. They show a good result and faster convergence speed than the traditional HNN.
6.1 Noise Neural Network

HNN can be used to compute a local minimum of a function $E(x)$ defined on a hypercube $[0,1]^n$, when using sigmoid amplifiers or on continuous decision variables when using linear amplifiers (Tank & Hopfield, 1986). However, if $E(x)$ is a nonlinear function or nonconvex function, the system will get trapped in a local minimum, which is near the initial value of $x_i$, because of its gradient descent optimization. Many techniques have been implemented to modify the HNN for global optimization as described in Chapter 2. The simple method that well suited for circuit implementation is diffusion machine.

Let consider the dynamics of a diffusion machine which is described as

$$
\frac{du_i(t)}{dt} = -\frac{\partial E(v(t))}{\partial v_i} dt + \frac{2T}{\sqrt{g'(u_i(t))}} dW_i(t)
$$

(6.1)

where $W_i$ are independent Wiener processes, and $T$ is artificial temperature same as the simulated annealing algorithm. The noise is independent of $E(v(t))$ or the weights of HNN. Thus, any weight adjustment procedure would not affect the performance of diffusion machine (Wong, 1991). For more analog realizations (Biro et al., 1996), Equation (6.1) is corrected and obtained

$$
\frac{du_i(t)}{dt} = -\frac{\partial E(v(t))}{\partial v_i} + \frac{T}{2} \frac{g''(u_i(t))}{(g'(u_i(t)))^2} + \sqrt{\frac{2T}{g'(u_i(t))}} \eta_i(t)
$$

(6.2)
in which \( \eta_i(t) = \frac{\partial W_i}{\partial t} \) are Gaussian white noises. With the continuous decision variables, Equation (6.2) is formulated as

\[
\frac{dx_i(t)}{dt} = -\frac{\partial E(x(t))}{\partial x_i} + \sqrt{2T} \eta_i(t) 
\]  
(6.3)

The energy function of the system, then, will be described as

\[
\tilde{E} = E(x(t)) + \sqrt{2T} \sum_{i=1}^{N} \eta_i(t)x_i(t) 
\]  
(6.4)

The term \( \sqrt{2T} \eta_i(t) \) of Equation (6.3) guarantees the \( x(t) \) to reach the global optimum with probability close to 1. Equation (6.3) coincides with the dynamic equation of HNN, that is added a noise term to shake the system off a local minimum. The noise increases the energy function value and, then, allows the system to quickly escape from some local minima by large random fluctuations in earlier iterations and finally backs to the original dynamic model when the temperature is in tiny value. The global optimization and analog implementation of the diffusion machine are analytically described in (Kesidis, 1995).

The noises injected to the system make the system unstable, even a small intensity of noise. Instead of using continuously maintained noise, pulsed noise (PN) is injected into the system at the certain time to make a cheaper implementation (Mandziuk, 2000). PN are Gaussian noises that are injected into the system at each time \( t = p\tau, p = 0,1,2,... \), and no noise in the system in the period between \( p\tau \) and \( (p+1)\tau \). Hence, the system only performs a gradient descent optimization between consecutive \( p\tau, p = 0,1,2,... \) times.
6.2 Branin Algorithm

Branin (1972) proposed an algorithm that changes the sign of convergence rate of HNN to find the extreme of the system. This algorithm has later been known as the continuous Newton method. (Diener & Schaback, 1990) The algorithm is described as

Step 1 Given a starting point $x_0$

Step 2 Solve the autonomous differential equation

$$\frac{dx}{dt} = -\mathbf{H}_j(x) \nabla f(x)$$

Step 3 If an extreme is found, save the output $x$ and reverse the sign of the autonomous differential equation.

Step 4. Continued Step 2 until the next extreme is found and so on until the termination criterion is satisfied.

Similar to the Branin Algorithm, Jeffrey, and Rosner (1986) analyzed the general differential equation of HNN and change the sign of learning rate to find the extrema.

In term of system simulation, the natural update transformation for HNN is defined as

$$V_i^{n+1} = V_i^n + \lambda_i \left[ \sum_{j=1}^{N} T_{ij} V_j^n + I_i \right] \Delta t$$  \hspace{1cm} (6.5)

where $\lambda_i$ is the gain for the $i$th neuron. If $\Delta t \to 0$, we have the continuous transformation

$$\frac{dV_i}{dt} = \lambda_i \left[ \sum_{j=1}^{N} T_{ij} V_j^n + I_i \right] = -\lambda_i \frac{\partial E}{\partial V_i}$$  \hspace{1cm} (6.6)

Thus,

$$\frac{dE}{dt} = -\sum_{i=1}^{N} \left[ \sum_{j=1}^{N} T_{ij} V_j^n + I_i \right] \frac{dV_i}{dt} = -\sum_{i=1}^{N} \frac{1}{\lambda_i} \left( \frac{\partial E}{\partial V_i} \right)^2 \leq 0$$  \hspace{1cm} (6.7)
From Equation (6.7) the value of an energy function always decreases unless the $\lambda < 0$.

Hence, the algorithm for finding a global minimum is

Step 1 Let the system find a minimum.

Step 2 Reverse the sign of $\lambda$ for a predetermined number of steps.

Step 3 Let the system find a maximum.

Step 4 Reverse the sign of $\lambda$.

Step 5 Let the system find a minimum and keep track of the lowest minimum found.

Step 6 Continue until the solution get trapping or a fixed number of minima has been found.

This algorithm itself starts searching for an extreme from a succeeding minimum or maximum point. Otherwise, the local extrema searching of this algorithm must not simply retrace the previous descent or ascent path. These two algorithms are essential in the modified HNN described in next section.

### 6.3 Modified Hopfield Neural Networks for Global Optimization

Recall that the general dynamic equation for HNN is expressed as

$$
\frac{dx_j(t)}{dt} = -\sum_{i=1}^{n} \mu_i(t) \frac{\partial E(x(t))}{\partial x_j(t)}, \quad x_j(0) = x_j^{(0)}, \quad j = 1, 2, ..., n \quad (6.8)
$$

HNN can perform minimize or maximize an energy function by changing the value of $\mu_i$ to be positive or negative, respectively. Once, HNN converges to a local extremum, it remains at that point because the gradient at the stable points is equal to zero, and make no difference between $x(t)$ and $x(t-\tau)$. Adding a noise term to the dynamic equation of HNN
can intentionally change the gradient value at the local extrema, and thus the neuron’s state moves “uphill” or “downhill” along the energy function. Additionally, noises give a possibility of states not to retrace the previous path, i.e. it randomly changes the function energy and the search direction. Hence, the dynamic equation of neurons is defined as

$$\frac{dx_i(t)}{dt} = \mu_i \left( -\frac{\partial E(x(t))}{\partial x_i} + \eta_i(t) \right)$$  \hspace{1cm} (6.9)

where \( \mu_i \) is the square wave with the magnitude of 1 and -1.

\( \eta_i(t) \) is the pulse noise with Gaussian \( N(0,1) \) distribution.

The above equation finds the path of states along the entire energy function. The local extrema and their energy value have been recorded. The local extrema searching is processes in a specific time or iteration \( T \). After time \( T \), the local extrema comparator is performed, When the latest energy function value is less than or equal to the lowest values recorded in the local extrema searching process, the optimization process is terminated. This modified HNN will itself use the local extrema as its succeeding starting points.

The procedure of the optimization algorithm is as follows:

Step 1. Specify an initial starting point \( x_0 \), a specific time \( T \). Set \( E_m \) is a large positive number.

Step 2. Calculate a local minimum by integration Equation (3.11) with \( \mu = 1 \).

Step 3. Calculate \( E(x) \). If \( E(x) \leq E_m \), set \( E_m = E(x) \); otherwise keep \( E_m \).

Step 4. Calculate a local maximum by integration Equation (3.11) with \( \mu = -1 \).

Step 5. If \( t < T \), go to step 2; otherwise

\[ \text{If } |E(x) - E_m| \leq \varepsilon, \text{ stop; otherwise go to step 2.} \]
The specific time $T$ should be large enough to ensure that the lowest energy has been found. The value of $\mu_i$ is changed to zero or very low positive value to control the system in a stable state, i.e. stop the algorithm when the near-optimal solution is found.

This algorithm can be modified by the proposed BFGS HNN or Nazareth HNN to increase the speed of convergence. Hence, it is possible to find the global minimum faster. The modified methods are formulated as

For BFGS HNN:

\[
\frac{dx}{dt} = \lambda (-\mu_{BFGS} \nabla E(x) + \eta) \quad (6.10a)
\]

\[
\frac{d\mu_{BFGS}}{dt} = H^{-1} \quad (6.10b)
\]

where

\[
H^{-1} = \mu_{BFGS} + (1 + \frac{s^T \mu_{BFGS} z}{s^T z}) \frac{ss^T}{s^T z} - \left( \frac{sz^T \mu_{BFGS} + \mu_{BFGS} z s^T}{s^T z} \right)
\]

\[
s = x(t) - x(t-\tau), \quad y = g(t) - g(t-\tau), \quad g = \nabla E(x)
\]

\[
z = y + t\|g\|s, \quad t = C + \max \left\{ \frac{-s^T y}{\|g\|}, 0 \right\} \|g\|^{-1}, \quad C > 0
\]

$\lambda$ is the square wave with the magnitude of 1 and -1.

$\eta$ is the pulse noise with Gaussian $N(0,1)$ distribution.

For Nazareth HNN

\[
\frac{dx}{dt} = \lambda (-\mu_{CG} \mu_{BFGS} \nabla E(x) + \eta) \quad (6.11a)
\]

\[
\frac{d\mu_{CG}}{dt} = Q \quad (6.11b)
\]

106
\[ \frac{d\mu_{BFGS}}{dt} = H^{-1} \]  

where

\[ Q = I - \frac{s_k y_k^* y_k^* + y_k^* y_k^*}{y_k^* s_k} + (1 + \frac{y_k^* y_k^*}{y_k^* s_k}) \frac{s_k s_k^T}{y_k^* s_k} \]

\[ H^{-1} = H^{-1} - \frac{s_k y_k^* H^{-1} + H^{-1} y_k^* s_k}{y_k^* s_k} + (1 + \frac{y_k^* y_k^* H^{-1}}{y_k^* s_k}) \frac{s_k s_k^T}{y_k^* s_k} \]

\[ s_k = x_{k+1} - x_k, \quad y_k^* = y_k + t_k \|g_k\|s_k, \]

\[ t_k = C + \max \left\{ -\frac{s_k^T y_k}{\|s_k\|^2}, 0 \right\} \|g_k\|^{-1}, \quad C > 0 \]

\[ \lambda \] is the square wave with magnitude of 1 and -1.

\[ \eta \] is the pulse noise with Gaussian \( N(0,1) \) distribution.

The square wave is generated by any nonlinear function generator or square wave generator, while the Gaussian noise can be generated from any hardware Gaussian noise generator.

### 6.4 Simulation Results

The proposed approach is implemented to two nonlinear functions with multiple extrema. The first function is Peaks function, which has an inflection point and two local minima. The second function is the Griewank function, which contains several local minima, and one global minimum. The simulations are performed using MATLAB/Simulink programming. The integration method is the fourth-order Runge-Kutta with an integration step of 0.01.
6.4.1 Peaks function

The equation for the peaks function is

\[ f(x_1, x_2) = 3 \cdot (1 - x_1)^2 \cdot e^{-(x_1^2 - x_2^2)^1} - (2x_1 - 10x_1^3 - 10x_2^5) \cdot e^{(x_1^2 - x_2^2)} - \frac{1}{3} \cdot e^{(-x_1^2 + x_2^2)} \]

As shown in Figure 6.1, there are two local minima: \( f(-1.3473, 0.2045) = -3.0498 \), and \( f(0.2263, -1.6256) = -6.5511 \). The initial conditions for the system are \( x_1(0) = 0 \) and \( x_2(0) = 2 \), and the specific time \( T \) is 50. The algorithm found the optimal solution as \( f(0.2284, -1.626) = -6.551 \). Figure 6.2 shows the path of neuron’s states. Note that the states walk along the surface of the energy function. Figure 6.3 depicts the trajectory of neuron’s states when using the standard HNN. This method converges to the global minimum within 180 seconds. Figure 6.4 and 6.5 illustrate the result from the BFGS HNN and Nazareth HNN for the Peaks function, respectively. The optimal solution can be found within 100 seconds.
Figure 6.1 Peak function
Figure 6.2 The path of neuron’s states for the initial condition $x_1(0) = 0$ and $x_2(0) = 2$
Figure 6.3  The trajectory of neuron’s states from HNN for Peaks function with $x_1(0) = 0$ and $x_2(0) = 2$. 
Figure 6.4  The trajectory of neuron’s states from BFGS HNN for Peaks function with $x_1(0) = 0$ and $x_2(0) = 2$. 
Figure 6.5  The trajectory of neuron’s states from Nazareth HNN for Peaks function with \( x_1(0) = 0 \) and \( x_2(0) = 2 \).
6.4.2 Griewank function

The equation of the Griewank function for two dimensions is defined as

\[ f(x_1, x_2) = \frac{1}{4000} (x_1^2 + x_2^2) - \cos(x_1) \cdot \cos\left(\frac{x_2}{\sqrt{2}}\right) + 1 \]

Figure 6.6 illustrates the surface of the function in a two-dimensional case over the box \([-14,14]^2\). The global minimum is located at the origin with a value of zero. The number of local minima in the range of \([-14,14]^2\) is 31 (Cho, Olivera & Guikema, 2008). Moreover, when increasing the dimension, the number of minima grows exponentially.

The initial condition for the system are \(x_1(0) = 50\) and \(x_2(0) = 50\). The specific time \(T\) is chosen as \(1 \times 10^6\) for HNN, and \(5 \times 10^5\) for BFGS HNN and Nazareth HNN to ensure the global states have been found. The optimal solutions are \(f(3.645 \times 10^{-6}, -1.06 \times 10^{-3}) = 2.812 \times 10^{-7}\) for HNN, \(f(6.22 \times 10^{-4}, -1.42 \times 10^{-4}) = 1.986 \times 10^{-7}\) for BFGS HNN, and \(f(8.924 \times 10^{-4}, -8.822 \times 10^{-4}) = 5.931 \times 10^{-7}\) for Nazareth HNN. Figure 6.7 shows the path of neuron’s states along the Griewank function. The neuron’s trajectory for HNN shows as Figure 6.8. The global solution has been found at the time around \(2.5 \times 10^6\) seconds. Figure 6.9 depicts the trajectory of neurons for BFGS HNN. Figure 6.10 shows the neuron’s trajectory for Nazareth HNN. The optimal points have been found within \(8 \times 10^5\) seconds for BFGS HNN and \(9 \times 10^5\) seconds for Nazareth HNN.
Figure 6.6 The graph of Griewank function
Figure 6.7 The path of neuron’s states along the Griewank function.
Figure 6.8 The trajectory of neuron’s states from HNN for Griewank function with 
\( x_1(0) = 50 \) and \( x_2(0) = 50 \).
Figure 6.9  The trajectory of neuron’s states from BFGS HNN for Griewank function
with $x_1(0) = 50$ and $x_2(0) = 50$. 
Figure 6.10  The trajectory of neuron’s states from Nazareth HNN for Griewank function with $x_1(0) = 50$ and $x_2(0) = 50$. 
6.5 Summary

The traditional HNN is always stuck in a local minimum near the starting point. The local minimum searching is the main feature of HNN because it is the steepest descent method. For the nonconvex functions or the multiple minimum functions, the traditional HNN must be used in combination with a heuristic method, such as simulated annealing, stochastic methods, or genetic algorithms.

We proposed a noise HNN that can be used to find the global minimum of a nonconvex function. The modified HNNs can be used to increase the time to find the global minimum. The proposed HNN model performs well with Peaks and Griewank function. However, the time to find the global minimum may vary depending on the random number of Gaussians noise. BFGS HNN and Nazareth HNN can be implemented with the noise and sign changing algorithm to find the global minimum of the functions. Additionally, from the results the optimal solutions have been found faster than the standard HNN model.
CHAPTER 7
INTEGRATED PROCESS PLANNING AND SCHEDULING

7.1 Job Shop Scheduling with Hopfield Neural Networks

Production scheduling is defined as a nondeterministic polynomial time complete (NP-complete) problems. As the size of scheduling problem increases, the computation time of the problem performs in a manner of exponential growth. In general, production scheduling allocates resources according to the process plan. An efficient scheduling is an optimal plan depending on production goal and related constraint conditions. The goals of scheduling are shortest production time and minimum production cost. The constraints of scheduling are, for example, operation time, product priority, equipment implementation, production rate, production batch, and due date. These goals and constraints affect significantly to the production management and organization.

Two models of Hopfield neural network (HNN) have been presented for solving the job-shop scheduling problem. One model is derived based on the HNN for Traveling Salesman Problem (TSP) (Foo & Takefuji, 1988b; Vaithyanathan & Ignizio, 1992; Satake et al., 1994; Wang et al., 2003; Chen, Lo & Huang, 2007; Chen, 2011). Another model is based on the mix-integer linear programming (MILP) which is related to the Hopfield’s linear programming circuit (Zhou, Cherkassky, Baldwin & Olson, 1991; Foo et al., 1994; Fnaiech et al., 2012; Kechadi, Low & Goncalves, 2013). MILP neural networks need less number of neurons and interconnections than TSP-type neural networks (Zhou et al., 1991). Therefore, MILP neural networks are preferred in hardware implementation.

Foo and Takefuji (1988a) modified the Hopfield networks to solve job-shop scheduling. The job-shop scheduling is a resource-allocation problem which machines are
resources and jobs are basic tasks. The $n$ jobs have to operate on $m$ machines in a prescribed order under certain restrictive assumptions. A feasible schedule is to all operations of each job can be placed on one-time axis in precedence order and without overlap. The formulation of a job-shop schedule can be described as follows.

Let $s_{ik}$ be the starting time of job $i$ on machine $k$, and $t_{ijk}$ be the processing time for operation $(i, j, k)$. Assuming operation $(i, j-1, h)$ precedes $(i, j, k)$, then the precedence constraints are:

\[ s_{ik} - s_{ih} \geq t_{i,j-1,h}, \quad 1 < j < m, \quad 1 < i < n \quad (7.1) \]

All starting times must be positive values. Hence the positive results constraints are

\[ s_{ik} \geq 0, \quad 1 < i < n \quad (7.2) \]

Another condition is that two operations are not performed simultaneously, i.e. two operations cannot be processed by the same machine at the same time. If operation $(p, q, k)$ starts after the completion of $(i, j, k)$, then

\[ s_{pk} - s_{ik} \geq t_{gk} \quad (7.3) \]

Conversely, if operation $(p, q, k)$ precedes the operation $(i, j, k)$, then

\[ s_{ik} - s_{pk} \geq t_{pqk} \quad (7.4) \]

Using a “zero-one” variable $y_{ipk}$ to defined the sequence of operation, i.e. $y_{ipk} = 1$ if job $i$ precedes job $p$ on machine $k$, and $y_{ipk} = 0$, otherwise. The operation sequencing constraints are

\[ s_{pk} - s_{ik} + H * (1 - y_{ipk}) \geq t_{gk} \quad (7.5) \]

\[ s_{ik} - s_{pk} + H * y_{ipk} \geq t_{pqk} \quad (7.6) \]
where constant $H$ is an arbitrary positive number greater than the maximum value of all processing times $t_{ijk}$. Hence, the formulation of job-shop problem is defined as minimizing the cost function

$$\sum_{i=1}^{n} s_{ik}$$  \hspace{1cm} (7.8)

Subject to

$$s_{ik} - s_{ih} \geq t_{i,j-1,h} \hspace{1cm} 1 < j < m, \hspace{0.5cm} 1 < i < n$$  \hspace{1cm} (7.9)

$$s_{pk} - s_{ik} + H * (1 - y_{ipk}) \geq t_{ijk} \hspace{1cm} i \geq 1, p \leq n, \hspace{0.5cm} 1 \leq k \leq m$$  \hspace{1cm} (7.10)

$$s_{ik} - s_{pk} + H * y_{ipk} \geq t_{pqk} \hspace{1cm} i \geq 1, p \leq n, \hspace{0.5cm} 1 \leq k \leq m$$  \hspace{1cm} (7.11)

$$s_{ik} \geq 0 \hspace{1cm} 1 < i < n$$  \hspace{1cm} (7.12)

where $k_i$ is the machine for last operation of job $i$.

There are $mn$ constraints of the precedence constraints and $mn(n-1)$ constraints for the operation sequencing constraints. Therefore, the total number of constraints are $mn^2$ constraints. Otherwise, there are $mn(n+1)/2$ variables, i.e. $mn$ number of $S_{ik}$ and $mn(n-1)/2$ number of $y_{ipk}$.

The modified Hopfield and Tank network for job-shop scheduling is shown in Figure 7.1. The model adds nonlinear h-amplifiers representing the zero-one variable $Y_i$. The g-amplifiers represent the starting time $V_i$ of each operation after minimization. The vector $A$ is $N$ coefficients for $N$ variables of variable vector $V$. The components of $A$ are the input currents fed into the g- and h-amplifiers. The input time constant $\rho_i C_i$ acts as a memory element for each g- and h-amplifier. The f-amplifiers represent constraints satisfaction which has $M$ outputs $\psi_j$.
Figure 7.1 The MILP network for job-shop scheduling (Foo et al., 1994).
The dynamics of the g-amplifier can be defined as:

\[
C_i \frac{dU_i}{dt} = -A_i - \frac{U_i}{R_i} - \sum_{j}^{M} D_{ji} f(D_j \cdot V - B_j) \quad (7.13)
\]

and the dynamics of the h-amplifier is:

\[
C_i \frac{dX_i}{dt} = -A_i - \frac{X_i}{R_i} - \sum_{j}^{M} D_{ji} f(D_j \cdot Y - B_j) \quad (7.14)
\]

An energy function for the entire circuit is in the form of

\[
E = \sum_{i}^{P} A_i V_i + \sum_{i}^{P} \frac{1}{R_i} \int_{0}^{V_i} g^{-1}(V) dV + \sum_{i}^{P} \sum_{j}^{M} f(z) dz \\
+ \sum_{i}^{P} \frac{1}{R_i} \int_{0}^{V_i} h^{-1}(Y) dY + \sum_{i}^{P} \sum_{j}^{M} f(w) dw \quad (7.15)
\]

Where

\[U = g^{-1}(V)\]
\[X = h^{-1}(Y)\]
\[z = D_j \cdot V - B_j\]
\[w = D_j \cdot Y - B_j\]
\[P = mn\], the total number of \(s_{ik}\)
\[Q = mn(n-1)/2\], the total number of \(y_{ipk}\)

The energy function \(E\) is always a decreasing function which seeks out a minimum in \(E\) and stops.

From the sequencing constraints equation (7.5) and (7.6), the modified HNN will add \(mn(n-1)/2\) neurons more to the system. Zhou et al. (1991) proposed a model that remains the number of neurons same as the number of the starting time variables of all operations.
The model uses diodes to feedback the sequencing constraints signal to the system. The energy function $E$ of the circuit is

$$E = \sum S_{ik} + \sum H_1 \ast F_1(S_{ik} - S_{i,k+1} + t_{ik}) + H_2 \ast F_1(-S_{i1})$$

$$+ \sum_{1 \leq k < j, 1 \leq p \leq k, 1 \leq i \leq j \leq m, m(i,k) = m(j,p)} H_3 \ast F_2(S_{ik} - S_{jp} + t_{ik}) \ast F_2(S_{jp} - S_{ik} + t_{jp}) \quad (7.16)$$

where $H_1$, $H_2$, and $H_3$ are large positive constants.

$$F_1(x) = \begin{cases} e^{bx} - bx & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

$$F_2(x) = \begin{cases} e^{bx} - 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

$b$ is a constant depending on the characteristic of the diode.

The dynamic of $S_{ik}$ is obtained by taking the derivative of Equation (7.16) over $S_{ik}$, that is

$$\frac{dS_{ik}}{dt} = [-1] - H_1 \ast b(\exp(b(S_{ik} - S_{i,k+1} + t_{ik})) - 1)$$

$$+ H_2 \ast b(\exp(b(-S_{i1}) - 1)$$

$$- H_3 \ast b(\exp(b(S_{ik} - S_{jp} + t_{ik})$$

$$+ H_3 \ast b(\exp(b(S_{jp} - S_{ik} + t_{jp})) \quad (7.17)$$

where [-1] is only for the last operation of each job, and the third term is for the first operation of each job.

The key feature of this modified HNN is that the multiplication operation is implemented as an addition operation. Hence, the number of neurons still be the same as the number of the starting time variables.
Instead of using a physical circuit of Hopfield network, the simulation-based
network may be implemented to find a solution to job-shop scheduling problems.
Liansheng et al. (2000) proposed a unified algorithm to search an optimal solution for the
Hopfield network energy function. The job-shop scheduling formulation and the unified
algorithm are described as follow.

Let $s_{i,k}$ be the starting time of operation $k$ for job $i$
$s_{i,1}$ be the starting time of the first operation for job $i$
$k_i$ be the last operation for job $i$
$t_{i,k}$ be the production time of operation $k$ for job $i$
$m_{i,k}$ be the machine tool assigned for job $i$ machining operation $k$

The goal function is the minimum total idle time of all machine tool. Therefore, the proper
function may be defined as

$$\min \sum_i s_{i,k} \quad \text{or} \quad \min \sum_i (s_{i,k} + t_{i,k})$$

and

$$\min \left[ \max\left(s_{i,k}\right) \right] \quad \text{or} \quad \min \left[ \max\left(s_{i,k} + t_{i,k}\right) \right]$$

A set of constraints are given as

$$s_{i,k} - s_{i,k+1} + t_{i,k} \leq 0 \quad (7.18)$$

$$s_{i,1} \geq 0 \quad (7.19)$$

$$\min \left[ \left(s_{i,k} - s_{j,p} + t_{i,k}\right), \left(s_{j,p} - s_{i,k} + t_{j,p}\right) \right], \quad 1 \leq i, j \leq n, 1 \leq k, p \leq k \quad (7.20)$$

Equation (7.18) is the precedence constraint. Equation (7.19) ensure the first operation of
each job is nonnegative. Equation (7.20) forces two jobs not to be produced simultaneously
on one machine.
Then, the job-shop schedule can be expressed by the energy function $E$ based on the Hopfield network:

$$E = D \cdot G(s_{i,k}) + \sum_j \sum_k A \cdot F_i \left(s_{i,k} - s_{i,k+1} + t_{i,k}\right)$$

$$+ \sum_i B \cdot F_2 \left(-s_{i,1}\right) + \sum_i \sum_{(j)} \sum_{(p)} C \cdot \min \left[F_3 \left(s_{i,k} - s_{j,p} + t_{i,k}\right), F_3 \left(s_{j,p} - s_{i,k} + t_{j,p}\right)\right]$$

(7.21)

where $G(x)$ is the goal function.

$F_{1,2,3}(x)$ are the penalty functions.

$A, B, C, \text{and} D$ are the weight parameters.

The dynamic state equation of function $E$ can be described as:

$$\frac{\partial E}{\partial s_{i,k}} = -\frac{du_{i,k}}{dt}$$

(7.22)

where $u$ is the neuron internal-state in the neural network at time $t$. The time derivative of the internal-state $u_{i,k}$ is the partial differential of $E$ by $s_{i,k}$ when $k = 1, k, k_i$ with the goal function $G(s_{i,k}) = \min[\max_i(s_{i,k})]$. Then, the neural network state equation of the job-shop schedule is

$$\frac{du_{i,k}}{dt} = D(-1)\delta(s_{i,k} - \max_i(s_{i,k}))$$

$$- A \left[f \left(s_{i,k} - s_{i,k+1} + t_{i,k}\right) - f \left(s_{i,k-1} - s_{i,k} + t_{i,k-1}\right)\right]$$

$$- B \cdot (s_{i,k})\delta(k - 1)$$

$$- \sum_{m(i,k)=m(j,p)} C \cdot \min \left[f \left(s_{i,k} - s_{j,p} + t_{i,k}\right), f \left(s_{j,p} - s_{i,k} + t_{j,p}\right)\right]$$

(7.23)

where
\[ f(x) = \frac{\partial (F(x))}{\partial x}, \quad x > 0 \]

\[ \delta(x) = \begin{cases} 
1, & x = 0 \\
0, & x \neq 0
\end{cases} \]

The value of parameters \( A, B, C, \) and \( D \) affect the convergence of the function. Analysis of each parameter is explained as follows. Firstly, because the starting time for the first operation of each job must be considered before the next operation, \( B > A, \) and \( B > C. \) Next, the constraint for avoiding two jobs produced on one machine tool simultaneously must be considered. Therefore, \( C > A. \) In conclusion, \( B > C > A. \)

The state of \( s_{i,k} \) is obtained in the following way:

\[ u_{\text{new}} = u_{\text{old}} + \frac{du_{i,k}}{dt} \times \Delta t \quad (7.24) \]

where \( u_{\text{new}} \) and \( u_{\text{old}} \) indicate the new state and old state of neuron \( s_{i,k}, \) respectively.

The unified algorithms used to find the minimization of the energy function \( E \) are as follows

Step 1 To initialize the parameters \( A, B, C, D \) and the increment step \( \Delta t. \)

Step 2 To set up the starting time for job \( i. \) The first operation’s starting time \( s_{i,1} = 0. \) The starting time \( s_{i,k} \) of operation \( k \) and its subsequent operations are accumulated: \( s_{i,k+1} = s_{i,k} + t_{i,k} \) \( (i = 1,2,\ldots,n,k = 1,2,\ldots,k-1). \)

Step 3 To determine \( E(t_0). \)

Step 4 To determine \( \frac{du_{i,k}}{dt}. \)

Step 5 To compute new \( u_{i,k}(t + \Delta t) \) according to Equation (7.24).

Step 6 Let \( s_{i,k} = u_{i,k}. \)
Step 7 To obtain $E(t_0 + \Delta t)$.

Step 8 To obtain $\Delta E = E(t_0 + \Delta t) - E(t_0)$.

Step 9 Let $(t_0 + \Delta t) \rightarrow t_0$.

Step 10 To check $\Delta E < \varepsilon$, else go to step 3, then output $s_{i,k}$.

7.2 Integrated Process Planning and Scheduling Modeling

In general, the goal of integrated process planning and scheduling (IPPS) is to seek out optimal schedules with the constraints of operation priority, processing time, and the set of alternative resources. A mathematical model of IPPS should be defined suitably to find the optimal solution.

Many researchers have proposed IPPS models in a different way depending on the optimization methods as described in the literature review. The best choice that can be implemented to a Hopfield neural network is an integer programming model.

Moon, Lee, Jeong, and Yun (2008) developed an IPPS model based on a mixed integer programming. The problem definition is that there are $n$ jobs to be processed using $m$ resources with alternative operation sequences. A process plan representation should contain all possible precedence constraints that occur among the planning and processing decisions. All jobs are operated according to a determined process sequence with resource selection in the process planning. Each job requires a number of operations which are performed on some alternative resources. The completion time for all jobs is calculated by summing machining times, setup times, transportation times, and waiting times of all operations. The makespan or the total length of time for finishing all jobs is an important criterion when the number of orders is finite.
Suppose that:

$i, j$ is the index for the job, $i, j = 1, \ldots, I$, where $I$ is the number of jobs

$k, l$ is the index for the operation, $k, l = 1, \ldots, K$, where $k$ is the number of operations

$p, q$ is the index for the resource, $p, q = 1, \ldots, M$, where $M$ is the number of resources

$K_i$ is the number of operations in job $i$

$R_i$ is the set of precedence relations of two operations, $R_i = \{r_{ij} | \forall j = 1, \ldots, J_i\}$

where $r_{ij} = \langle k, l \rangle$, $J_i$ is the number of precedence relations for job $i$, and $k$ and $l$ are indices for operations

$B_i$ is the set of pairs of operations for job $i$, $B_i = \{r_{ij} | \forall j = 1, \ldots, J_i\}$, where $r_{ij} = \langle k, l \rangle$ or $r_{ij} = \langle l, k \rangle$ in any job, $J_i$ is the number of all pairs of two operations without precedence relations for job $i$, and $k$ and $l$ are indices for operations

$G_p$ is the set of operations to be performed on resource $p$

$p_{ikp}$ is the processing time for operation $k$ of job $i$ on resource $p$

$M$ is an arbitrarily large positive number

\[ d_{ilk} = \begin{cases} 1, & \text{if operation } k \text{ precedes operation } l \text{ of job } i, \\ 0, & \text{otherwise} \end{cases} \]

$x_{ikp}$ is the completion time of operation $k$ of job $i$ on resource $p$

\[ y_{ilk} = \begin{cases} 1, & \text{if operation } p \text{ is selected for operation } k \text{ of job } i, \\ 0, & \text{otherwise} \end{cases} \]
The objective of the model is to minimize the makespan. Hence, the mixed integer programming model for solving the integrated model is:

Minimize  \[ F = \max_{\forall i, k \text{ and } p} \{ x_{ikp} \} \]  \hspace{1cm} (7.25)

subject to

\[ x_{ilq} - x_{ikp} \geq p_{ilp} \quad \forall (k, l) \in R_i, i, p \text{ and } q \]  \hspace{1cm} (7.26)

\[ x_{ilq} - x_{ikp} + M(1-d_{ilk}) \geq p_{ilq} \quad \forall (k, l) \in B_i, i, p \text{ and } q \]  \hspace{1cm} (7.27)

\[ x_{ikp} - x_{ilq} + M d_{ilk} \geq p_{ikp} \quad \forall (k, l) \in B_i, i, p \text{ and } q \]  \hspace{1cm} (7.28)

\[ x_{jlq} - x_{ikp} + M(1-d_{ikl}) \geq p_{jlq} \quad \forall (k, l) \in G_p, i, p, q \text{ and } i \neq j \]  \hspace{1cm} (7.29)

\[ x_{ikp} - x_{jlq} + Md_{ikl} \geq p_{ikp} \quad \forall (k, l) \in G_p, i, p, q \text{ and } i \neq j \]  \hspace{1cm} (7.30)

\[ \sum_{p=1}^{M} y_{ikp} = 1 \quad \forall i \text{ and } k \]  \hspace{1cm} (7.31)

\[ x_{ikp} \geq \begin{cases} p_{ikp}, & \text{for } i, k \in B_i, p \\ 0, & \text{for all other } i, k \end{cases} \hspace{1cm} (7.32) \]

\[ y_{ikp} \in \{0,1\} \quad \forall i, k, \text{ and } p \hspace{1cm} (7.33) \]

The first constraint means that the operations of each job are processed according to the precedence required. The second and third constraints ensure that any two operations belonging to the same job cannot be processed at the same time. The fourth and fifth constraint conditions ensure that a resource cannot process more than one job at the same time. The sixth constraint ensures that only one resource for each operation should be selected. The seventh and eight constraints imply non-negativity and integrality of the corresponding variables.
This IPPS model have been solved by using an evolutionary search approach. The pseudocode of the approach is shown as below.

Procedure: Evolutionary search approach

Begin

\[ t \leftarrow 0; \]

Initialize parent population \( P(t) \) using operation number and resource number;

Evaluate \( P(t) \)

While (not termination condition) do

Recombine \( P(t) \) to yield \( C(t) \) by crossover and mutation;

Evaluate \( C(t) \)

Select \( P(t + 1) \) from \( P(t) \) and \( C(t) \) by selection strategy;

\[ t \leftarrow t + 1; \]

End

End

The chromosomes contain the sequences of operations and selected resources. The researchers use a topological sort procedure to produce an operation sequence. Corresponding resources are selected randomly from the set of alternative resource for each operation. The fitness evaluation is to minimize the time required for completing all of the operation or the makespan. The selection process is an elitist strategy. Two points swapping crossover and the random mutation are the crossover operator and mutation operator, respectively, for the proposed method. This evolutionary search approach shows an efficient result comparing to traditional approaches.
The objective or criteria for scheduling is not only the minimization of makespan but also be other performances such as job tardiness (i.e., how long after the due date a job was completed) and the machine utilization. Li, Gao, Shao, Zhang, and Wang (2010) proposed the mathematical models for IPPS with a set of objective functions. The model can be optimized in the mono-objective problems or the multi-objective problems. The descriptions of the model are as follows.

Suppose that the IPPS have the following assumptions.

1) Jobs are independent. Job preemption or running job is not allowed, and each machine can handle only one at a time.

2) The different operation of one job cannot be processed simultaneously.

3) All jobs and machines are available at time zero simultaneously.

4) After a job is processed on the machine, it is immediately transported to the next machine in its process, and the transmission time is assumed to be negligible.

5) Setup time for the operations on the machines is independent of the operation sequence and is included in the processing times.

Some notations for the model are:

- $N$ is the total number of jobs
- $M$ is the total number of machines
- $d_i$ is the due date of job $i$
- $\omega_i$ is the weight of job $i$ indicating a priority factor.
- $G_i$ is the total number of alternative process plans of job $i$
- $o_{ijl}$ is the $j$th operation in the $l$th alternative process plan of the job $i$
\( P_{il} \) is the number of operation in the \( l \)th alternative process plan of the job \( i \)

\( k \) is the alternative machine corresponding to \( o_{ijl} \)

\( t_{ijlk} \) is the processing time of operation \( o_{ijl} \) on machine \( k \), \( t_{ijlk} > 0 \)

\( c_{ijlk} \) is the earliest completion time of operation \( o_{ijl} \) on machine \( k \)

\( c_i \) is the completion time of job \( i \)

\( L_i \) is the lateness of job \( i \)

\( T_i \) is the tardiness of job \( i \)

\( E_i \) is the earliness of job \( i \)

\( \nu_{ijlk} \) is the processing cost of operation \( o_{ijl} \) on machine \( k \)

\( A \) is a very large positive number

\[ U_i = \begin{cases} 1, & \text{if } c_i > d_i \\ 0, & \text{otherwise} \end{cases} \] is the unit penalty of job \( i \)

\[ X_{il} = \begin{cases} 1, & \text{the } l \text{th alternative process plan of job } i \text{ is selected} \\ 0, & \text{otherwise} \end{cases} \]

\[ Y_{gijpqsk} = \begin{cases} 1, & \text{the operation } o_{gj} \text{ precedes the operation } o_{pqsk} \text{ on machine } k \\ 0, & \text{otherwise} \end{cases} \]

\[ Z_{ijk} = \begin{cases} 1, & \text{if machine } k \text{ is selected for } o_{ijl} \\ 0, & \text{otherwise} \end{cases} \]

Hence, the mathematical model of IPPS is:

Objectives:

Minimizing the makespan:

\[
\min \text{ makespan} = \max \left\{ c_{ijlk} x_{il} z_{ijk} \right\}
\]

\[ \forall i \in [1,N], \forall j \in [1,P], \forall l \in [1,G], \forall k \in [1,M] \] (7.34)
Minimizing the total processing cost:

$$\min \sum_{i=1}^{N} \sum_{j=1}^{P_i} \left( v_{ijkl} \times X_{il} \times Z_{ijkl} \right)$$

$$\forall i \in [1, N], \forall j \in [1, P_i], \forall l \in [1, G_i], \forall k \in [1, M]$$  \hspace{1cm} (7.35)

Minimizing the lateness:

$$\min L_{\text{max}} = \max \{L_1, L_2, \ldots, L_N\}, \quad L_j = c_j - d_i, \quad \forall i \in [1, N]$$  \hspace{1cm} (7.36)

Minimizing the total weighted tardiness:

$$\min \sum_i \omega_i T_i, \quad T_i = \max \{c_i - d_i, 0\}, \quad \forall i \in [1, N]$$  \hspace{1cm} (7.37)

Minimizing the weighted number of tardy jobs:

$$\min \sum_i \omega_i U_i, \quad \forall i \in [1, N]$$  \hspace{1cm} (7.38)

Minimizing the total earliness plus the total tardiness:

$$\min \sum_i E_i + \sum_i T_i, \quad E_i = \max \{d_i - c_i, 0\}, \quad \forall i \in [1, N]$$  \hspace{1cm} (7.39)

Subject to:

For the first operation in the alternative process plan $l$ of job $i$:

$$\left( c_{ili} \times Z_{ili} \times X_{il} \right) + A \left( 1 - X_{il} \right) \geq \left( t_{ili} \times Z_{ili} \times X_{il} \right)$$

$$\forall i \in [1, N], \forall l \in [1, G_i], \forall k \in [1, M]$$  \hspace{1cm} (7.40)

For the last operation in the alternative process plan $l$ of job $i$:

$$\left( c_{iopl} \times Z_{iopl} \times X_{il} \right) + A \left( 1 - X_{il} \right) \leq \text{makespan}$$

$$\forall i \in [1, N], \forall l \in [1, G_i], \forall k \in [1, M]$$  \hspace{1cm} (7.41)

The different operations of one job cannot be processed simultaneously:
\[
(c_{ijk} \times Z_{ijk} \times X_{il}) - (c_{(i-1)jk} \times Z_{(i-1)jk} \times X_{il}) + A(1 - X_{il}) \geq (t_{ijk} \times Z_{ijk} \times X_{il})
\]

\[\forall i \in [1,N], \forall j \in [1,P_i], \forall l \in [1,G_i], \forall k \in [1,M]\] (7.42)

Each machine can handle only one job at a time:

\[
(c_{pqsk} \times Z_{pqsk} \times X_{ps}) - (c_{ijk} \times Z_{ijk} \times X_{il}) + A(1 - X_{il}) + A(1 - X_{ps})
\]

\[+ A(1 - Y_{ijlpqs} \times Z_{ijlk} \times Z_{pqsk} \times X_{il} \times X_{ps}) \geq (t_{pqsk} \times Z_{pqsk} \times X_{ps})\] (7.43)

\[
(c_{ijk} \times Z_{ijk} \times X_{il}) - (c_{pqsk} \times Z_{pqsk} \times X_{ps}) + A(1 - X_{il}) + A(1 - X_{ps})
\]

\[+ A(Y_{ijlpqs} \times Z_{ijlk} \times Z_{pqsk} \times X_{il} \times X_{ps}) \geq (t_{ijk} \times Z_{ijk} \times X_{il})\]

\[\forall i \in [1,N], \forall j \in [1,P_i], \forall l \in [1,G_i], \forall k \in [1,M]\] (7.44)

Only one alternative process plan can be selected of job \(i\):

\[\sum_l X_{il} = 1 \quad \forall i \in [1,N]\] (7.45)

Only one machine for each operation should be selected:

\[\sum_{k=1}^M Z_{ijk} = 1 \quad \forall i \in [1,N], \forall j \in [1,P_i], \forall l \in [1,G_i]\] (7.46)

There is only one precedence relation between two operations in a scheduling plan:

\[Y_{ijlpqs} \times Z_{ijlk} \times Z_{pqsk} \times X_{ps} \times X_{il} \leq 1\] (7.47)

\[\left(Y_{ijlpqs} \times X_{il}\right) \leq \left(Z_{ijlk} \times X_{il}\right)\] (7.48)

\[\left(Y_{ijlpqs} \times X_{ps}\right) \leq \left(Z_{pqsk} \times X_{ps}\right)\]

\[\forall i, p \in [1,N], \forall j, q \in [1,P_i], \forall l, s \in [1,G_i], \forall k \in [1,M]\] (7.49)

\[\sum_i \sum_j \sum_l \left(Y_{ijlpqs} \times Z_{ijlk} \times X_{il}\right) = \sum_{o_{ik}} Z_{o_{im}} \]

137
\forall i \in [1,N], \forall j \in [1,P_i], \forall l \in [1,G_j], \forall k \in [1,M] 
(7.50)

where \( \sum_{o_{pqk}^{-1}} Z_{o_{pqk}} \) means the total number of operations before \( o_{pqk} \) on machine \( k \); \( o_{k1} \)
means the first operation on machine \( k \); \( o_{pqk} \) means the current operation on machine \( k \).

The completion time of each operation should be either positive or zero:

\( c_{ijkl} \times Z_{ijkl} \times X_{ij} \geq 0 \quad \forall i \in [1,N], \forall j \in [1,P_i], \forall l \in [1,G_j], \forall k \in [1,M] \)
(7.51)

The above IPPS model has been solved based on an evolutionary approach. Each chromosome represents the alternative process plan part and job-machine part. The selection process is the tournament selection scheme. The crossover operator is initially swapped the different alternative process plan from the parent chromosomes. Then the job-machine part is swapped corresponding to the swapped alternative process plan to avoid infeasible offspring chromosomes. Three mutation operators have been chosen randomly: two-point swapping mutation, changing one job’s alternative process plan, and the mutation of alternative machines. This evolutionary approach resulted in an efficient method for optimizing complex and large-scale problems.

**7.3 Modified Hopfield Neural Networks for the IPPS problem**

**7.3.1 IPPS problem definition**

The IPPS problem can be defined as follow (Guo et al., 2009):

Given a set of \( N \) jobs, and each job has a number of operations \( K \). The jobs are processed on \( M \) resources with alternative process plans. The plans include the operations of the jobs with the corresponding resources. Each operation can be processed on different resources. Consequently, the processing time of each operation alters to the resource
performance. The objective of the problem is to select suitable resources and sequence the operations satisfying the corresponding objectives and constraints.

The assumptions for the IPPS problems are (Kim, Park & Ko, 2003; Li & McMahon, 2007; Zhang & Wong, 2015):

1. Processing is non-preemptive or different job has the same priority.
2. Each resource can handle only one operation at a time.
3. It is not allowed to produce more than one operation of the same job at the same time.
4. All jobs and machines are available at $t = 0$, simultaneously.
5. The transmission time is assumed to be negligible. After a job is processed on a resource, it is immediately transported to the next machine in its routing.
6. Setup time for the operations on the machines is independent of the operation sequence and is included in the processing times of the corresponding operation.
7. Resources are continuously available for manufacturing.

### 7.3.2 IPPS model I

From the previous sections, IPPS can be formulated in an integer programming model that is possible to solve by an HNN. The proposed IPPS for HNN is defined as

Suppose that:

- $i, j$ the index for the job, $i, j = 1, \ldots, N$, where $N$ is the number of jobs
- $k, l$ the index for the operation, $k, l = 1, \ldots, K$, where $K$ is the number of operations
\( p, q \) the index for the resource, \( p, q = 1, \ldots, M \), where \( M \) is the number of resources

\( t_{ikp} \) the processing time for operation \( k \) of job \( i \) on resource \( p \)

\( S_{ikp} \) the starting time of operation \( k \) of job \( i \) on resource \( p \)

\( S_{i, p} \) the starting time of the last operation of job \( i \) on resource \( p \)

\( Y_{ikp} = \begin{cases} 1, & \text{if resource } p \text{ is selected for operation } k \text{ of job } i \\ 0, & \text{otherwise} \end{cases} \)

The objective of the model is to minimize the makespan. Hence, the mixed-integer nonlinear programming model for solving the IPPS model is:

Minimize \( \sum_{i=1}^{N} (S_{ikp} Y_{ikp} + t_{ikp} Y_{ikp}) \) (7.52)

subject to

\[ S_{ikp} Y_{ikp} Y_{i(k+1)q} - S_{i(k+1)q} Y_{ikp} Y_{i(k+1)q} + t_{ikp} Y_{ikp} Y_{i(k+1)q} \leq 0 \] (7.53)

\[ S_{i, p} Y_{i, p} \geq 0 \] (7.54)

\[ (S_{ikp} Y_{ikp} Y_{jlp} - S_{jlp} Y_{ikp} Y_{jlp} + t_{ikp} Y_{ikp} Y_{jlp}) \lor (S_{jlp} Y_{ikp} Y_{jlp} - S_{ikp} Y_{ikp} Y_{jlp} + t_{jlp} Y_{ikp} Y_{jlp}) \] (7.55)

\[ \sum_{p=1}^{M} Y_{ikp} = 1 \] (7.56)

\[ Y_{ikp} \in \{0,1\} \] (7.57)

Constraint (7.53) is the precedence constraint to ensure that two operations of each job cannot be processed at the same time. Constraint (7.54) implies the starting time of the first operation of each job must be at least zero. Constraint (7.55) ensures that a resource cannot process more than one job simultaneously. Constraint (7.56) means that only one resource
can perform in each operation. The last constraints imply integrality of the corresponding variables.

Constraint (7.56) can be transformed into two inequality constraints, which are,

$$\sum_{p=1}^{M} Y_{ikp} - 1 \leq 0 \quad (7.58a)$$

and

$$-\sum_{p=1}^{M} Y_{ikp} + 1 \leq 0 \quad (7.58b)$$

The constraint (7.57) is discrete. It is impossible to compute the partial derivative $$\frac{\partial E(S_{ikp}, Y_{ikp})}{\partial Y_{ikp}}$$ (Tateishi & Tamura, 1994; Galan-Marin & Munoz-Perez, 2001). Hence, any gradient-based optimization methods cannot solve the mixed integer programming problems. A solution to dealing with the difficulty is to relax the discrete variables to the continuous variables (Shandiz & Mahdavi-Amiri, 2011; Yu, Teo & Bai, 2013; Lucidi & Rinaldi, 2013). The discrete variables are changed to $$Y_{ikp} \in [0,1]$$, and a constraint is added to the model. Some forms of the constraint are

$$Y(1-Y) \quad \text{or} \quad 1 - \cos 2\pi Y$$

From the Hopfield’s A/D converter (Tank & Hopfield, 1986), the variables in the circuits have a sigmoid monotonic input-output relation, which is the continuous variables. The energy function of the converter is added the digital representation terms in the form

$$-\frac{1}{2} \sum_{i=0}^{n} (2^i)^2 [V_i(V_i - 1)] \quad (7.59)$$

This term has minimal value when, either $$V_i = 1$$ or $$V_i = 0$$. Consequently, the network would be forced to the minimum energy when $$V_i = 1$$ or 0. Foo and Takefuji (1988a) solve an integer linear programming by treating the zero-one variables as linear variables. Then, the
variables have been determined by very high-gain sigmoid amplifiers. This technique need not require any additional energy terms or penalty terms.

Therefore, constraint (7.57) can be implemented by using high-gain sigmoid amplifiers. Thus, the variables $Y_{ikp}$ have been relaxed to an interval $[0,1]$, i.e. $Y_{ikp} \in [0,1]$, with a form of sigmoid transfer function as

$$f(x) = \frac{1}{1 + e^{-a(x-c)}} \quad (7.60)$$

where $a$ is the steepness of the curve, and $c$ is the sigmoid’s midpoint.

Therefore, the energy function $E$ of our IPPS is

$$E = A \sum_{i=1}^{N} (S_{ik,p} Y_{ik,p} + t_{ik,p} Y_{ik,p}) + \sum_{i=1}^{N} \sum_{k=1}^{K-1} H_1 F_1 (S_{ikp} Y_{ikp} Y_{i(k+1)q} - S_{i(k+1)p} Y_{ikp} Y_{i(k+1)q} + t_{ikp} Y_{ikp} Y_{i(k+1)q})$$

$$+ \sum_{i=1}^{N} H_2 F_2 (-S_{i1p} Y_{i1p})$$

$$+ \sum_{1 \leq k \leq K; 1 \leq l \leq M; 1 \leq q < j \leq N \atop p(i,k) = q(j,l)} \left[H_3 F_2 (S_{ikp} Y_{ikp} Y_{jlp} - S_{jlp} Y_{ikp} Y_{jlp} + t_{ikp} Y_{ikp} Y_{jlp}) \times F_2 (S_{jlp} Y_{ikp} Y_{jlp} - S_{ikp} Y_{ikp} Y_{jlp} + t_{jlp} Y_{ikp} Y_{jlp}) \right]$$

$$+ \sum_{i=1}^{N} \sum_{k=1}^{K} H_4 F_1 \left[ \sum_{p=1}^{M} Y_{ikp} - 1 \right] + \sum_{i=1}^{N} \sum_{k=1}^{K} H_4 F_1 \left[ -\sum_{p=1}^{M} Y_{ikp} + 1 \right] \quad (7.61)$$

where $A$, $H_1$, $H_2$, $H_3$, and $H_4$, are constants

$$F_1(x) = \begin{cases} e^{bx} - bx & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

$$F_2(x) = \begin{cases} e^{bx} - 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

$b$ is a constant depending on the characteristic of the diode.
As mention in (Zhou et al., 1991), the starting time variables should be scaled down to prevent overflow in the exponential functions during simulation. In our model, we propose $F_1(x)$ and $F_2(x)$, as

$$F_1(x) = \begin{cases} e^{-bx} + bx & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

$$F_2(x) = \begin{cases} e^{-bx} - 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

The dynamic equation of $S_{ikp}$ is

$$\frac{dS_{ikp}}{dt} = A[-Y_{ikp}]$$

$$+ H_1 b \left( Y_{ikp} - Y_{(k+1)q} \right) \left( \exp(-b(S_{ikp} - Y_{ikp} Y_{(k+1)q} - S_{i(k+1)q} Y_{ikp} + t_{ikp} Y_{ikp} Y_{(k+1)q})) - 1 \right)$$

$$+ H_2 b Y_{ip} \left( \exp(-b(-S_{iip} Y_{iip})) - 1 \right)$$

$$- H_3 b \left( Y_{ikp} - Y_{iq} \right) \left( \exp(-b(S_{ikp} Y_{ikp} Y_{iq} - S_{ikp} Y_{ikp} Y_{iq} + t_{ikp} Y_{ikp} Y_{iq}))) \right)$$

$$+ H_4 b \left( Y_{ikp} Y_{iq} \right) \left( \exp(b(S_{ikp} Y_{ikp} Y_{iq} - S_{ikp} Y_{ikp} Y_{iq} + t_{ikp} Y_{ikp} Y_{iq}) - 1) \right)$$

(7.62)

where $[-Y_{ikp}]$ is only for the last operation of each job, and the third term is for the first operation of each job.

The dynamic equation of $Y_{ikp}$ is

$$\frac{dY_{ikp}}{dt} = A[-S_{ikp}]$$

$$+ H_1 \left[ b \left( S_{ikp} Y_{(k+1)q} - S_{i(k+1)q} Y_{ikp} + t_{ikp} Y_{ikp} Y_{(k+1)q} \right) \right] \left( \exp(-b(S_{ikp} Y_{ikp} Y_{(k+1)q} - S_{i(k+1)q} Y_{ikp} + t_{ikp} Y_{ikp} Y_{(k+1)q}))) - 1 \right)$$

$$+ H_2 b \left( -S_{iip} Y_{iip} \right) \left( \exp(-b(-S_{iip} Y_{iip})) - 1 \right)$$
\[-H_j b \left( S_{ikp} Y_{jliq} - S_{jliq} Y_{jliq} + t_{ikp} Y_{jliq} \right) \left( \exp\left( -b\left( S_{ikp} Y_{ikp} Y_{jliq} - S_{jliq} Y_{jliq} + t_{ikp} Y_{jliq} \right) \right) \right)\]

\[-H_j b \left( S_{jliq} Y_{jliq} - S_{ikp} Y_{jliq} + t_{jliq} Y_{jliq} \right) \left( \exp\left( -b\left( S_{jliq} Y_{ikp} Y_{jliq} - S_{ikp} Y_{jliq} + t_{jliq} Y_{jliq} \right) \right) \right)\]

\[+H_d b \left( \exp\left( -b \left[ \sum_{p=1}^{M} Y_{ikp} - 1 \right] \right) - 1 \right)\]

\[+H_d b \left( \exp\left( -b \left[ \sum_{p=1}^{M} Y_{ikp} + 1 \right] \right) + 1 \right)\]  

(7.63)

where \([-S_{ikp}]\) is only for the last operation of each job, and the third term is for the first operation of each job.

### 7.3.3 IPPS model II

The model I in the previous section is an IPPS which is considered only the alternative resources for each operation. However, the operation sequences of each job can be set into two or more different ways. The IPPS, then, is dealing with both alternative process plans and resources. The model for this IPPS can be formulated as below.

Suppose that:

- \(i, j\) the index for the job; \(i, j = 1, \ldots, N\), where \(N\) is the number of jobs
- \(k, l\) the index for the operation; \(k, l = 1, \ldots, K\), where \(K\) is the number of operations
- \(p, q\) the index for the resource; \(p, q = 1, \ldots, M\), where \(M\) is the number of resources
- \(r, s\) the index for the alternative process plan; \(r, s = 1, \ldots, R\), where \(R\) is the number of alternative process plans
- \(t_{ikpr}\) the processing time for operation \(k\) of job \(i\) on resource \(p\)
- \(S_{ikpr}\) the starting time of operation \(k\) of job \(i\) on resource \(p\)
\[ S_{ik,pr} \] the starting time of the last operation of job \( i \) on resource \( p \)

\[ Y_{ikpr} = \begin{cases} 1, & \text{if resource } p \text{ is selected for operation } k \text{ of} \\ \text{in the } r\text{th alternative process plan of the job } i \\ 0, & \text{otherwise} \end{cases} \]

\[ Z_{ir} = \begin{cases} 1, & \text{if } r\text{th alternative process plan of job } i \text{ is selected} \\ 0, & \text{otherwise} \end{cases} \]

The mixed-integer nonlinear programming model for solving the IPPS model II is:

\[
\text{Minimize} \quad \sum_{i=1}^{N} (S_{ik,pr} Y_{ikpr} Z_{ir} + t_{ikpr} Y_{ikpr} Z_{ir}) \tag{7.64}
\]

subject to

\[
S_{ik,pr} Y_{ikpr} Y_{i(k+1)qr} Z_{ir} - S_{i(k+1)qr} Y_{ikpr} Y_{i(k+1)qr} Z_{ir} + t_{ikpr} Y_{ikpr} Y_{i(k+1)qr} Z_{ir} \leq 0 \tag{7.65}
\]

\[
S_{i1,pr} Y_{i1pr} Z_{ir} \geq 0 \tag{7.66}
\]

\[
\left[ (S_{ikpr} - S_{jlpq} + t_{ikpr}) Y_{ikpr} Y_{jlpq} Z_{ir} Z_{js} \right] \vee \left[ (S_{jlpq} - S_{ikpr} + t_{jlpq}) Y_{ikpr} Y_{jlpq} Z_{ir} Z_{js} \right] \tag{7.67}
\]

\[
\sum_{p=1}^{M} Y_{ikpr} = 1 \tag{7.68}
\]

\[
\sum_{r} Z_{ir} = 1 \tag{7.69}
\]

\[
Y_{ikpr} \in \{0,1\} \tag{7.70}
\]

\[
Z_{ir} \in \{0,1\} \tag{7.71}
\]

Constraint (7.65) is the precedence constraint to ensure that two operations of each job cannot be processed at the same time. Constraint (7.66) implies the starting time of the first operation of each job must be either positive or zero. Constraint (7.67) ensures that a resource cannot process more than one job simultaneously. Constraint (7.68) means that
only one resource can perform in each operation. Constraint (7.69) ensures that only one alternative process plan can be selected for job \( i \). Constraints (7.70) and (7.71) imply integrality of the corresponding variables.

The energy function \( E \) of the IPPS model II is

\[
E = A \sum_{i=1}^{N} (S_{ikpr} Y_{ikpr} Z_{ir} + t_{ikpr} Y_{ikpr} Z_{ir}) + \sum_{i=1}^{N} \sum_{r=1}^{R} H_1 F_1 \left[ (S_{ikpr} - S_{i(k+1)qr} + t_{ikpr}) Y_{ikpr} Y_{i(k+1)qr} Z_{ir} \right] \\
+ \sum_{i=1}^{N} \sum_{r=1}^{R} H_2 F_1 (-S_{i1ip} Y_{i1ip} Z_{ir}) + \sum_{1 \leq k, l \leq K : p, q, s, t, j, s \leq M : 1 \leq r, s \leq R} \left[ H_3 F_2 \left[ (S_{ikpr} - S_{ijlps} + t_{ikpr}) Y_{ikpr} Y_{ijlps} Z_{ir} Z_{js} \right] \right] \\
+ \sum_{r=1}^{R} \sum_{i=1}^{N} \sum_{k=1}^{K} H_4 F_1 \left[ \sum_{p=1}^{M} Y_{ikpr} - 1 \right] + \sum_{i=1}^{N} \sum_{r=1}^{R} \sum_{k=1}^{K} H_4 F_1 \left[ -\sum_{p=1}^{M} Y_{ikpr} + 1 \right] \\
+ \sum_{i=1}^{N} H_4 F_1 \left[ \sum_{r=1}^{R} Z_{ir} - 1 \right] + \sum_{r=1}^{R} H_4 F_1 \left[ -\sum_{r=1}^{R} Z_{ir} + 1 \right]
\]

(7.72)

where \( A, H_1, H_2, H_3, \) and \( H_4 \) are constants.

By using the same penalty functions as described in IPPS model I, the dynamic equation of \( S_{ikpr} \) is

\[
\frac{dS_{ikpr}}{dt} = A * [-Y_{ikpr} Z_{ir}] + H_1 b \left( Y_{ikpr} Y_{i(k+1)qr} Z_{ir} \right) (\exp(-b((S_{ikpr} - S_{i(k+1)qr} + t_{ikpr}) Y_{ikpr} Y_{i(k+1)qr} Z_{ir}))) - 1) \\
+ H_2 b Y_{ikpr} Z_{ip} (\exp(-b(-S_{i1ip} Y_{i1ip} Z_{ip}))) - 1) \\
-H_3 b (Y_{ikpr} Y_{ijlps} Z_{ir} Z_{js}) (\exp(-b(S_{ikpr} - S_{ijlps} + t_{ikpr}) Y_{ikpr} Y_{ijlps} Z_{ir} Z_{js}))
\]
\[ + H_4 b \left( \sum_{p=1}^{M} Y_{ikpr} - 1 \right) - 1 \]

\[ + H_4 b \left( \exp \left( -b \sum_{p=1}^{M} Y_{ikpr} - 1 \right) \right) \]

where \([-S_{ik,pr} Z_{ir}]\) is only for the last operation of each job, and the third term is for the first operation of each job.

The dynamic equation of \(Z_{ir}\) is

\[
\frac{dZ_{ir}}{dt} = A[-(S_{ik,pr} Y_{ik,pr} + t_{ik,pr} Y_{ik,pr})]
\[
+ H_1 \left[ b ((S_{ikpr} - S_{i(l+k)qr} + t_{ikpr}) Y_{ikpr} Y_{i(k+1)qr} Z_{ir}) \right]
\]

\[(7.74)\]

\[
\times (\exp(-b((S_{ikpr} - S_{i(l+k)qr} + t_{ikpr}) Y_{ikpr} Y_{i(k+1)qr} Z_{ir}))) - 1) \]

\[
+ H_2 b (-S_{ik,pr} Z_{ir}) \ast (\exp(-b(-S_{ik,pr} Y_{i(l+1)pr} Z_{ir}))) - 1) \]

\[
- H_3 b \left( ((S_{ikpr} - S_{i(l+1)pr} + t_{ikpr}) Y_{ikpr} Y_{i(k+1)pr} Z_{ir})\right) \]

\[
(\exp(-b((S_{ikpr} - S_{i(l+k)+1pr} + t_{ikpr}) Y_{ikpr} Y_{i(k+1)pr} Z_{ir}))) - 1) \]

\[ + H_4 b \left( \exp \left( -b \sum_{p=1}^{M} Y_{ikpr} + 1 \right) \right) + 1 \]
\[ \begin{align*}
+ H_2 b \left( -S_{rt, pr} Y_{rt, pr} \right) \ast (\exp(-b(-S_{rt, pr} Y_{rt, pr} Z_{ir} Z_{jr} Z_{sj})) - 1) \\
-H_3 b \left( (S_{ik, pr} - S_{jl, pr} + t_{ik, pr}) Y_{ik, pr} Y_{jl, pr} Y_{i, j, s} Y_{i, j, s} \right) (\exp(-b((S_{ik, pr} - S_{jl, pr} + t_{ik, pr}) Y_{ik, pr} Y_{jl, pr} Y_{i, j, s} Z_{i, j, s} Z_{i, j, s}))) \\
-H_4 b \left( (S_{jl, pr} - S_{ik, pr} + t_{jl, pr}) Y_{ik, pr} Y_{jl, pr} Y_{i, j, s} Y_{i, j, s} \right) (\exp(-b((S_{jl, pr} - S_{ik, pr} + t_{jl, pr}) Y_{ik, pr} Y_{jl, pr} Y_{i, j, s} Z_{i, j, s} Z_{i, j, s}))) \\
+ H_2 b \left( \exp\left(-b \left[ \sum_{r=1}^{R} Z_{ir} - 1 \right] \right) - 1 \right) \\
+ H_3 b \left( \exp\left(-b \left[ -\sum_{r=1}^{R} Z_{ir} + 1 \right] \right) + 1 \right)
\end{align*} \] (7.75)

where \([-S_{ik, pr} Y_{ik, pr} + t_{ik, pr} Y_{ik, pr}]\) is only for the last operation of each job, and the third term is for the first operation of each job.

### 7.4 Simulation Results

#### 7.4.1 Problem 1: Scheduling with alternative resources.

The proposed HNN was tested with the work of Lee and Dicesare (1992). Their work consisted of two jobs and three resources. The job related-information is given in Table 7.1. The first job has two operations. Its first operation can be performed by resource no. 1 or no. 2. Its second operation can be implemented by resource no. 2, or no 3.

For the second job, there are also two operations. The first one can be operated by resource no. 1 or no. 3, while the second operation of job no. 2 must be run by resource no. 1, no. 2, or no. 3.

Therefore, there are 18 neurons in this simulation. Nine neurons are for starting time variables, which are \(S_{111}, S_{112}, S_{122}, S_{123}, S_{211}, S_{213}, S_{221}, S_{222},\) and \(S_{223}\). Other nine neurons are for resource selection variables, which are \(Y_{111}, Y_{112}, Y_{122}, Y_{123}, Y_{211}, Y_{213},\)
There are ten precedence constraints, six resource constraints, and eight machine selection constraints. There are two constraints for the nonnegative of the first operation starting time. However, these constraints can be neglected by using a positive linear function or a saturating linear function.

The IPPS problem is a kind of the constrained optimization problem. Hence, we use the concept of constrained optimization described in Chapter 5 for solving the IPPS problem. The simulation is performed by MATLAB/Simulink. The parameter settings are $A = 10^{-10}, H_1 = 2, H_3 = 4, H_4 = 1,$ and $C = 10^7$. The initial values of all variables are 1. The integration method is Euler method with $10^{-4}$ step size.

The trajectories of all starting time variables are shown in Figure 7.2, 7.3, and 7.4 for HNN, BFGS HNN, and Nazareth HNN, respectively. The convergence rate of Nazareth HNN is 2 seconds while the BFGS HNN is 3 seconds. The convergence rate of HNN is around 10 seconds. The Nazareth HNN converges in the fastest speed comparing to BFGS HNN and HNN. The convergence rate of BFGS HNN and Nazareth HNN is more than four times of HNN. The values of resource selection variables are depicted in Figure 7.5, 7.6, and 7.7 for HNN, BFGS HNN, and Nazareth HNN, respectively. The results of starting time variables for HNN, BFGS HNN, and Nazareth HNN are shown in Table 7.2. The results of resource selection variables are presented in Table 7.3.

The selected process plans are O112-O123 and O213-O221, for job 1, and for job 2, respectively. The makespan is 11 for all proposed HNN. The results of the modified HNN are the same as the function $h_3(m)$ in the work of Lee and Dicesare (1992). Figure 7.8 depicts the Gantt chart of this simulation.
Table 7.1  Order related-information of problem 1 (Lee & Dicesare, 1992)

<table>
<thead>
<tr>
<th>Job no.</th>
<th>Operation sequences</th>
<th>Operation for alternative resources (operation no, resource no., processing time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>O11 – O12</td>
<td>(O11, 1, 3), (O11, 2, 4), (O12, 2, 3), (O12, 3, 2)</td>
</tr>
<tr>
<td>2</td>
<td>O21 – O22</td>
<td>(O21, 1, 4), (O21, 3, 2), (O22, 1, 3), (O22, 2, 4), (O22, 3, 4)</td>
</tr>
</tbody>
</table>
Figure 7.2  The trajectories of starting time neurons for problem 1 using HNN.
Figure 7.3  The trajectories of starting time neurons for problem 1 using BFGS HNN.
Figure 7.4  The trajectories of starting time neurons for problem 1 using Nazareth HNN.
Figure 7.5 The trajectories of resource selection variables for problems 1 using HNN.
Figure 7.6  The trajectories of resource selection variables for problem 1 using BFGS HNN.
Figure 7.7 The trajectories of resource selection variables for problem 1 using Nazareth HNN.
Table 7.2  The results of starting time variables for problem 1.

<table>
<thead>
<tr>
<th>Method</th>
<th>$S_{111}$</th>
<th>$S_{112}$</th>
<th>$S_{122}$</th>
<th>$S_{123}$</th>
<th>$S_{211}$</th>
<th>$S_{213}$</th>
<th>$S_{221}$</th>
<th>$S_{222}$</th>
<th>$S_{223}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>HNN</td>
<td>0</td>
<td>2.253</td>
<td>4</td>
<td>0.4913</td>
<td>0</td>
<td>2.001</td>
<td>2.003</td>
<td>1.999</td>
<td></td>
</tr>
<tr>
<td>BFGS HNN</td>
<td>0.1034</td>
<td>0</td>
<td>1.987</td>
<td>4</td>
<td>0.6806</td>
<td>0</td>
<td>2</td>
<td>1.99</td>
<td>1.839</td>
</tr>
<tr>
<td>NazarethHNN</td>
<td>0.1709</td>
<td>0</td>
<td>1.878</td>
<td>4</td>
<td>0.7704</td>
<td>0</td>
<td>2</td>
<td>1.981</td>
<td>1.911</td>
</tr>
</tbody>
</table>
Table 7.3  The results of resource selection variables for problem 1.

<table>
<thead>
<tr>
<th>Method</th>
<th>$Y_{111}$</th>
<th>$Y_{112}$</th>
<th>$Y_{122}$</th>
<th>$Y_{123}$</th>
<th>$Y_{211}$</th>
<th>$Y_{212}$</th>
<th>$Y_{221}$</th>
<th>$Y_{222}$</th>
<th>$Y_{223}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>HNN</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BFGS HNN</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Nazareth HNN</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Figure 7.8  Gantt chart of problem 1.
7.4.2 Problem 2: Scheduling with alternative process plans

This simulation consists of two jobs and four resources. The first job has three alternative process plans while the second job has only one alternative process plan. The related-information of the problem is adopted from Moon et al. (2008). Table 7.4 shows the processing time and alternative process plans of the problem.

There are 38 neurons in this simulation: 17 for the starting time variables, 17 for the resource selection variables, and other four variables for alternative process plan selection. There are 17 precedence constraints, nine resource constraints, 11 machine-selection constraints, and four process plan selection constraints.

The parameter settings are \( A = 10^{-10}, H_1 = 2, H_3 = 4, H_4 = 1, \) and \( C = 10^7. \) The initial values of all variables are 1. The integration method is Euler method with \( 10^{-3} \) step size. The convergence rate \( \mu \) of HNN is set to 5 to increase the computation time of the system.

The trajectories of all starting time variables are shown in Figure 7.9, 7.10, and 7.11 for HNN, BFGS HNN, and Nazareth HNN, respectively. The values of resource selection variables are depicted in Figure 7.12, 7.13, and 7.14 for HNN, BFGS HNN, and Nazareth HNN, respectively. Figure 7.15, 7.16, and 7.17 illustrate the trajectories of alternative process plan selection variables for HNN, BFGS HNN, and Nazareth HNN, respectively. The convergence rate of Nazareth HNN is 4.5 seconds while the BFGS HNN is 5.5 seconds. The convergence rate of HNN is around 10 seconds. The Nazareth HNN converges in the fastest speed comparing to BFGS HNN and HNN. The convergence rate of BFGS HNN and Nazareth HNN is more than seven times of HNN. The results of starting
time variables for HNN, BFGS HNN, and Nazareth HNN are shown in Table 7.5. The results of resource selection variables and process plan selection are presented in Table 7.6. Figure 7.18, 7.19, and 7.20 illustrate the Gantt charts of the problem by using HNN, BFGS HNN, and Nazareth HNN, respectively.

From Table 7.6, the result of Nazareth HNN is different from HNN and BFGS HNN. The $Y_{1123} = 1$ and $Y_{1113} = 0$ for HNN, and BFGS HNN while $Y_{1123} = 0$ and $Y_{1113} = 1$ for Nazareth HNN. Additionally, $Y_{1233} = 1$ and $Y_{1243} = 0$ for HNN, and BFGS HNN but $Y_{1233} = 0$ and $Y_{1243} = 1$ for Nazareth HNN. However, the Gantt chart of HNN, BFGS HNN, and Nazareth HNN are similar. The algorithms choose the process plan #2 for the job 1, that is $r = 2$. Consequently, the algorithms determine only the variables that involve the process plan #2 and neglect the others.
Table 7.4  Job related-information and alternative operation sequences of example 2

<table>
<thead>
<tr>
<th>Job no.</th>
<th>Alternative operation sequences</th>
<th>Operation for alternative resources (operation no, resource no., processing time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>O11 – O12 – O13</td>
<td>(O11, 2, 4), (O11, 1, 5), (O12, 3, 2), (O12, 4, 3), (O13, 2, 5)</td>
</tr>
<tr>
<td></td>
<td>O11 – O13 – O12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>O13 – O11 – O12</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>O21 – O22</td>
<td>(O21, 3, 4), (O22, 4, 5)</td>
</tr>
</tbody>
</table>
Figure 7.9 The trajectories of starting time variables of problem 2 using HNN
Figure 7.10  The trajectories of starting time variables of problem 2 using BFGS HNN
Figure 7.11  The trajectories of starting time variables for problem 2 using Nazareth HNN
Figure 7.12  The trajectories of resource selection variables of problem 2 using HNN
Figure 7.13  The trajectories of resource selection variables of problem 2 using BFGS HNN
Figure 7.14 The trajectories of resource selection variables of problem 2 using Nazareth HNN
Figure 7.15  The trajectories of alternative process plan selection variables of problem 2 using HNN.
Figure 7.16 The trajectories of alternative process plan selection variables of problem 2 using BFGS HNN.
Figure 7.17  The trajectories of alternative process plan selection variables of problem 2 using Nazareth HNN.
Table 7.5  The results of starting time variables for problem 2

<table>
<thead>
<tr>
<th>Method</th>
<th>$S_{1121}$</th>
<th>$S_{1111}$</th>
<th>$S_{1122}$</th>
<th>$S_{1112}$</th>
<th>$S_{1123}$</th>
<th>$S_{1113}$</th>
<th>$S_{1231}$</th>
<th>$S_{1241}$</th>
<th>$S_{1232}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>HNN</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.826</td>
<td>1.005</td>
<td>4</td>
<td>3.44</td>
<td>9.008</td>
</tr>
<tr>
<td>BFGS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.4594</td>
<td>0.09759</td>
<td>4.002</td>
<td>3.097</td>
<td>9.686</td>
</tr>
<tr>
<td>Nazareth</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.633</td>
<td>1.327</td>
<td>4.084</td>
<td>4.467</td>
<td>9.441</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>$S_{1242}$</th>
<th>$S_{1233}$</th>
<th>$S_{1243}$</th>
<th>$S_{1321}$</th>
<th>$S_{1322}$</th>
<th>$S_{1323}$</th>
<th>$S_{2131}$</th>
<th>$S_{2241}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>HNN</td>
<td>4.93</td>
<td>5.812</td>
<td>4.603</td>
<td>6</td>
<td>4.004</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>BFGS</td>
<td>4.708</td>
<td>4.846</td>
<td>4.027</td>
<td>6.022</td>
<td>4.226</td>
<td>0</td>
<td>0</td>
<td>4.005</td>
</tr>
<tr>
<td>Nazareth</td>
<td>5.457</td>
<td>5.415</td>
<td>5.891</td>
<td>6.863</td>
<td>4.208</td>
<td>0</td>
<td>0</td>
<td>4.006</td>
</tr>
</tbody>
</table>
Table 7.6  The results of resource selection and process plan selection variables for problem 2

<table>
<thead>
<tr>
<th>Method</th>
<th>$Y_{121}$</th>
<th>$Y_{111}$</th>
<th>$Y_{112}$</th>
<th>$Y_{113}$</th>
<th>$Y_{123}$</th>
<th>$Y_{114}$</th>
<th>$Y_{124}$</th>
<th>$Y_{123}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>HNN</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>BFGS HNN</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Nazareth HNN</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>$Y_{124}$</th>
<th>$Y_{131}$</th>
<th>$Y_{132}$</th>
<th>$Y_{133}$</th>
<th>$Y_{213}$</th>
<th>$Y_{224}$</th>
<th>$Z_{11}$</th>
<th>$Z_{12}$</th>
<th>$Z_{13}$</th>
<th>$Z_{21}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>HNN</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>BFGS HNN</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Nazareth HNN</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Figure 7.18 The Gantt chart of the problem 2 using HNN.
Figure 7.19  The Gantt chart of the problem 2 using BFGS HNN.
Figure 7.20  The Gantt chart of the problem 2 using Nazareth HNN.
The results from BFGS HNN and Nazareth HNN are near-optimal solutions. Their Gantt charts depict idle times for machine 2 and 3. Therefore, we can apply the global optimization concept described in Chapter 6 to find the optimal solutions or reduce the idle times in the schedules. Figure 7.21 and Figure 7.22 show the trajectories of all starting time variables of the global BFGS HNN and the global Nazareth HNN, respectively. The corresponding Gantt chart for the global BFGS HNN is illustrated in Figure 7.23. Its makespan is 20. The Gantt chart of the global Nazareth HNN is depicted in Figure 7.24. Its makespan is 20.01.
Figure 7.21 The trajectories of starting time variables of problem 2 using BFGS HNN and the global optimization.
Figure 7.22 The trajectories of starting time variables of problem 2 using Nazareth NNN and the global optimization.
Figure 7.23  The Gantt Chart of the global BFGS HNN for Example 2.
The result of Nazareth-HNN for EX2

Figure 7.24  The Gantt Chart of the global Nazareth HNN for Example 2.
7.5 Summary

An integrated process planning and scheduling model has been proposed for Hopfield neural network implementation. For solving the problem, mix-integer programming methods and penalty function are used. The integer variables are relaxed to continuous variables by using sigmoid transfer function. Therefore, the gradient of energy function is possible to find with the values of continuous variables. The energy function and dynamic equation of IPPS are proposed. In the simulation, the modified HNNs converge to a feasible region faster than the conventional HNN.
CHAPTER 8
CONCLUSIONS

8.1 Summary

The convergence rate of Hopfield neural network is increased when changing its
descent direction to BFGS direction or conjugate gradient direction. By defining these
directions as dynamical systems, they can be integrated with other methods that are based
on the gradient direction. The modified Hopfield neural networks perform well with the
integrated process planning and scheduling. A summary of this dissertation is given as
follow.

1) The BFGS method is one of the most effective optimization approaches. It
approximates the Hessian or inverse Hessian of the objective function. In our
proposed methods, the BFGS updating is defined as a dynamic equation.
Integrators are used to control the initial condition and process termination. They
also do the BFGS computation for the Hopfield neural network. The convergence
rate of the modified BFGS Hopfield neural network is increased around six times
than the traditional Hopfield neural network.

2) The Nazareth direction which is an improvement of conjugate gradient
direction can increase the convergence rate of the traditional Hopfield neural
network. The gradient direction of Hopfield neural is added with two dynamic
systems. One is the dynamic system of conjugate gradient parameter. Another one
is the dynamic system of BFGS updating. The modified Hopfield network also
shows a good result in convergence rate increasing. The convergence rate of Nazareth Hopfield neural network is faster than the BFGS Hopfield neural network.

3) The modified Hopfield neural network can also be implemented for constrained optimization problems. The constraints are included in the objective function by using a penalty function. Electronic devices can construct the penalty functions. The proposed penalty function method can use small values of penalty parameters. The simulation shows that BFGS Hopfield neural network and Nazareth Hopfield neural network converge to the optimal solution faster than the traditional Hopfield neural network.

4) Normally, the solution of Hopfield neural network is a local minimum. Integrating the Gaussian noise with the Hopfield neural network forces the solution to escape from a local minimum. Our modified Hopfield neural networks combine Gaussian noise and sign changing of convergence rate to make an efficient method of finding the global minimum.

5) The integrated process planning and scheduling model is in the form of mixed integer nonlinear programming. The model is defined in term of energy function and its dynamic system for each variable. The modified Hopfield neural networks are implemented to the integrated process planning and scheduling model. BFGS and Nazareth Hopfield neural network demonstrate good results in term of convergence speed and solution quality.
8.2 Recommendations for Future Research

In the future, some directions could be further developed both in methodology and simulations.

1) The mixed integer nonlinear programming model of IPPS can be transformed to a linear programming by some techniques. The linear transformation will reduce the complexity of the model and computation.

2) Stability and convergence analysis of the modified HNN could be investigated.

3) Multi-objective IPPS could also be implemented. The model would be more realistic in terms of industrial aspects.

4) Hardware simulation of BFGS updating and Nazareth algorithm could be investigated.
REFERENCES


201