On FPGA Implementations for Bioinformatics, Neural Prosthetics and Reinforcement Learning Problems

MAK Sui Tung Terrence

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Systems Engineering and Engineering Management

June 2005

Abstract

Along with the remarkable success of Human-Genome-Project and the launch of Genome-to-Life project, voluminous biological data was accumulated and the secrets of life were yet to be deciphered. Innovations and implementations of high performance computing infrastructure to support the computational analysis of large-scale data set, complex model simulations and achieving effective access to biological data are considered to be equally important as continuing to generate new biological data from pipelining biotechnological machineries.

In parallel with the development of the computational bioinformatics, neural prosthetics, on the other hand, emerges as an innovative and interdisciplinary subject, in which the need of real-time performance is critical. Neuron-machine interfaces such as dynamic clamp and brain-implantable neuro-prosthetic devices require real-time simulations of neuronal ion channel dynamics, in which silicon devices mimic the living cell mechanics. A hardware-based, application-specific implementation of the neuronal ion channel dynamics would circumvent the limitations of computational speed and flexibility in general-purpose computers.

Further, for future combat to reduce human causalities, gathering scientific data from distant planets or in extreme environments, intelligent robots such as Unmanned Aerial Vehicles (UAVs), and Mars Rover, should be equipped with capability learning and real-time decision making, which are both computational intensive tasks. Delay in the decision making, due to heavy computational burden, would be disastrous. The distributed and parallel computational learning architecture could circumvent the limitation of the sequential computation machineries.

Field programmable gate array (FPGA) has emerged as a high-speed digital platform for application-specific computation devices. However, it is a challenge to have efficient design with good mapping from algorithms to hardware logics, due to the design complexity and the algorithm complexity. The objective of this thesis is to investigate and develop efficient methods to address the computational and implementation challenges in FPGA The research covers area from computational genomics, neural prosthetics applications to theoretical reinforcement

learning implementation in a distributed environment, with a goal to improve existing software-based implementation methodologies and sequential computational framework.

In light of the advance programmable distributed logics technology, we have worked on the FPGAs designs based on three different application-oriented examples. Firstly, we propose a novel approach for searching equivalence set from genetic network and offer an efficient FPGAs implementation. The approach is beneficial to genetic network dynamics development. Secondly, we propose a FPGA-based architecture for computational intensive phylogenetic tree topology evaluation. Significant acceleration for the tree evaluation can be obtained based on the proposed hardware architecture. Thirdly, we present an FPGA design of a neuromorphic Hebbian synapse which mimics the NMDA and non-NMDA ion channel dynamics observed experimentally in hippocampal neurons. The proposed design can be readily extended to high-speed implementations of dynamic clamp and neuro-prosthetics as replacements for damaged neurons in the brain.

In addition, we consider a distributed computational framework for continuous-time inference network for solving dynamic programming problems. It is a computational intensive task which becomes the bottleneck of real-time reinforcement learning and decision making. We offer an FPGAs implementation scheme with distributed arithmetic with adaptation of the embedded multipliers and logics. We also consider an analog Very Large Scale Integrated (VLSI) design for the realization and verification of the theoretical formulation. We show that the inference network converges to the optimal Bellman optimality condition, for which the convergence rate can be made arbitrarily fast and is practically independent of the problem size. Lastly, based on the framework of the Bellman Inference Network, a novel Q-learning network architecture is proposed. We introduce the exploration and memory components to the original Bellman inference network to form a connectionist network with the capability of parallel learning and optimization. We found that the Q-learning network significantly outperforms the conventional Q-learning algorithm under a distributed unknown environment. We also proposed two design alternatives for the realization of Q-learning network using FPGAs with the network dedicated to fit different applications.

摘要

隨著「人類基因組計劃」革命性的成功和近期展開的大規模微生物基因排序「從基因研究了解生命」，國際基因庫屯積了大量生物數據，但是我們對生命密碼的破解還是處於起步的階段。高效能計算分析、複雜模型運算，及有效存取生物數據的創新性實現方法研究應和繼續採集生物數據有相同的重要性。在生物計算高速發展的同時，腦─機界面(或稱神經義肢)成爲對實時計算有高要求的新興學科。當中，以動態鉗和腦部可植入晶片的研究對高效能實時運算有最高要求。通過在專用硬件上實現，此研究可用作模擬真細胞的運作和模擬腦細胞離子通道運作並可改善普通電腦在運算速度和彈性上的限制。另外，爲應付未來戰爭或到外星極地進行科學研究，智能機械人，例如無人駕駛飛機，火星挑戰者號，應具備實時學習及決策的能力。但這些都是需要大量運算的。如果因要應付大量運算而延誤了決策，可能導致嚴重的後果。

近年，現場可編程門陣列已成爲現代專門高速運算硬件平台。但因爲硬件設計和算法的複雜性，令算法在硬件實現上出現很大的困難。本論文的目的是研究在生物信息、腦─機界面、及機器學習的問題上如何有效地實現分佈式硬體以達到提高單以軟體爲基礎的實現方法。我們針對三個不同的領域的問題進行現場可編程問陣列硬件實現的研究。第一，我們提出遺傳算法及動態規劃的綜合算法以解決同類基因搜索的問題。此新算法對建造基因動態模形有很大幫助。此外，我們還對算法提供了硬體的實現方法，以大大縮短所需計算時間。第二，我們提出了以並行硬件實現模式以加快基因遺傳樹重建的密集計算。第三，我們提出一套對腦細胞離子通道模擬的硬件實現方法。硬件設計對高速互動鉗和腦─機介面的建造有極大的幫助。

除此以外，本論文也針對增強式學習問題提出了間斷性和連續性推理網絡的理論基礎及實現方法。該方法可以突破實時學習的計算速度瓶頸。我們提供了現場可編程門陣列及模擬電路在超大規模集成電路的實現。最後爲解決在信息不全的情況下增強學習的問題，我們提出了 *Q*-學習網絡的構想。從初步的模擬測試中，*Q*-學習網絡表現了較有效率的學習及優化計算。此外，我們也提出了不同的硬件實現方法以應付不同的需求。

List of Tables

List of Figures

Acknowledgements

First of all, I would like to express my greatest gratitude to my supervisor, Professor Kai-Pui Lam, for his invaluable suggestions and comments on this thesis. More importantly, he had showed me the philosophy of engineering and the basic elements that a researcher should be equipped with during the last two years. Also, I have learnt a lot through our discussions on different research aspects. Specifically, I would also like to thank for his kind support and arrangement for my visiting study at MIT.

I would also like to express my gratefulness to Dr. Chi-Sang Poon, my supervisor when I was a visiting student at MIT. I was greatly impressed by his passionate and meticulous research attitude and his confident and cheerful personality. I have learnt a lot about research through our discussion. Also, I would like to thank Shirley, Guy, Yunguo, Chung, Armon, Dr. Song, Shawnon, Mary, Andy, Vera, Louis, Ricky and friends at Boston for their kind support and accompany at the time when I was in Boston. I would like to specially thank Guy Rathmuth for teaching me analog VLSI design, layout and fabrication technical techniques.

I greatly appreciate the members of my dissertation committee, Professor Peter Cheung, Professor Kam-Fai Wong and Professor Wai Lam for their useful feedback and suggestions.

I am also greatly indebted to Professor Shuzhong Zhang and Professor Duan Li on behalf of Department of Systems Engineering and Engineering Management to support my visiting study at MIT during my M.Phil study. I would also like to thank the graduate school of The Chinese University of Hong Kong to support the MIT visit.

My greatest gratitude goes to my parents for their endless love and encouragement. Most importantly, I would express my love to my girlfriend, Miss Ophelia Tsui for her love and support.

Table of Contents

Chapter 1

Introduction

1.1 Bioinformatics

February 2001, it was the landmark of human technological advance, the remarkable success of Human Genome Project (Baltimore 2001; Consortium 2001; Venter et al. 2001), by raveling the underpinning of numerous genomes and DNA sequences from microbes to plants to mammals, have created a revolution in biology that has no equal in the history of science. Genomics is now the starting point for studies in biology, making tractable for the first time a systematic and deep understanding of life's process. While stunning in the impact of the human genome project, our foray into genomes has touched upon only the tiniest fraction of life on earth. The diversity and range of the environment adaptations of microbes mean that they long ago evolved solutions to many problems that scientists must now address. Comprising about fifty percent of the earth's biomass (Whitman, Coleman et al. 1998), microbes are in consequence the foundation of the biosphere, controlling earth's biogeochemical cycles and affecting the productivity of the soil, quality of water, and global climate. The desire and aspiration of understanding the life in the microbe's would, leads to the launch of another project, Genome to Life, with scale equivalent to the Human Genome Project (Frazier, Thomassen et al. 2003).

The advent of the Human Genome Project and Genome to Life projects have impacted all of the biomedical research and brought the exhilaration and celebration of the entering of post-genomic era. However, the development of large sets of genomic data has challenged biologists who as a rule have never encountered data of this scale. The secrets of life are yet to be

deciphered. Implementations of conventional statistical techniques have brought a level of order and results to the biologist's desktop. But significant problems remain to be solved. The challenge remains to extract the maximum useful information from genomic data. Consider biology is awash by the amount of data and its complexity, which demands increasing computation power. In addition to generation new data from pipelining biotechnological machineries, innovations and implementations of computing infrastructure to support the computational analysis of large-scale data set, complex model simulations and achieving effective access to biological data are equally important.

Starting from gene sequencing, the combination of DNA base pairs of thousands genes can be identified, and clues for a variety of structural and functional features might be provided. However, to our knowledge, only molecular machinery of the cell can interpret the encrypted sequences to determine the complex biochemical mechanism and to define the organism behavior (D'haeseleer, Liang et al. 2000). Analysis of genomic sequences aims to understand mechanisms of molecular machinery. Data mining and modeling approaches try to conceptualize and unravel the functional relationships implicitly indicated in the genetic data set. Clustering algorithms are typically popular on automatically grouping of genes based on their expression measure and prior knowledge of the biological experiments. Some of these techniques, such as Singular Value Decomposition (SVD) for genes clustering (Alter, Brown et al. 2000; Wall, Dyck et al. 2001), expectation-maximization algorithm to cluster yeast data (Mjolsness, Mann et al. 1999), standard agglomerative hierarchical clustering algorithm for average-linkage analysis on gene expression data (Eisen, Spellman et al. 1998), are considered as popular unsupervised learning methods, from the engineering perspective.

Still, clustering approach typically only tells us correlative information, but not causality and regulating models about genes, though it is a relatively easy way to extract useful information out of large-scale gene expression data sets (D'haeseleer, Liang et al. 2000). More advance analysis aims to infer causal connections between genes and network dynamics. One way to make progress in understanding the principles of network dynamic is to radically simplify the individual molecular interactions, and focus in the collective outcome (D'haeseleer, Liang et al. 2000; Maki, Tominaga et al. 2001). Boolean network (Kauffman 1969) represents such a simplification: each gene is considered as a binary variable – either ON or OFF – regulated by other genes through logical or Boolean functions. Although there are effective measurements of

genetic data from the microarray experiments, computational analysis is complex and is a serious bottleneck in the study of genetic network using contemporary computing technology. New computational paradigm of massive parallel and high performance computers are therefore in great demand.

Among the computational genomic examples, evolutionary studied of different organisms, through the judicious evaluation of DNA or protein sequences, is still the main stream in biology. This is because an organism is best understood in the light of its evolutionary relationship to other organisms (Karp 2003). Comparative study based on the theory of evolution can be traced back to Charles Darwin (Darwin 1929). Instead of comparing the phenotype, such as size of eye ball, or palm and height, nowadays, metrics based on DNA sequences are applied for evolutionary studies. Computational model and methods to infer (or reconstruct) the evolutionary history of organism, namely phylogentic tree reconstruction, becomes everyday practices of biologists in many laboratories.

Phylogenetic tree study is one of the most fundamental bases in biology. It has been shown that phylogeny analysis has its profound implication in pharmaceutical research which focus on the evolution of virus for the purposes of drug-resistance prediction, immune-escape of mutations (Rambaut, Posasa et al. 2004), etc. Also the strong drug adaptation of HIV virus can be explained by its fast mutation rate. In (Worobey, Santiago et al. 2004), the origin of HIV is using meticulous phylogeny analysis. To decipher HIV interaction with the immune system and to develop effective control strategies, close relatives of the virus are studied using phylogeny (Rambaut, Posasa et al. 2004). Besides virology and pharmaceutical applications of phylogeny, recently, it has been shown that evolutionary studies of species can benefit the genomic studied and the regulatory genetic network prediction as well (Eisen and Fraser 2003).

Phylogenetic tree reconstruction is computational intensive. Especially for recent advance probabilistic-based model, phylogeny computation becomes a challenging engineering issue and considered intractable for desktop computers. It is a difficult task to find the optimal solution based on the maximum likelihood criterion, simply because of the exponentially growth of the possible tree topologies with the number of taxa[1]. The possible unrooted, bifurcating n-taxa tree topologies is corresponding to nearly 16 billion different trees for 12 taxa and 3×10^{84} trees for 55 taxa. The optimal phylogenetic tree search problem is regarded as NP-hard (Lemmon and

[1] Taxa is a general term referring to any kind of taxonomic unit including DNA sequences and nucleotide site.

Milinkovitch 2002) which implies that no known algorithm can find the optimal solution in polynomial time. Heuristics are often used to search the near optimal tree within a reasonable time, which essentially applies hill climbing or genetic algorithm approaches as the search strategies (Strimmer and Haeseler 1996; Swofford, Olsen et al. 1996; Lewis 1998; Lemmon and Milinkovitch 2002). The heuristics can reduce the search space for a near-optimal solution. The tree evaluation is computationally demanding and is used repeatedly in the search. In general, the computational cost of likelihood, accounts for the greatest portion of the execution time (i.e. 95% in sequential execution) (Stamatakis, Ludwig et al. 2002). The computational intensive fixed-topology tree evaluation is time consuming and this would result in the reduction of the overall search speed.

1.2 Neural Prosthetics

In parallel with the success of computational genomics, neural science, is another important subject in biology and medical science. Tremendous efforts have been put on innovation and development in this area in the past two decades (Kandel, Schwartz et al. 2000). One of the frontiers is repair of the human brain: developing prosthetics for the central nervous systems to replace higher thought processes that have been lost due to damage or disease. The type of neural prosthetic that performs or assists a cognitive function is qualitatively different from the cochlear implant or artificial retina, in which transducer converts physical energy from the environment into electrical stimulation or nerve fibers (Loeb 1990), and qualitatively different from functional electrical stimulation (FES), in which preprogrammed electrical stimulation protocols are used to activate muscular movement (Mauritz and Peckham 1987). Although there is still a long way before the ultimate goal is achieved, neural prosthetic silicon neurons would have functional properties specific to those of the damaged neurons, and would both receive as inputs and send as output electrical activities to regions of the brain with which the damaged region previously communicated (Berger et. al.2001).

Real-time simulation is an important step in the implementation of brain-machine interaction (BMI), and is fundamental to several emerging neuromorphic, biomimetic and prosthetics applications. For example, in electrophysiological studies of neuronal membrane properties using the dynamic clamp technique (Sharp et. al. 1993; Butera et. al. 2004) , a digital computer is used

to generate virtual ion channel conductance which continuously interacts with a biological neuron in real time. Such software-based experimental applications are highly computation-intensive and often require judicious choice of operating system (Sharp et. al. 1993) and numerical procedures (Butera et. al. 2004) to improve the computational speed and flexibility. A hardware-based, application-specific implementation of the dynamic clamp technique would circumvent the limitations of general-purpose computers.

Another example of neural prosthetics brain-machine-interface (BMI) is real-time neuronal ion channel dynamics computation. For example, a robotic arm controlled by central brain activities have been shown to be capable of generating complex motions (Taylor 2002), and such capability may find important applications in patients with Parkinson's disease, Essential Tremor, and dystonia (Isaacs et. al. 2000). One such technology is neuromorphic analog VLSI circuits (Mead 1990). Towards this end, Rachmuth and Poon have previously proposed neuromorphic Hebbian synapse design using analog CMOS circuits operating in subthreshold regime (Rachmuth and Poon 2003; Rachmuth and Poon 2004). However, the relatively long design and fabrication cycle for analog CMOS circuits is a bottleneck in the development of such devices.

1.3 Learning in Uncertainty

Besides the real-time requirement of the brain-machine-interface application, rapid learning under stochastic and uncertainty is also important. This is widely used in knowledge acquisition for making time-critical decision in real-time. Autonomous robots and vehicles are assigned to perform missions in highly hazardous and extreme environments. In most of the time, good path planning and quick reaction to avoid dangerous spots can increase the chance to reach the target or accomplish a mission, as in the cases of gathering scientific information from a distant planets or searching and rescuing life from a mass casualty incident site (Team 1997; Volpe, Estlin et al. 2000; Casper and Murphy 2003). One of the examples is the design of Mars rover, which is used for exploration of Mars. Robust navigation through rocky terrains by small mobile robots is challenging as little information about the uncertain environment at extreme conditions and there is only limited number of communication with earth controller, i.e. twice a day (Volpe, Estlin et al. 2000). Highly autonomous robots with intelligent and capability of learning and making decision for path planning, execution are acquired for maximizing scientific data return from the

Mars exploration mission. Sojourner, the Mars Pathfinder rover was sent to Mars making observations on the rocks and other deposits at the Ares site to collect information and prior knowledge to increase the chance for latter rover success (Rover Team 1997).

Since the environment is highly dynamic and unpredictable, path planning relying only on the prior knowledge is risky. Several real-time and on-line planning navigation techniques are proposed to enhance the intelligence and on-line decision making capability of the rover for the coming projects (Williams, Kim et al. 2001). On the other hand, exploration of the surface of a distant planet by networks of autonomous cooperating vehicles would be an effective alternative approach. Collective information from a distributed environment would increase the content of the information and increase the chance of survival of the autonomous robots. Given this option, distributed approaches for rapid learning in an uncertain environment and for making real-time decision is important (Williams, Kim et al. 2001).

Among the examples of real-time path planning and decision making is the development of intelligent Unmanned Aerial Vehicles (UAVs) for future combats to reduce human causalities. The major challenge for intelligent UAVs development is path planning in uncertain and even adversarial environments, for which the objective is to complete the given mission, to arrive at the given target within a pre-specified time, while maximizing the safety of the UAVs. The problem can be modeled as a typical stochastic learning and sequential decision problem (Jun and D'Andrea 2003). However, in practice, UAV path planning is difficult because of two main reasons. Firstly, in the adversarial environment, information is always incomplete and is highly uncertain. It is difficult to acquire knowledge to decide on a reasonably well trajectory of the flight. Secondly, the computational load grows quickly as the number of radar sites increases. Delay in decision making, due to heavy computational burden, would be disastrous. The distributed and parallel computational learning architecture could circumvent the limitation of the sequential computation machineries (Tin 2004).

1.4 The Field Programmable Gate Array (FPGAs)

The latest high-end microprocessors utilize 90*nm* complementary metal oxide semiconductor (CMOS) technology with 64-bit data bus, multiple functional units and megabytes of integrated

cache packed on a single die with up to a hundred million transistors operations at clock frequencies over 3 GHz (Intel 2005). Although the computational abilities of these advanced microprocessors are useful to a wide range of civil, business and engineering applications, they cannot always fulfill the needs of real-time signal processing, high-throughput computational genomic systems and large-scale optimization, which require even higher computational power. High power requirements and large heat dissipation are also shortcomings of microprocessors. Essentially, the limitation of microprocessor system is due to the nature that software programs executing follows a sequential operation within a microprocessor (Leong 2001). In contrast, hardware implementation would utilize hardware parallelism and dedicated logic leading to performance improvement over a microprocessor. The dedicated hardware architecture can be designed to be more power and computational efficient.

Field Programmable Gate Array (FPGAs) becomes unique, power efficient, robust and high performance computation engine. Originally FPGAs simply regarded as a fast prototyping tool for microprocessor design in early 80's. Evolution of silicon reconfiguration technology leads to revolutionary replacement of legacy microcontroller. When comparing to traditional microprocessors, the FPGA allow parallel implementation of computational logics and arithmetic. Given a judicious design, throughput of the applications can be greatly enhanced. For example, FPGA-based computing platform called a Programmable Active Memory (PAM) machine (Bertin and Touati 1994; Mencer, Morf et al. 1998) achieved the fastest reported encryption/decryption rate in history. In their work, it has been shown that an implementation of the International Data Encryption Algorithm (IDEA) on a Xilinx Virtex-XCV300 FPGA achieved 10 times faster than software implementation on a Sun Enterprise E4500 machine equipped with twelve 400 MHz processors. In (Trimberger, Pang et al. 2003), implementation of the Data Encryption Standard (DES) on a Xilinx XCV-300 device achieved 12Gbits/sec encryption rate while only 9.3Gbits/sec was reported in the fastest ASIC implementation.

The technological advancement of FPGAs does not limited to civil applications. but aeronautic and military applications are abounded. Mars Exploration Rover mission utilizes Xilinx FPGAs in critical applications for both the lander and rover vehicles. According to NASA's Jet Propulsion Laboratory in Pasadena, the Spirit Mars Exploration Rover (MER) launched June 10, 2003 and the Opportunity MER launched July 7, 2003 employed some of the most advanced radiation tolerant Xilinx Virtex FPGAs once they reached Mars. The Xilinx

devices was used to control the pyrotechnic devices on the lander, and several motor control functions on the rover, including controllers for the wheels, steering, and antenna gimbals (Xilinx 2003). In (Burke, Cozy et al. 2004), it was found that the operation of FPGAs, both Actel and Xilinx parts, is in good condition at very low temperatures e.g. down to -165°C. Investigation and implementation of control logics using FPGAs would greatly enhance its robustness and speedup the development cycle. This would greatly benefit the autonomous robotic and aeronautic machineries design application for distant planet exploration.

The capacity, functionalities and efficiency of FPGAs technologies does not stop evolving in recent years. The earliest architecture XC4000 FPGAs had up to 180 thousands system gates. Programmable floating gate transistor can be programmed becomes non-volatile memory together with programmable switches form a network of programmable logics. Thousands of logic operators could be connected together. Early architecture performance was limited by its capacity. The later Xilinx Virtex series FPGAs had up to 4 millions system gates and 832 Kbytes embedded memory. It was the first reconfigurable architecture with embedded memories, which had greatly enhanced the competitiveness of FPGA to other signal processing and ASIC processors. Later, Virtex-II series architectures were proposed. It was considered that signal processing performance required large volume of arithmetic operator. Introduction of 18-bit embedded multipliers increased the computational speed of the application. Further, it facilitated higher robustness, power efficiency and logic utilization. More recently, Virtex-II Pro and Virtex-4 FPGAs series enabled with IBM PowerPC immersed into the FPGAs fabric while high performance internal bus architecture dedicating for the communication between the distributed logics, embedded memory, multipliers and the embedded microprocessor (Xilinx 2002). The platform offers great flexibility for users to define task partitioning between the FPGA hardware logic and software running on the embedded microprocessor. It also provides a tightly-coupled HW/SW computing environment and supports a high-speed internal bus, which significantly reduces communication overhead. A central bus infrastructure with dedicated sub-buses and interconnected bridges is essential for providing a high throughput communication gateway connecting the microprocessor and FPGA. More specifically in (IBM 2002), the PowerPC core accesses high-speed system resources (such as instructions and data) through the Processor Local Bus (PLB); and On-Chip Peripheral Bus (OPB) provides the connectivity to the FPGA. As

integrated circuit technology continue to improve at a noticeable rate, one could be assured that even more powerful FPGA devices would be available in the future.

Accumulation of biological data, leads scientists and biologists to look for effective alternatives to process the stacking data and to circumvent the computational problem attributed to desktop computers. FPGAs, on the other hand, would be an effective implementation platform for tackling the computational problem by its distributed architecture and effective use of parallelism. It is only the beginning to apply FPGA technology on biomedical, bioinformatics and even neural prosthetics. Pioneer TimeLogic Inc. commercially provides solutions to the problem of searching genetic databases (TimeLogic 2002), in which 160 FPGAs are used to form a powerful cluster of computational system. Although the system is only one fifth of the computational speed of the machines used in the Human Genome Project, it is more power efficient and inexpensive. (Guccione and Keller 2002) found that, over ten times acceleration achieved for using more advance FPGA processors. In that case 4000 processing units were used to match gene sequences in parallel. This showed that FPGAs was an excellent solution to the bioinformatics applications. Few of the latest achievements in DNA sequence alignment were presented in (Yu, Kwong et al. 2003; Dydek and Bala 2004), where different systolic array parallelization schemes were adopted to gain higher acceleration.

In addition, FPGA implementation is an effective solution to neural prosthetics and reinforcement learning problems mentioned above. Based on the programmable distributed logic cells and other embedded resources, algorithms can be mapped to FPGA. The reconfigurable nature of FPGAs enables multiple designs to be programmed on the same hardware device at different times. Thus the costly design and fabrication process delay associated with Very Large Scale Integration (VLSI) design are avoided. The continuous improvement in silicon technology offers faster and larger FPGA devices over time. Design based on FPGA is versatile. With improved density and shorter design time in the future, designers may implement more sophisticated algorithms leading to further improvement in system performance.

Unfortunately, the FPGAs design and mapping of algorithms to FPGAs are always not straightforward. It is challenging to have efficient design with good mapping from the algorithms to the hardware logics. This is always due to the design complexity and the algorithm complexity. Efficient and simple algorithms are not necessarily efficient to be executed in hardware with trivial implementation. Besides, one difficulty in designing FPGA applications is that hardware

resources are strictly limited. Hardware area efficiency is still a major issue, even though density of FPGA devices has improved substantially. Very often, performance is sacrificed to fit a design into a given FPGA device. It is important to able to explore the tradeoff between area and performance, and a single design description can lead to multiple implementations with area and performance tradeoff.

1.5 Scope of the Thesis

The objective of this thesis is to investigate and develop efficient methods to address the computational challenges, from bioinformatics, neural prosthetics problems to theoretical reinforcement learning implementation in a distributed environment. Our goal are to explore new applications and to improve the performance based on software-based implementation methodologies and sequential computational framework.

Reverse engineering of genetic network is one of the greatest challenges in today's bioinformatics. In (Akutsu, Kuhara et al. 1998; Maki, Tominaga et al. 2001; Kimura, Hatakeyama et al. 2003), the idea of partitioning genes into equivalence sets, from a large genetic network and with reference to a Boolean matrix obtained from the gene expression patterns resulted from microarray experiments was proposed. The equivalence sets facilitate effective grouping of "closed-loop" genes, and can be derived from transitive-closure computation. Subsequently, dynamic models are estimated from the identified equivalence sets for capturing network behavior. Therefore, computation of the transitive-closure for the equivalence sets is as an important step for building static and dynamic models of genetic network (Maki, Tominaga et al. 2001). In Chapter 2, we present a hybrid method, which integrates genetic algorithm (GA) with dynamic programming approach address the computational intensive equivalence sets search problem. This approach converts a high dimension computation problem into a search problem which is solved by GA using dynamic programming. The computation of transitive-closure forms the basic fitness evaluation in GA. This is used for selecting candidate chromosomes generated by applying basic genetic operators. Small transitive-closure equivalence sets can be found from large genetic network with less computational effort. We also offer an efficient Field Programmable Gate Array (FPGAs) implementation platform for the required computation. Application of FPGA processors for

searching equivalence set minimizes experimental delay due to computational intensive data analysis.

Chapter 3 addresses the computational problem in the evolutionary study of DNA sequences. We specifically study different approaches to speedup the phylogenetic tree reconstruction. Phylogeny (phylogenetic tree) is a meaningful representation for the evolutionary history of different organisms and it had been shown tremendous impact to the biological and medical science. Due to the exponentially increasing search space for the optimal Maximum Likelihood (ML) criterion, the phylogeny inference is classified as NP-hard. Heuristic search makes use of the likelihood evaluation function extensively to give score for the candidate solutions. This tree evaluation process is a critical but computationally demanding task. We present the design of a dedicated FPGA-based hardware system that performs ML tree evaluation in a more efficient manner than a software implementation. With simplification of the ML function, a recursive ML evaluation algorithm is proposed. This algorithm can be mapped to FPGAs using digital logics. In addition, based on the DNA state and nucleotide site independence, fine-grained parallelism is introduced into the proposed FPGA-based architecture to provide significant speed-up.

In Chapter 4, we study the FPGAs implementation for one of the basic mechanisms in neural prosthetics. Neuron-machine interfaces such as dynamic clamp and brain-implantable neuro-prosthetic devices require real-time simulations of neuronal ion channel dynamics. We present an FPGA design of a neuromorphic Hebbian synapse, which mimics NMDA and non-NMDA ion channel dynamics observed experimentally in hippocampal neurons. The proposed design can be readily extended to high-speed implementations of dynamic clamp and neuro-prosthetics for replacements of damaged neurons in the brain.

Dynamic programming is a step of crucial importance in real-time decision making reinforcement learning problem where problem solution time can be an implementation bottleneck. In Chapter 5, we consider a distributed computational framework for both discrete-time and continuous-time inference network for solving dynamic programming problems. Interconnected computational units, forming a network, participate simultaneously in the computation while maintaining coordination by information exchange via continuous communication link. The implementation of the value-iteration algorithm of dynamic programming for the expected average cost criterion is presented. We show that the inference

network converges to the optimal Bellman optimality condition, for which the convergence rate can be made arbitrarily fast and is practically independent of the discounting factor and the number of states. Numerical simulation of using such continuous-time inference network for random-walk and stochastic shortest path problems are also described and compared. We also derived a discrete-time version of the inference network that the network can be well mapped to FPGAs for solving dynamic programming. Besides, we investigate the realization of continuous-time inference network using analog CMOS VLSI. Inference cell with embedded computational units based on analog arithmetic components would be developed for verification and realization of the continuous-time inference network model.

In a Markov decision process, knowledge of the state transition probability function and the reinforcement reward is not always available. The agent must interact with its environment directly to obtain information. This is by means an appropriate algorithm, to produce optimal decisions. In Chapter 6 assuming the probability and rewards are unknown, to circumvent the limitation of the dynamic programming, Q-learning, proposed by Watkins, as a simple yet strikingly powerful learning algorithm contributes the on-line reinforcement learning. However, the slow sequential learning process under the conventional Q-learning algorithm would delay the solution time for many time-critical and real-time decision making problems. Based on the framework of the Bellman Inference Network, a novel Q-learning network architecture is proposed. We introduce the exploration and memory components to the original Bellman inference network to form a connectionist network with together the capability of parallel learning and optimization. We found that the Q-learning network outperforms significantly the conventional Q-learning algorithm under a distributed unknown environment. We also proposed two design alternatives for the realization of Q-learning network using FPGAs. The network would be dedicated to fit different applications.

In Chapter 7, we summarize and conclude our studies on FPGA-based architecture design with respect to the problems in the area of bioinformatics, neural prosthetics and reinforcement learning.

Chapter 2

A Hybrid GA-DP Approach for Searching the Equivalence Sets

The computation of transitive-closure equivalence sets has recently emerged as an important step for building static and dynamic models of genetic network. We present a hybrid method of integrating genetic algorithm (GA) with dynamic programming (DP) approach and offer an efficient Field Programmable Gate Array (FPGA) implementation platform for the required computation. This approach converts a computational problem of high dimension into a search problem with DP embedded in GA. The DP computation of transitive-closure forms the basic fitness evaluation in GA, for selecting candidate chromosomes generated by applying basic genetic operators. Small transitive-closure equivalence sets can be found from large genetic network with less computational effort. Mutation and crossover operators are specially designed, that always keeps the chromosome in feasible solution domain. The results show that GA-DP is able to locate the small equivalence set from a large network with less than 100 generation even with a small population size (i.e. 30). In our FPGA implementation, a Boolean Relation Inference Network (BRIN) is embedded in the hardware GA that realizes the parallel transitive-closure computation. Implementation of parallel mutation and crossover operation will also be discussed.

Keywords: Equivalence Set, Genetic Algorithm, Transitive Closure, FPGA

Abbreviations and Acronyms

GA	Genetic Algorithm
ES	Equivalence set

GA-DP	Genetic Algorithm and Dynamic Programming
FPGA	Field Programmable Gates Array
BRIN	Boolean Relation Inference Network
RNG	Random Number Generator

Nomenclature

R	Boolean matrix representing the genetic network
$R(i, j)$	The accessibility from gene i to gene j
R'	Transitive-closure of the genetic network R
$R'(i, j)$	The accessibility from gene i to gene j in the transitive-closure R'
N	The genetic network
n	The size of the genetic network (i.e. the number of nodes in N)
A	The sub-matrix that representing the Boolean relationships of elements in M
A'	The transitive-closure of A
$ER(i, j)$	The evaluation of the element i and j, whether they belong to the same equivalence set
M	The equivalence set
m	The number of elements in the equivalence set
P	Population size of GA
CR	the population of GA, that it is $P \times M$ matrix
$CR(c)$	the c chromosome in the population
$CR(c, g)$	gene g from the chromosome c

2.1 Introduction

The rapid advance in technology such as DNA microarrays makes available voluminous experimental data (Smyth, Yang et al. 2002) for the analysis and development of models for large scale genetic network comprising several thousands of genes. Information extraction and reduction to meaningful patterns from the data has become an urgent but rather computationally demanding task. In (Akutsu, Kuhara et al. 1998; Maki, Tominaga et al. 2001; Kimura, Hatakeyama et al. 2003) the idea of partitioning genes into equivalence sets, from a large network was proposed, with reference to a Boolean matrix R obtained from the gene expression patterns resulting from disruption or forced expression. The equivalence sets allow effective grouping of "closed-loop" genes (where the effects of gene 'a' on gene 'b' and gene 'b' on gene 'a' coexist in the set), and can be derived from transitive-closure computation (See Figure 2.1). Subsequently, dynamic models are estimated from the identified equivalence sets for capturing network behavior (Maki, Tominaga et al. 2001).

While the transitive-closure computation for an n-node genetic network N can be readily obtained by sequential dynamic programming techniques such as Floyd-Warshall or Bellman-Ford algorithm (Cormen, Leiserson et al. 1990), the procedure is computationally intensive for practical network size where the network size is large. Subsequent search to obtain all possible m-node (where m is much smaller than n) equivalence sets is needed based on the result of transitive-closure computation. Since the transitive closure computation is a computational intensive task with $O(n^3)$, it is not efficient and wasting computation effort to find a m-node equivalence sets from a large n-node genetic network ($n >> m$) that requires the computation of transitive-closure for the whole genetic network as an initial step.

Instead of locating the equivalence set by applying dynamic programming on the network, we introduce a searching strategy based on an equivalence set criterion. It is only required to compute the transitive-closure for a candidate solution M, which is a small sub-network from N. The high dimension computational intensive task is broken down into smaller pieces, such that only a small sub-network transitive-closure computation is required instead of a large network.

However, an exhaustive Brute-Force search for testing all the possible m-node sets of candidates solutions on the equivalence criterion from an n-node genetic network is not viable, simply because that the number of combinations increases drastically. However, Genetic

Algorithm (GA) has long been known to be a highly efficient global search procedure than Brute-Force, provided that a meaningful fitness evaluation for candidate solutions (or so-call chromosome in a population) is known (Cheng 1998; Langdon 1998; Leung, Li et al. 1998). In this chapter, we introduce a hybrid genetic algorithm, dynamic programming (GA-DP) approach, which provides an efficient alternative solution to search the equivalence genes from a network of genes. The specific problem in this chapter is to maximize the equivalence set criterion for an m-node network, which is selected by GA from the whole n-node network. The equivalence set is found when GA converges to the optimal solution. Instead of locating the equivalence set in a network by an exhaustive computation, the GA-DP approach can reduce the computation effort in transitive-closure computation and offers a fast searching strategy.

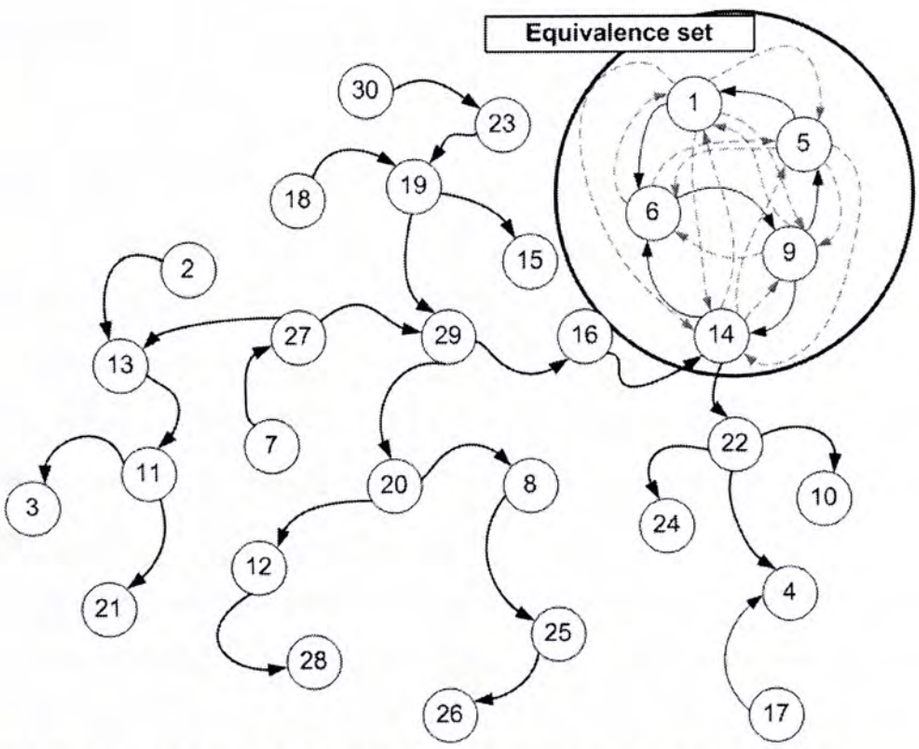


Figure 2.1 A 30-node genetic network, in which node represents gene and solid edges represents the causality relations between genes, while the dotted edges represented the transitive-closure relationships. An equivalence sets {1, 5, 6, 9, 14} is shown, where within this set every gene is affecting the others directly or by transitive-closure.

In addition, we have implemented the hybrid GA-DP algorithm using Field Programmable Gates Arrays (FPGA), a real-time custom computing platform that has seen growing usage for GA (Graham and Nelson 1996; Martin 2001; Shackleford, Snider et al. 2001; Canham and

Tyrrell 2003). Intrinsic fine-grained parallelism for the genetic operators, such as mutation and crossover, in GA-DP are exploited in digital hardware. In addition, a Boolean Relation Inference Network (BRIN) is realized for the parallel transitive-closure computation. This design can effectively reduce the computation complexity of transitive-closure to $O(\log_2(n\text{-}1))$ from $O(n^3)$ (Ng and Lam 2003). With these hardware designs, the GA-DP in hardware can outperform the speed performance of GA-DP in software.

In section 2, we give an introduction of the equivalence set problem. In the section 3, the GA-DP algorithm is presented. The overview of the GA-DP architecture is presented, with further details of individual GA and DP block given in section 4. The simulation results and discussion are in section 5. Limitations and future works are presented in section 6, while conclusion is in section 7.

2.2 Equivalence Set Criterion

In line with the work in (Maki, Tominaga et al. 2001), the accessibility relationships of genes can be represented as an $n \times n$ Boolean matrix $[R_{ij}]_{n\times n}$ where $R_{ij} = 1$ when gene 'i' and gene 'j' affect each other, otherwise $R_{ij} = 0$. Let the node of network N be $V = \{v_1, v_2, \ldots, v_n\}$, and consider a subset $M = \{v_1, v_2, \ldots, v_m\}$ of vertices for some m. An equivalence set M is introduced, in which a set of genes are affecting each other in a group and the group is assumed to be one gene. To evaluate the equivalence relationships, we can base on the transitive-closure matrix $[T_{uv}]_{n\times n}$ of the original matrix where there is a path from vertex v_u to vertex v_v in G denoted as $T_{uv}=1$, otherwise $T_{uv}=0$.

Then, we can use the transitive-closure relationship to evaluate a set of genes in the genetic network, whether they belong to the same equivalence set. Suppose, we have two genes, gene 'a' and gene 'b', if the two genes are forming an equivalence set $ER(a, b)$, that their corresponding transitive-closure will be $T_{ab} = 1$ and $T_{ba} = 1$. This can be extended to an equivalence set with m genes, such that all pairs of elements in M are fulfilled the conditioned of an equivalence set which is given as follows,

$$ER(a,b) = \begin{cases} 1: & T_{ab} = 1 \wedge T_{ba} = 1 \\ 0: & T_{ab} \neq 1 \vee T_{ba} \neq 1 \end{cases} \qquad \forall a,b \in M \text{ and } a \neq b \qquad (2.1)$$

Making partition genes into equivalence set, all genes are included in only one group. In other words, there are no repeating genes in an equivalence sets and no one genes belongs to more than

one equivalence set. The partitioning of a set of equivalence genes is represented as $S = \{S_i\}$, such that the sets are *pairwise disjoint*, that is S_i, $S_j \in S$ and $i \neq j$ imply $S_i \cap S_j = \varnothing$.

To determine whether a set of genes M from the genetic network N is an equivalence set, Eq. (2.1) is readily applied. Firstly, a sub-matrix A is constructed from R, where A is the Boolean relation matrix for elements in M. Then A', which is the transitive-closure of A, is computed by using dynamic programming. Then, we can use A' to evaluate whether the set M is an equivalence set. If two genes are in the same equivalence set, the sum of their corresponding entries in the transitive-closure matrix will be two[1]. As there are $m(m-1)/2$ pairs of the elements in M, and if M *is* an equivalence set, the sums of all entries in the transitive-closure matrix A' will be $m(m\text{-}1)$. In other words, the optimal condition is, that all the entries in A', excluding the diagonal, are one (See Figure 2.2).

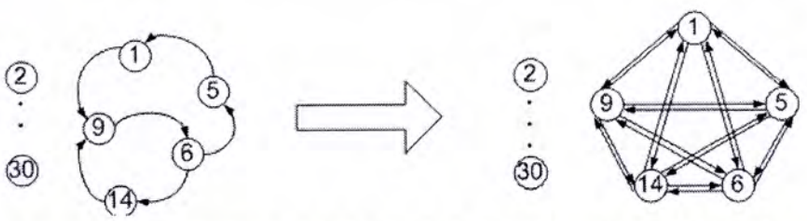


Figure 2.2 A sub-network, which consists of node {1, 5, 6, 9, 14} is extracted from the genetic network N. Transitive closure computation is applied to the sub-network, then the total number of edges is 20, as this sub-network forms an equivalence set.

2.3 Genetic Algorithm and Dynamic Programming

In (Maki, Tominaga et al. 2001), the procedure based on dynamic programming to infer equivalence set from a network has been outlined that the transitive closure of the whole network has to be derived as the first step. After that equivalence sets are extracted from the transitive-closure network subsequently by identifying groups of genes affecting each others. It happens to be the equivalence set searching becomes a time consuming process due to the computationally intensive transitive-closure inference. As an alternative, the equivalence set problem can be formulated into a search problem, that the algorithm intents to search a set of node from the genetic network that the equivalence criterion is maximized.

[1] If the set $\{a, b\}$ is an equivalence set, then the transitive-closure $R'(a, b) = 1$ and $R'(a, b) = 1$, thus $R'(a, b) + R'(b, a) = 2$

The computation of transitive-closure from the candidate set M is only based on the connectivity of vertexes in M. Therefore, we can construct a Boolean matrix $[Q_{st}]_{m\times m}$ represents the accessibility between genes in M. Also an indexing function $\xi(v_i)$ is introduced that re-index the vertex v_i in M where $v_i \in M$. Therefore the relationship between matrix Q and R is given as

$$Q_{\xi(v_i)\xi(v_j)} = R_{v_i v_j} \tag{2.2}$$

We also denote the transitive-closure of Q as Q'. Based on the definition of equivalence set given in Eq. (2.1), the objective of the optimization is to maximize the number edges in the transitive-closure of the sub-graph M that is given as

$$\max_{\forall M} \quad \sum_{i=1}^{m}\sum_{j=1}^{m} Q'_{ij} \tag{2.3}$$
$$\text{where } i, j = \xi(v_k) \text{ and } v_k \in M \ , \forall k = 1, 2, ..., m$$

Under this objective function, the evaluation of the candidate solution, only the transitive-closure of the candidate of equivalence sub-network is necessary to compute while the sub-network is much smaller than the network (n>>m). However, the number of possible solutions is increasing exponentially[2] with the network size, though the dimension of the transitive-closure matrix is reduced. It will also be time consuming to use the Brute-force approach to enumerate all possible solutions.

2.3.1 Genetic Algorithm Formulation

Genetic Algorithm (GA), which is a well-known efficient heuristic search algorithm, which is adopted to search the equivalence set from the network. The idea is that we let the fitness function as the objective function in Eq. (2.3) and using genetic operators to search equivalence sets from the network. Candidate with higher fitness score indicates that the transitive-closure of the chromosome has larger number of edges. Thus GA tends to leave more offspring, which is with large number of edges, in the next generation, and natural selection increases the average density of the chromosomes in the population. At the same time, the mutation and the crossover operators in GA will explore the search space looking for alternative solution.

There is an interesting class of partitioning problems, which require partitioning n object into k categories. One of the categories of evolution programs was based on representing all objects as

[2] The possible solutions equals to select m genes from n genes, therefore the solution space equals to ${}_nC_m$.

a permutation list; special operators can be applied and a decoder makes the decisions on the assignments (Davis 1991). However, in genetic network equivalence set search problem, there are thousands objects to be partitioned. This leads to a very long chromosome string and a possibly slow convergence, as there are many objects to be partitioned. A straightforward approach is to encode the equivalence set as a chromosome, as the size of the equivalence set is much smaller. The partition can be represented as m-strings of integer numbers, $C_i = (x_1, x_2, \ldots, x_m)$ where the j-th integer $x_j \in M$ as a gene in the genetic network. Chromosome C_i is candidate equivalence set of the network. Therefore, the size of the chromosome only is the size of the equivalence set, which is much smaller than the network.

2.3.2 Bounded Mutation

Integer chromosome representation is common used in many evolutionary algorithms. However, most of the operators are dedicatedly designed for meeting the constraint of the problem or objective formulation. For the equivalence criterion in Eq. (2.2), the most important constraint for the candidate solution is that there are no repeating genes in an equivalence sets. As there is no existing genetic operator that can provide mutation and crossover operation that the chromosome will not violate the constraint. Therefore, we develop the bounded mutation and conditioned crossover operator, which provide genetic operation on the chromosomes while the offspring chromosome is within the feasible domain.

A bounded mutation operator is developed that makes the new offspring not violating the equivalence set constraint. As in an equivalence set, there is no repeating gene within one equivalence set. If we replace the gene randomly in the mutation process, there is a chance for a gene to appear within one chromosome more than once. It is an infeasible solution for the fitness evaluation function. The idea is to randomly replace a node in the chromosome by a new node, which is different from all existing nodes in the chromosome.

It is defined as follows. For a parent $\boldsymbol{x}$, if the element x_k was selected for this mutation, the result is $\boldsymbol{x}' = (x_1, \ldots, x_k', \ldots, x_m)$ where,

$$x_k' = y, \qquad \text{where } y \in V - \{x_1, x_2, \ldots, x_m\} \tag{2.4}$$

where V is the complete set of nodes in network N and y belongs to the set of nodes, that the nodes in the parent chromosome are removed. The bounded mutation generates offspring with no repeating genes as the new 'gene x' belongs to the set that offspring genes are removed.

2.3.3 Conditioned Crossover

The crossover operation has a similar problem that is faced in the mutation, as random genes exchanging between two chromosomes will probably result in gene duplication in the offspring. For two parents are defined as follows: if $\boldsymbol{x}_1 = \{x_1, x_2, \ldots, x_m\}$ and $\boldsymbol{x}_2 = \{y_1, y_2, \ldots, y_m\}$ are crossed after the k-th position, the resulting offspring are: $\boldsymbol{x}_1' = \{x_1, x_2, \ldots, x_k, y_{k+1}, \ldots, y_m\}$ and $\boldsymbol{x}_2' = \{y_1, y_2, \ldots, y_m, x_{k+1}, \ldots, x_m\}$. However, if $\{x_1, x_2, \ldots, x_k\} \cap \{y_{k+1}, \ldots, y_m\} \neq \varnothing$ and $\{y_1, y_2, \ldots, y_k\} \cap \{x_{k+1}, \ldots, x_m\} \neq \varnothing$, the offspring $\boldsymbol{x}_1'$ and $\boldsymbol{x}_2'$ will be infeasible solution. Therefore, comparison is applied between every element of the two parents. If there is a gene x_i in $\boldsymbol{x}_1$ equals to y_j in $\boldsymbol{x}_2$, crossover will be prohibited in the i-th position of chromosome $\boldsymbol{x}_1$. Thus, we denote the single offspring from the two parent chromosome $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ as $\boldsymbol{x}' = \{x_1, x_2, \ldots, x_k, s_1, \ldots, s_{m-k}\}$, such that s_i is given as

$$s_i = \begin{cases} x_{i+k}, & \text{if } y_{i+k} \cap \{x_1, x_2, \ldots, x_m\} \neq \varnothing \\ y_{i+k}, & \text{if } y_{i+k} \cap \{x_1, x_2, \ldots, x_m\} = \varnothing \end{cases} \tag{2.5}$$

The offspring from the two parents generates offspring in a domain of the feasible solution that can be evaluated by the equivalence criterion given the parents are feasible set. The following example is used to illustrate the idea of conditioned crossover

Example: Suppose we have two parent chromosomes, $\boldsymbol{x}_1 = \{1, 3, 7, 11, 19\}$ and $\boldsymbol{x}_2 = \{2, 14, 19, 23, 3\}$. A condition crossover is applied directly on these two chromosomes, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. Suppose the crossover is at the first position, which means starting from the second genes in $\boldsymbol{x}_1$ can be exchanged to $\boldsymbol{x}_2$. As the gene 19 and 3 in the 3-rd and 5-th position of $\boldsymbol{x}_1$ that are appearing in $\boldsymbol{x}_2$. Then the offspring is = {1, 14, 7, 23, 19}, in which no crossover on the 3-rd and 5-th position.

2.3.4 Implementation

A implementation for bounded mutation would be start with preparing a list, in which is indexes of all nodes. Then remove the node, which can be found in the chromosome, from the

list. Lastly, a node is randomly drawn from the list to replace one of gene in the chromosome. However, this approach could be slow and memory consuming, as the length of the list could be lengthy once the network is large. An alternative approach is to apply a sorting operation on the chromosome as first step. After that a new gene can be generated based on any interval between two genes, which the new gene is not repeating with the existing genes. The alternative approach does not require prepare a lengthy list.

The chromosome is sorted as the first step. Then there will be m+1 intervals between the m sorted genes[3], as each gene $CR(c, g)$ in chromosome c is represented by an integer value from 1 to n. Then two random numbers are generated, that *rand_pos* is for deciding which interval to use and the *rand_node* is used for generating new integer between the chosen intervals. Lastly, one of the nodes from chromosome is chosen randomly to be replaced by this new node. The process is repeated for all the chromosomes, which are subjected to mutate. The pseudo-code of the bounded mutation is presented in Figure 2.3.

```
Bounded Mutation (CR)
For c = 1 to P
Sort(CR)
If (rand > m_rate)
rand_pos := round(rand*(m+1))
                    if(rand_pos=1)
rand_node=round(rand*( CR(c, rand_pos)))
                    else if(rand_pos=l)
rand_node=round(rand*( n-CR(c, rand_pos+1))
else
rand_node=round(rand*( CR(c, rand_pos)-CR(c, rand_pos+1)))
end
rand_pos := round(rand*m)
CR(c, rand_pos) = rand_node
end if
Return CR;
end
```

Figure 2.3 The pseudo-code of bounded mutation. In all pseudo-code, we use *rand* to represent random generated number in (0,1).

Conditioned crossover is performed with probability *c_rate*. After a first parent $CR(c)$ has been chosen and given rise to an offspring chromosome in the next generation, a decision is made whether or not to allow a second parent to recombine with this new offspring individual. With

[3] We represent each gene as a positive integer from 1 to n. Once m genes in the chromosome are sorted, we can have m+1 interval, which is the difference between two genes (integers).

the probability *c_rate* for each chromosome, a second chromosome $CR(p\text{-}c+1)$ is selected from the parental population. A preprocessing task is performed before the crossover operation is applied on these two chromosomes that it is to ensure that the offspring will not violate the equivalence set constraint. A temporary array *CR'*, which records whether the elements in the first chromosome has elements appearing in the second chromosome. The array *CR'* can be accomplished by comparing the two parent chromosomes and if the element in the i-th position of the first chromosome appearing in the second chromosome, $CR'(i)=1$, otherwise, $CR'(i)=0$.

```
Crossover (CR)
        For c = 1 to P
                If (rand > c_rate)
                        For i = 1 to m
                          For j = 1 to m
                             If (CR(c, i)=CR(p-c+1, j))
                                 CR'(i) = 1
                             Else
                                 CR'(i) = 0
                             endif
                          endfor
                        new_CR(c) = Condition_Crossover(CR(c), CR(p-c+1), CR')
                End
        End
        Return new_CR;
End

Condition_Crossover(CR1, CR2, CR')
for i = 1 to rand_pos
   if(CR'=0)
        CR1(i) = CR2(i);
   end if
endfor
Return CR1;
```

Preposessing task to prepare *CR'*

Figure 2.4 The pseudo-code of the conditional crossover

2.4 Digital Implementation of GA-DP

The GA-DP approach can be implemented using digital logic, which provides efficient hardware logical operation and parallel computation. In this section, we present the hardware implementation of the GA-DP algorithm, that (1) system architecture overview, (2) parallel computation for transitive-closure and (3) the design of the genetic operators will be discussed specifically.

2.4.1 System Overview

Figure 2.5 shows a GA-DP system overview. It consists of 2 major components: (1) the GA and (2) the fitness evaluation. There are three GA operators, which are selection, crossover and mutation, and they are working independently and sharing with the same memory. Data communication relies on the *temp_RAM*, which stores the chromosomes. Each operator retrieves the chromosomes from the *temp_RAM* before execution, and stores the results back to the temp_RAM after finishing the work. An asynchronous mechanism is adopted, that there are an *enable* input and a *ready* output within each functional block. The genetic operators will start to run only after the enable signal is 1, and its ready output will be 1, once it has finished the work. Theses control signal are centralized and controlled by the *System Controller.* It ensures the functional blocks are executed sequentially.

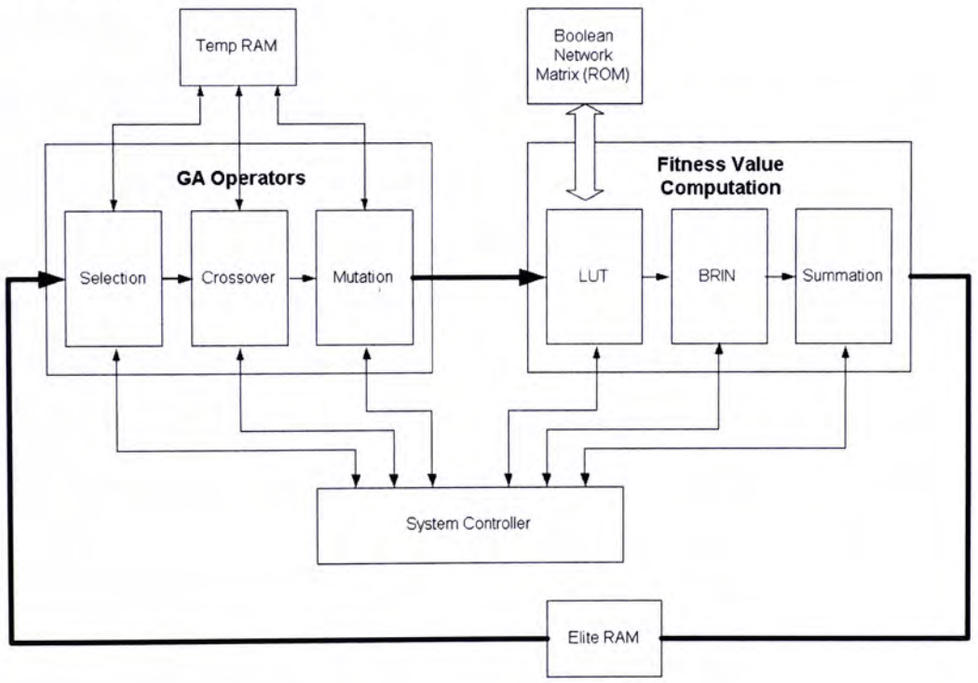


Figure 2.5 The architecture of the GA-DP hardware system

The fitness value computation module is comprised of 3 sub-modules, which are the sub-matrix construction, transitive-closure computation and the summation operation. The first sub-module is to reconstruct the sub-matrix from the chromosome, that it requires reading the values from Boolean matrix. It is tightly coupled with the ROM, which stores the Boolean matrix *R*. The Boolean Matrix is mapped to a vector with aligned by rows while storing in the ROM. The

address for the accessibility value in the Boolean matrix can be easily calculated by the Eq. (2.6), where i and j are the row and column of the Boolean matrix respectively.

$$R(i,j)=\begin{cases} j+i\times n\ , & \text{if } i<j \\ j+i\times n+1\ , & \text{otherwise} \end{cases} \tag{2.6}$$

where n is the network size.

The sub-matrix is readily used for transitive-closure computation. Though it is possible to use the Floyd-Warshall algorithm as implemented in the software, it cannot utilize the parallelism capability in hardware. A Boolean Relation Inference Network (BRIN) is realized, that it provides a parallel architecture for the required computation. We will discuss the design in the next section in details. BRIN inputs the sub-matrix and outputs its transitive-closure. The last step is to sum up all values from this output. Then the result, which is the fitness value of the chromosome, is stored in the RAM.

2.4.2 Parallel Computation for Transitive-Closure

Floyd-Warshall algorithm is adopted in software programming for GA-DP. The computation complexity is in $O(n^3)$. A Boolean Relation Inference Network (BRIN) is introduced, that in the hardware implementation the transitive-closure computation can be efficiently parallelized. The computation complexity is reduced to $O(\log_2(n\text{-}1))$. BRIN is able to solve different optimization problems, which included critical path, and transitive closure problem (Lam and Tong 1996; Ng and Lam 2003). The architecture has been found to show promise in obtaining the global optimal solution in logarithm time in the FPGA implementation (Lam 1996; Lam and Su 1996).

Transitive closure is a special case of the general shortest path problem, in which one has to find a route between two nodes with minimum arc cost when given a set of nodes and arcs. However, in a transitive closure calculation, the arc cost in the network becomes a binary representation. For all-pairs transitive closure problem, the transitive-closure has to be derived by dynamic programming like Bellmen-Ford and Floyd-Warshall (Cormen, Leiserson et al. 1990). The complexity of these algorithms is $O(n^3)$, which is computation costly. An n-node network is represented in n row square matrix R form with entries for n^3 times either 1 or 0. For the Bellmen-Ford algorithm, network cost is updates sequentially by the relaxation equation.

However this is only for the single-destination problem. When considering the all-pair problem, the network update is repeated n times.

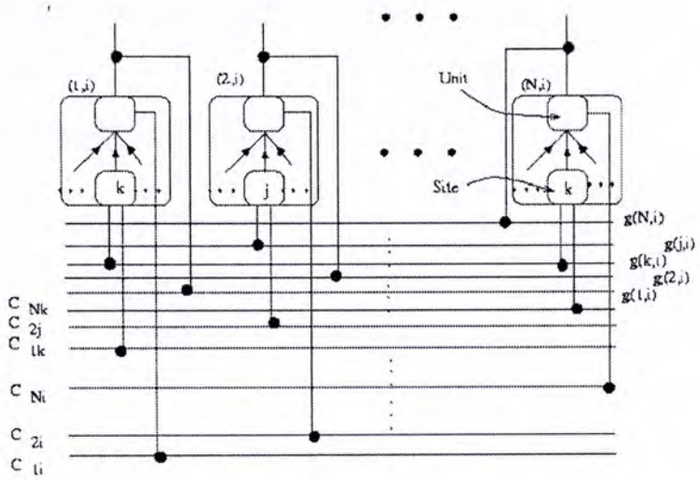


Figure 2.6 Connectionist network topology diagram of BRIN. The diagram shows three computation units of BRIN, which consists of the unit function and the site function. For transitive closure calculation, site function is implemented by an AND-gate while the unit function is implemented by an OR-gate.

A Boolean Relation Inference Network (BRIN) was implemented with a parallelized Bellmen-Ford algorithm to speedup the calculation from $O(n^3)$ to $O(log_2(n-1))$. A connectionist network, which can realize the network updating process, parallelizes the required network relaxation step. One way to compute the transitive closure is based on the Bellman-Ford formulation, that the transitive-closure can be stated in the semi-ring form $d_{xz} = \oplus_y [d_{xy} \otimes d_{yz}]$ where $\oplus$ is the summary operator and $\otimes$ is the extension operator and d_{ij} is the cost for the path from i to j. In the transitive closure problem, 'OR' operator substitutes $\oplus$ and 'AND' substitutes $\otimes$. The transitive pathway from node x to z (d_{xz}) is the binary arc cost from node x to node y (d_{xy}) and from node y to node z (d_{yz}), the 'AND' operator ensures there is a possible path from x to z through y.

The general BRIN framework in Figure 2.6 can readily implement the semi-ring structure stated. There is a site function and an unit function in a BRIN computational unit. The unit function implements the OR-gate while the site function implements the AND-gate. The n-1 units can be constructed in a parallel way such that the output of a unit is a feedback to the input of other units. In the worst case, the computation time for such parallel processing is $O(log_2(n-$

1)). On a higher level, Bellmen-Ford modules can be run in parallel for all-pairs transitive closure computation since all BRIN computational units are independent. Therefore even for computing all-pairs transitive closure, a complexity of $O(log_2(n-1))$ is required.

2.4.3 Genetic Operation Realization

As there are no repeating elements within one equivalence set, the mutation and crossover operator are designed that the offspring generated will not infringe the constraint. The mutation operator consists of 3 major components, which are the random number generator (RNG), node generator and the mutation position decision unit. The RNG is comprised of an XOR gate and a circuit of linear feedback shift registers. Using feedback from the various stages of an k-bit shift register, connected to the first stage with an XOR gates, a sequence of 2^k-1 patterns that have the characteristics of randomly generated numbers (Peterson and Jr. 1972). The pseudorandom binary sequence generation gives a simple method of generating random number for the mutation operation. This random number is readily mapped to the zero to one interval, that the number is used as the random number *rand* from Figure 2.7.

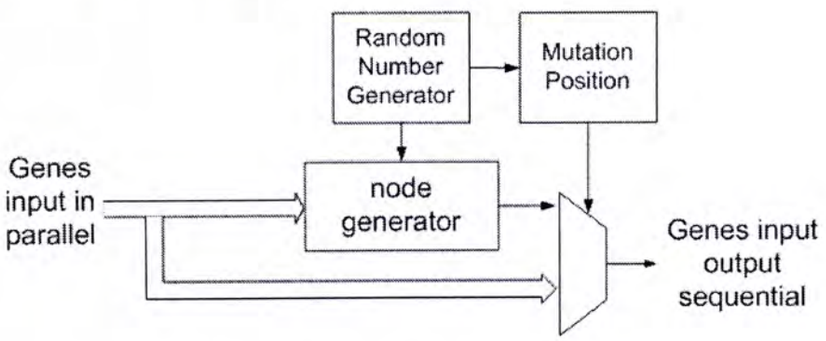


Figure 2.7 The data flow of the mutation operator

The node generator comprises of a sorting network and the new node generation circuit. The sorting network sorts the genes in the chromosome in ascending order. As applying parallelism, the sorting can be finished in logarithm time (Cormen, Leiserson et al. 1990). Then, the genes are inputted into the new node generation circuit, which generate new node as the sequential execution in software algorithm. The new node will replace the once of the gene from the parent chromosome to produce an offspring with mutated gene.

The crossover unit inputs two chromosomes in parallel and output a new offspring. The unit has two sub-module, which the first sub-module is to find the repeating elements in the two parent chromosomes, while the second sub-module is for realizing the condition crossover. We have *m* comparison units (*CM*) for testing all combinations of genes between the two parent chromosomes. There are *m-1* comparison modules in each *CM*, thus the comparison can be done in parallel. The output of each CM_i will be either 1, if there exists a gene in the second chromosome, which equals to the *i* gene in the first chromosome, or otherwise 0. This signal will inform the conditional crossover unit which gene in *CR(c)* is not viable for the crossover to the *CR(p-c+1)*.

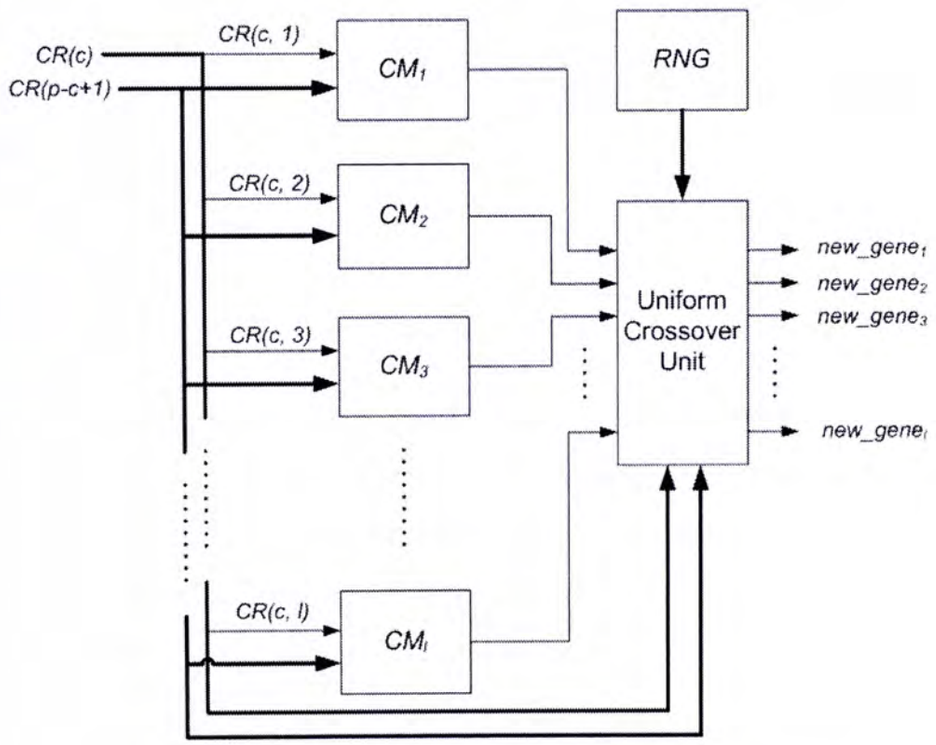


Figure 2.8 Data flow of the Crossover operation

The uniform crossover unit comprises *m* new gene generation circuit running in parallel, that one of them has been shown in Figure 2.8. The crossover happens following simple rule, that the gene *i* from *CR*(*c*) will appear in the offspring only if the *rand_pos*[4] is greater than *i* and CM_i is 1. If these conditions are satisfied, the multiplexer will choose the *CR*(*c*), otherwise, *CR*(*p*-*c*+1) will be chosen.

[4] rand_pos is the position, which begins the crossover. It is generated by round(rand*l), where rand is the random number between 0 and 1.

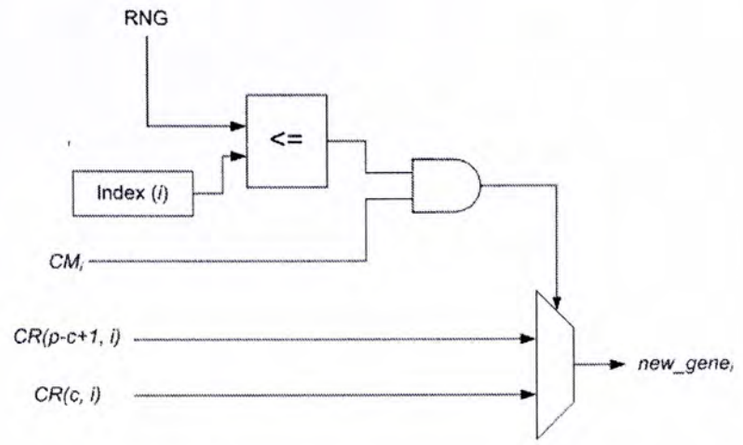


Figure 2.9 The schematic for generating one *new_gene* in the crossover unit

For each comparison unit (CM), there are m-1 comparators running in parallel, and followed by the OR operators.

2.5 Discussion

We investigate the potential of the GA-DP algorithm in searching small equivalence set from a large genetic network. Consider a typical network of 30-node with one 5-node equivalence set. Let us relate the problem with using GA-DP algorithm for searching the equivalence set. In this case, the size of the chromosome equals to five, and hence the genes are labeled from one to thirty. Hence, the optimal criterion for this problem becomes twenty.

As observed from a typical run (Figure 2.10) under 100 GA generations, the GA-DP converges to the optimal solution at the 40-*th* iteration[5]. In this example, GA-DP is able locate the equivalence set from the network, which is much large than the equivalence set. The result is encouraging as GA-DP is very effective in searching the equivalence set from a 30-node network.

[5] The population size is 50, the mutation rate is 0.05 and the crossover rate is 0.2.

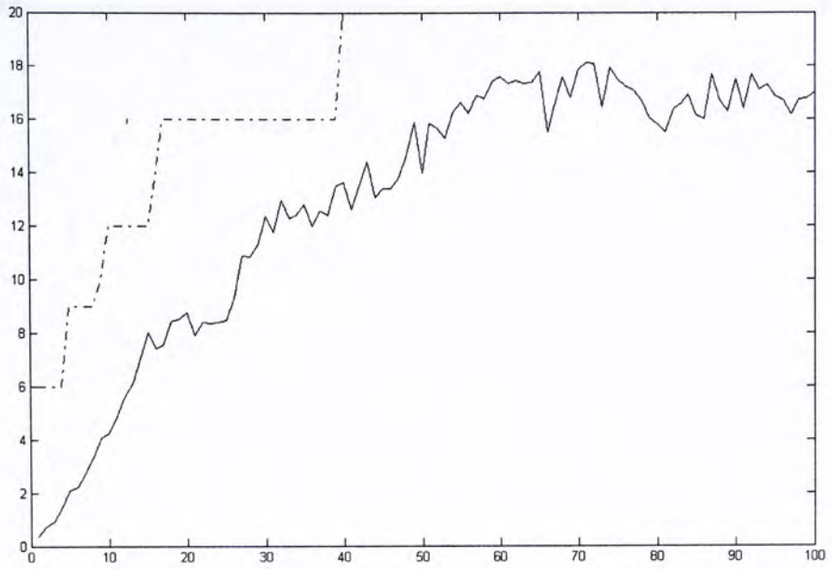


Figure 2.10 Software simulation result of the GADP on searching the 5-node equivalence genes from a 30-node network. The dotted line is the best fitness value while the solid curve is average fitness value among the population in every iteration.

On another experiment, we investigate the relationship between the number of GA generations required to find the equivalence set and the population size of GA. Consider two experimental setups with networks in size 30 and 50. We change the population size from 10 to 100. The generations, which the GA-DP locates the equivalence set, is measured. We randomly generate network of different topology and the equivalence set is placed randomly in the network. For each population size, we test on 50 different networks. Figure 2.11 shows the averaged results of 50 runs. It can be observed that the generation number decreases with the population size for both setups. The curve decreases rapidly when the population size is ranged from 10 to 30. After that, the generation number decreases slightly. In most of cases, it appears that if the population size of GA-DP is greater than 30, the GA-DP is able to locate the solution within 100 generations for a network size of 30 and 50. This is attributes to the fact that the large population size can largely diversify the search space of equivalence set from a network, and GA-DP can be more effective to locate the optimal solution with larger population size.

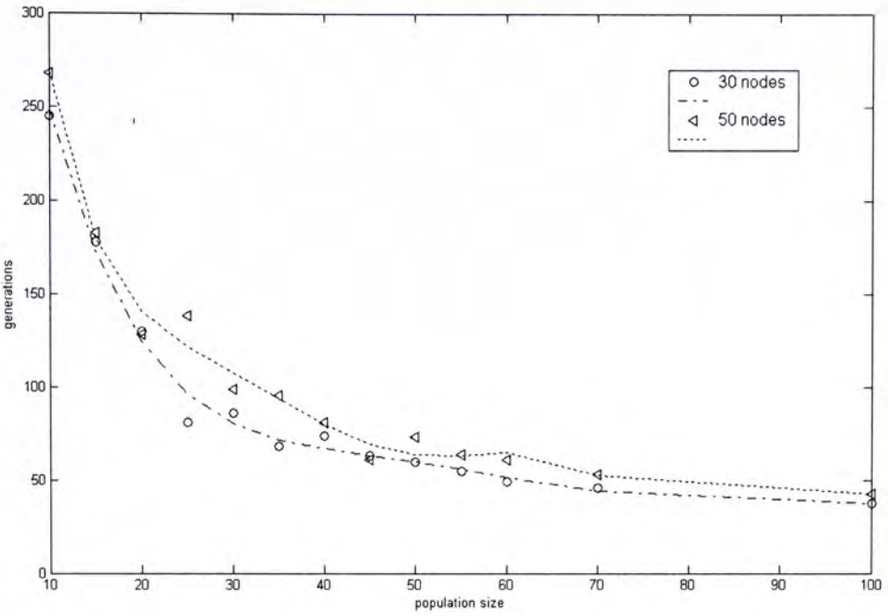


Figure 2.11 The number of generations versus population size on the GA-DP algorithm. Two experimental setups were tested, that the network size are 30 and 50 respectively.

Furthermore, experiments have confirmed that GA-DP scales well with respect to the size of the network. Figure 2.12 presents the results of the computation time for GA-DP to find the equivalence set from a network of size varied from 30 to 500. The GA-DP algorithm shows little increase in time for handling larger network. On the other hand, we found that the computation time for original approach[6], which evaluates the transitive-closure of the whole network before extracting the equivalence set, increases rapidly with the network size. From the Figure 2.12, it is interesting to find out that when the network size smaller than 150, TC spends less time than GA-DP. Since the fitness evaluation and the genetic operations of GA-DP is approximately a constant time, the TC spends less time than this constant when network size is smaller than 150.

[6] We called it the TC approach, which TC stands for transitive-closure

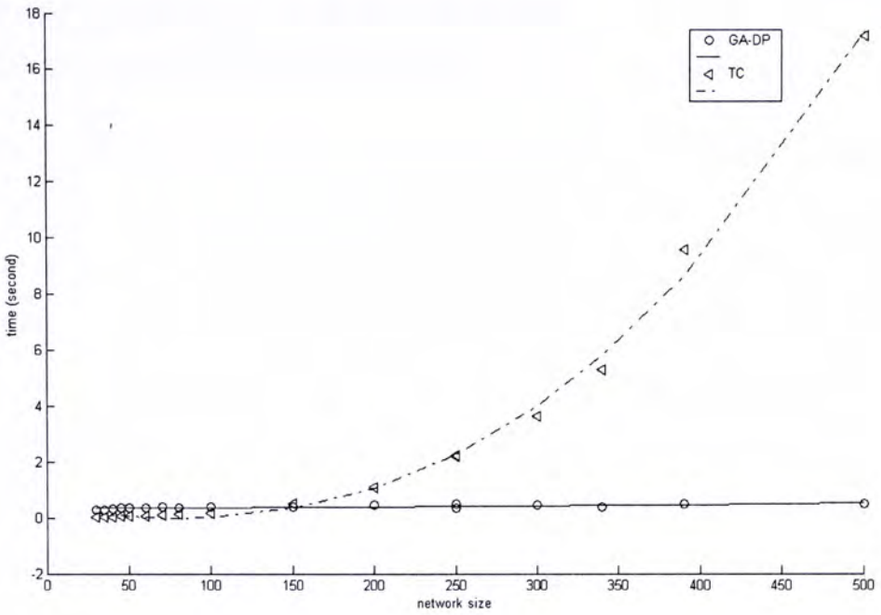


Figure 2.12 The comparison on the computation time between the GA-DP and the transitive-closure approach from (Maki, Tominaga et al. 2001) for searching the equivalence set from network of different size

We had successfully implemented the GA-DP system into the one FPGA processor. The hardware system is tested on a 30-node genetic network problem. The system found the equivalence set using 0.01361 seconds while the 0.26 second is used in the software for the same problem. The hardware system offers 20 times acceleration in this example. This is mainly attributed to the fine-grained parallelism on the operations in GA-DP. The system consumes 64% of the FPGA resources called slices[7] in the XESS XSV-800 FPGA prototyping board with a Virtex-800 devices.

2.6 Limitation and Future Work

So far the GA-DP approach has been shown that is able to find the equivalence set of small size from a large network. Currently, we only investigate the case of single equivalence set existing in the network. In a more practical manner, multiple equivalence sets can be identified, which seems to be a more difficult problem than single equivalence set searching. This is because the existing design of GA-DP will converge to one equivalence solution only. Also

[7] Slice is regarded as a measure of hardware logic consumption in FPGA.

multiple equivalence sets may create a more complicated landscape of search space, that GA-DP may not always reach the optimal solution, has been considered as future work.

In addition, the work on digital FPGA implementation indicated that performance of GA-DP can be improved through parallel digital implementation. In reviewed work concerning FPGA for GA implementation, indicated that performance improvement over a software implementation of two or three orders of magnitude can be achieved by introducing GA in hardware. Though we have realized fine-grained parallelism in the system, our work described so far has not achieved this level of improvement. This is probably due to the fact that parallelism has not been introduced in the chromosome mutation and fitness evaluation, that chromosomes are mutated and evaluated sequentially. The hardware implementation shows that 64% of slice (i.e. equivalent to the number of logic gates) has been used in a XCV-800 FPGA device. There is room for improvement on GA-DP with more parallelism implemented in FPGA.

2.6 Conclusion

The search of equivalence sets in large scale genetic network emerges as an important step in gene expression data analysis. Our proposed GA-DP algorithm has shown to give an efficient solution to the rather computation demanding problem. A bounded mutation and conditional crossover operator are introduced to constrain the offspring of GA within the feasible solution domain. We found that the GA-DP algorithm can effectively locate the small equivalence set from large genetic network in few generations. Also, the GA-DP is more efficient than mere the dynamic programming approach, which requires to derived the transitive-closure of the whole genetic network, when the network size is large (>150 nodes). We also provide a realization of GA-DP algorithm and Boolean Relation Inference Network (BRIN) for the required dynamic programming embedded in GA using FPGA platform. Study was shown that performance of GA-DP can be improved through parallel digital implementation significantly.

Chapter 3

An FPGA-based Architecture for Maximum-Likelihood Phylogeny Evaluation

Study the evolutionary history of organism, namely phylogeny or phylogenetic tree reconstruction, is an important but computational intensive task. Due to the exponentially increasing search space based on the optimal criterion, maximum-likelihood, the phylogeny inference is classified as NP-hard. Heuristic search makes use of the probabilistic evaluation function repeatedly to give score for each candidate solution. The evaluation becomes a critical but computationally demanding task. We propose a dedicated FPGA-based hardware design for the critical computation. Modified floating point arithmetic and parallel recursive architecture are proposed for supporting the precision demanding probabilistic computation and the basic data structure operation of phylogenetic tree respectively. The bit-error computation of the proposed arithmetic system has been studied and the error formula has been derived. Significant acceleration for the tree evaluation can be obtained based on the proposed hardware architecture.

Keywords—phylogenetic tree reconstruction, maximum-likelihood, pruning algorithm, hardware architecture, Field Programmable Gate Array (FPGA)

3.1 Introduction

Molecular phylogeny studies the evolutionary processes of different organisms using molecular data (i.e. DNA and protein), which provides significant information for molecular biologists to understand bacteria and virus behaviors (Yang 2001; Rambaut, Posasa et al. 2004). Numerous interesting and important examples of evolutionary studying based on DNA analysis were presented during the last few decades. Such studies often have significant implications on virus identification, drug design and genomic analysis (Kishino, Miyata et al. 1990; Bader, Moret et al. 2001; Yang 2001). In (Drosten et al. 2003), a phylogenetic tree was generated from DNA sequences of Severe Acute Respiratory Syndrome (SARS) virus and other coronavirus. By using phylogeny analysis, the SARS virus was identified as a novel virus that is a close relative to some known coronavirus. Besides, Rambaut et. al, showed that phylogeny analysis has its profound implication in the pharmaceutical research that to study the evolution of virus can predict drug-resistance and immune-escape of mutations (Rambaut, Posasa et al. 2004). Also the strong drug adaptation of HIV virus can be explained by its fast mutation rate. In (Worobey, Santiago et al. 2004), the origin of HIV is studied and found through meticulous phylogeny analysis. To decipher HIV interaction with the immune system and developing effective control strategies, close relatives of the virus are studied using the phylogeny (Rambaut, Posasa et al. 2004). In addition, for the virology and pharmaceutical application of phylogeny, it has been shown that evolutionary studies of species can benefit the genomic studied and the regulatory genetic network as well (Eisen and Fraser 2003).

The evolutionary relationships between organisms are often encoded as a bifurcating unrooted[1] tree, which are made by meaningful arrangement of nodes and branches (See Figure 3.1). Nodes at the tips of the branches (external node) correspond to a gene or an organism while internal nodes usually represent an inferred common ancestor, which gives rise to two independent lineages at some point in the past. In Figure 3.1, node A, B, C and D are external nodes that represent species of which molecular sequence data is available. In contrast, the internal node E and F represent inferred ancestors for which empirical data is not available. A phylogenetic tree can be inferred from nucleotide sequences, or DNA (Deoxyribonucleic acid),

[1] In *rooted trees* a single node is assigned as a common ancestor, while *unrooted trees* only specify the relationship between nodes and say nothing about the direction in which evolution occurred.

in which each DNA only exists as one of the four possible bases or so-called "states": guanine (*G*), adenine (*A*), thymine (*T*) and cytosine (*C*).

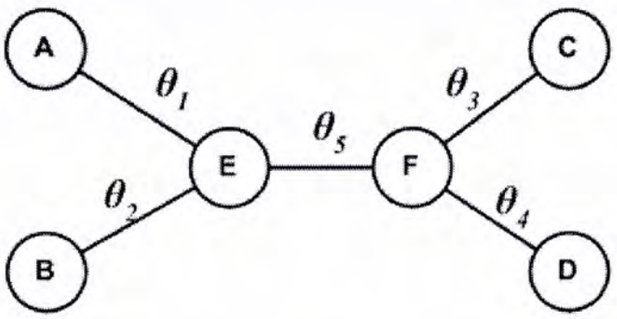


Figure 3.1 A 4-taxa[1] unrooted bifurcating tree

Although there are a variety of methods for phylogenetic tree reconstruction, Maximum Likelihood (ML) becomes one of the most popular approach for phylogeny analysis (Felsenstein 1981). The ML approach reduces the complexity of biological mutation process to simple stochastic models with small number of parameters. The likelihood value under the ML model offers an evaluation for the phylogenetic trees, which is a probability of the observed DNA sequences conditioned on tree topology T and model parameter M. The likelihood L is a conditional probability $P(D|T,M)$ where D is the observed nucleotide sequences given as an $n \times l$ matrix with its elements $D_{rs} \in \{A,T,G,C\}$. The ML methodology is then used to determine the best match for the tree topology T and model parameters M, such that it can maximize the likelihood probability to give the observed nucleotide sequences under the ML criterion.

However, it is a difficult task to find the optimal solution based on the ML criterion, simply because of the exponentially growth of the possible tree topologies with the increasing number of taxa[2]. The possible unrooted, bifurcating n-taxa tree topologies S is

$$S(n) = \prod_{i=3}^{n} (2i-5) \quad \text{i.e.} \frac{(2n-5)!}{(n-3)!2^{n-3}} \tag{3.1}$$

corresponding to nearly 16 billion different trees for 12 taxa and 3×10^{84} trees for 55 taxa. The optimal phylogenetic tree search problem is regarded as NP-hard (Lemmon and Milinkovitch

[2] Taxa is a general term referring to any kind of taxonomic unit including DNA sequences and nucleotide site.

2002) which implies that no known algorithm can find the optimal solution in polynomial time. Heuristics are often used for searching a near optimal tree within a reasonable time, which essentially apply hill climbing or genetic algorithm approaches as the search strategies (Strimmer and Haeseler 1996; Swofford, Olsen et al. 1996; Lewis 1998; Lemmon and Milinkovitch 2002). Applying the heuristics can reduce the search space for a near-optimal solution; the tree evaluation is computationally demanding and is being used repeatedly. In general, the computational cost of the likelihood, which accounts for the greatest portion of the execution time (i.e. 95% in sequential execution) (Stamatakis, Ludwig et al. 2002).

Overall computational cost for the phylogeny solution consists of two parameters: (1) evaluation time per tree topology and (2) number of candidate solutions to evaluate, which can be simply modeled as

$$\text{Overall time} = \text{Evaluation time per tree topology} \times \text{number of tree topologies} \quad (3.2)$$

If the tree evaluation process can be speedup, the overall searching time can be reduced. In this chapter, we address this issue where a speed-up strategy is proposed by introducing dedicated hardware for the computational intensive evaluation function.

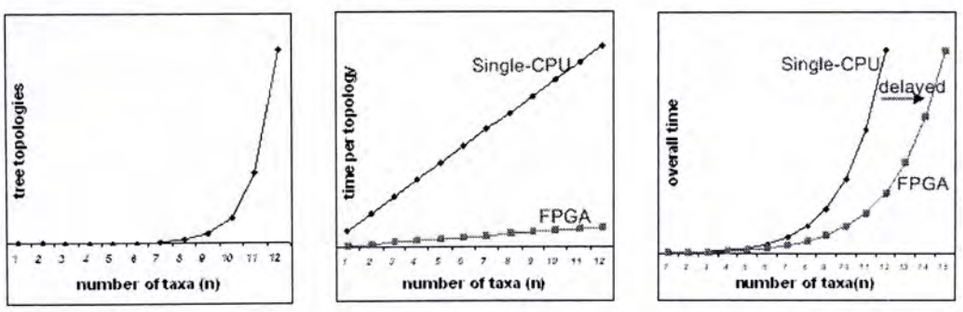


Figure 3.2 a.) The topological space of the phylogenetic tree grows exponentially with the problem size. b.) the FPGA-based processor for ML evaluation provides hundreds acceleration when comparing to the software implementation. c.) The overall time in phylogeny reconstruction can be reduced that in a reasonable time, the FPGA solution can handle problem of larger size.

The idea is illustrated in Figure 3.2. The first diagram is the number of tree topologies against the number of taxa. It grows exponentially even using heuristics search strategy, which is the same regardless of the implementation platform. The second diagram shows the computation time for tree evaluation against the number of taxa. By using dedicated hardware device, the tree

evaluation time can be reduced. As a result, the overall computation time can be reduced (in the third diagram). Though the computation time is still growing exponentially with dedicated hardware, the curve is delayed. It implies that the solution with dedicated hardware would have potential to solve problem of larger scale in reasonable time which is not feasible using desktop computers.

However, it is challenging to implement the evaluation function in hardware. Firstly, the probabilistic maximum-likelihood evaluation is precision demanding. The fixed-point arithmetic architecture in FPGAs would be difficult to support the precision requirement. Secondly, in hardware, there is lack of dynamic data structure, such as binary tree and stack, which are trivial basics in software implementations. Thus, it is difficult to implement the basic phylogenetic tree operations, such as binary tree traversal, which is a critical step in the tree evaluation. Thirdly, logarithm and exponential evaluation are required, which are essentially difficult to be implemented in hardware with high precision requirement and with limited hardware resources.

To circumvent the limitations and difficulties, we simplified the phylogeny likelihood evaluation function as recursive routine. A pseudo-binary tree data structure is used to represent a phylogeny that can be mapped to the memory effectively. Parallelization is introduced in the hardware architecture that provides significant speedup when comparing to the sequential execution in desktop computers. On the other hand, we introduce a simplified floating point number representation scheme, in which a high precision and dedicated for the probabilistic computation can be obtained. The dedicated processor may be readily adapted to the heuristics search algorithms to improve the computational speed of the tree evaluation, and thus the overall phylogenetic tree reconstruction task can be speedup.

3.2 Maximum-Likelihood Model

We consider the likelihood value of interest as a probability function which has three inputs: an unrooted bifurcating tree T, a molecular substitution model M, and n aligned DNA sequences with l based pairs (i.e. each individual base pair is called a *site*), given as an $n \times l$ matrix D. The substitution model has a set of parameters that defines the rate of mutation from one base DNA to another. Following (Felsenstein 1981), the probabilistic likelihood is defined as

$$\ln L = P(D \mid T, M) = \sum_{s=1}^{l} \ln P(D_s \mid T, M) \tag{3.3}$$

where $D = [D_{rs}]_{n \times l}$, $D_{rs} \in \{A, T, G, C\}$, D_s is the site pattern of D at site s (i.e. the s^{th} column of D), and $P(D|T,M)$ is the conditional probability calculated from the specified substitution model. M and tree T are yet to be determined.

Although the likelihood probability $P(D|T,M)$ can be expressed in a general form, it is presented here with an example. Consider a given tree T with 4 taxa (see Figure 3.1) with reference to the site pattern D_s; the lengths of the branches of the tree are θ_i. All nodes in the tree are labeled $\{A, B, C, D, E, F\}$, as there is a total of $2n-2$ nodes in an unrooted bifurcating tree. The specific states (here the number of possible states for DNA is four) of nodes A, B, C and D are simply denoted as x_1, x_2, x_3 and x_4 as given in D_s, and the states of nodes E and F are denoted as x_5 and x_6 respectively. The likelihood probability for site pattern D_s equals to

$$P(D_s \mid T, M) = \sum_{x_5=1}^{c} \sum_{x_6=1}^{c} \pi_{x_5} P_{x_5 x_1}(\theta_1) P_{x_5 x_2}(\theta_2) P_{x_5 x_6}(\theta_5) P_{x_6 x_3}(\theta_3) P_{x_6 x_4}(\theta_4) \tag{3.4}$$

where P_{ij} is the transition probability which represents the probability of a nucleotide changing from state i to j (e.g. if i equals to state A and j equals to state C then P_{ij} equals to the probability of a nucleotide mutate from state A to C). π_i is the priori probability for state i. The likelihood $P(D_s|T,M)$ is the product of five transition probabilities and the prior probability representing 5 incurred mutations (see Figure 3.1). The expression will have 16 terms, and in general the expression for n taxa will have 4^{n-2} terms, which can be increasing exponentially. Fortunately, Eq. (3.4) can be simplified by moving the summation signs rightwards (See Eq. (3.5)) to reduce the number of terms. Eq. (3.5) gives an exact evaluation of the tree topology which equals to Eq. (3.4).

$$P(D_s \mid T, M) = \sum_{x_5=1}^{c} \pi_{x_5} P_{x_5 x_1}(\theta_1) P_{x_5 x_2}(\theta_2) \left(\sum_{x_6=1}^{c} P_{x_5 x_6}(\theta_5) P_{x_6 x_3}(\theta_3) P_{x_6 x_4}(\theta_4) \right) \tag{3.5}$$

A "pruning" algorithm described in (Felsenstein 1981) can effectively compute the likelihood value based on this idea and it is formally formulated by introducing the idea of *partial likelihood* (Adachi and Hasegawa 1996). The likelihood value can be computed by using a recursive routine, which the expression is evaluated by working outwards from the innermost summation sign. The detail of the algorithm is described below.

3.3 Hardware Mapping for Pruning Algorithm

3.3.1 Related Works

Early work of Maximum-likelihood tree evaluation implementation can be traced back to the DNAml software, which is bundled in the phylogeny package Phylip (Felsenstein 1989). The software implements the "pruning" algorithm proposed by Felsenstein in (Felsenstein 1981). In the last decade, many software packages for ML phylogenetic tree reconstruction are released as introducing new search strategies or implemented in new programming languages. For examples, MORPHY proposed a complicate date structure of phylogeny that is dedicated for an effective branch length optimization (Adachi and Hasegawa 1996) and Phylogenetic Analysis Library (PAL) is package for phylogeny reconstruction written in Java (Drummond and Strimmer 2001). Besides, a commercial version phylogeny package PAUP is very popular among biologist (Swofford 2003) and fastDNAml is an improved version of DNAml (Olsen, Matsuda et al. 1994).

A few reasons that make the hardware implementation become difficult. Firstly, the maximum-likelihood computation involves large amount of floating-point multiplications and additions. Floating-point computation in hardware is expensive, in terms of speed and area, to many current digital technologies while most of the current custom digital hardware designs are in fixed-point due to its simplicity. Large truncation error will be introduced on the ML computation that makes the design is difficult to scale-up handle problem of larger dimension. For example, in (Ewe, Cheung et al. 2004), a dual fixed-point scheme provides a wider dynamic range number representation method was proposed. On the other hand, it is not flexible as software for developing hardware with dynamic data structure, of which it is widely available in software (Horowitz, Sahni et al. 1996). Effort of engineering reduction should be spent to simplify the procedure that the algorithm can be implemented in hardware efficiently. In this section, we present the methodologies on hardware design for the ML computation. Details about number representation, tree representation and traversal and the hardware-based recursive algorithm are presented.

3.3.2 Number Representation

Fixed-point architecture commonly used in FPGA digital logic design, provides the advantages of economic hardware resource consumption and high speed arithmetic. However,

the number range for a fixed-point architecture can represent is rather limited. Particularly, when we deal with probability numbers, which would be a small number in many case. Based on the idea of floating point, we issue a dynamic range of number representation with based on the existing fixed-point arithmetic. As the numbers are always between zero and one that generally a probability number F is represented by a pair (M, E) having the value

$$F = M \cdot 2^{E} \tag{3.6}$$

where M is the significant (or mantissa), E is the exponent. In our design, the base is 2 as it is convenient in digital logics design. In addition, E is always a non-positive integer and M is between 0.5 and 1. This is because F is always between zero and one in our case, and M is assumed to be normalized (shifting the leading 1 to the leftmost bit). Then we can use two fixed-point numbers to represent a simplified floating-point representation in hardware as assigning a m-bit string with binary point at zero position to represent M and a p-bit string to represent E as the first sign bit can be ignored as we have assumed the exponent term is always negative.

It is flexible in reconfigurable computing design that the number of bits assigned to represent a number or the arithmetic operations. Suppose, m bits are used to represent the mantissa M and p bits used to represent the exponent term E. Dynamic range is defined by the ratio between the largest and the smallest absolute number in the data format. The smallest absolute value of the representation on Eq. (3.6) is $2^{-2^{p-1}+1}$ while the largest absolute value is $(1\text{-}2^{-m})$, hence the dynamic range of this number representation is given as

$$\text{Dynamic range} = 20\log_{10}(2^{2^{p}+1}(1-2^{-m}))\ \text{dB} \tag{3.7}$$

Having two number of bits realization gives the proposed number representation is better range of capability than fixed-point as shown in Table 3.1.

Table 3.1 Dynamic range comparison between Simplified Floating Point (SFPT), Fixed-point and Single Precision Floating Point (FPT)

Number representation	SFPT	SFPT	Fixed-point	Floating Point
Format[3]	32_8	32_16	32-bit	32-bit IEEE
Dynamic Range	767dB	394kdB	187dB	1529dB

[3] For the format x_y, where x is the number of bits assigning to mantissa and y is the number of bits assigning to the exponent.

The proposed number representation can handle a much larger number range when comparing to fixed-point number representation. This is importation in the maximum-likelihood computation, as there are many multiplication operations on the probability values, which is a number less than one (See Eq. (3.4)). Large truncation is expected on the fixed-point number, that the system requires large number of bit assigning to the binary representation, which is hardware costly. Noted that the simplified floating-point representation with 16 exponent bits has a larger number range than the single precision floating point representation. In other words, this representation has potential for handling very large scale phylogeny maximum-likelihood computation.

Modification on the fixed-point arithmetic operators can handle the SFPT arithmetic. In addition, only additions and multiplications are required in the system for the probability calculation. The design of the SFPT arithmetic operators is referring to the appendix.

3.3.3 Binary Tree Representation

Binary tree is one of the most fundamental and important data structure in computer science, that it also has been investigated intensively in the field of computer science for a very long time (Horowitz, Sahni et al. 1996). Binary tree is a finite set of nodes that is either empty or consists of a root and two disjoint binary trees called the left subtree and the right subtree. A lot of tree operations based on the binary tree are well studied.

Binary tree is similar to a phylogenetic tree, excepting that the *root* node in the phylogenetic is unknown. In most of the case, the tree is unrooted. To represents a phylogeny using a binary tree, we can arbitrary assign an imaginary root to the phylogeny, that root is the parent of two neighboring internal node. It can be done by split the tree by removing any one edge between two nodes. Then add another node as root, and which is also the parent of the two nodes (See Figure 3.3). This can convert a phylogeny to a binary tree. On other hand, the binary tree can be easily converted back to the phylogeny by removing the root.

An unrooted tree can be converted into a pseudo-rooted binary tree with one additional root node. An unrooted bifurcating phylogenetic tree has n leaf nodes (or taxa) and n-2 internal nodes. There are a total of $2n$-2 nodes with $2n$-3 branches. The pseudo-rooted binary tree then has a total of $2n$-1 nodes (one root, n terminals, and n-2 non-terminals). FPGA implementation should use either a ROM (or RAM) or fast registers. A binary tree node should have five address attributes: node index or address (N), parent address (P), left child address (L), right child

address (R). Information of a node can be accessed through the node index, e.g., branch_length(N). A $2n$×24-bit RAM/ROM table is used for the n-taxa rooted or unrooted tree, each row has 3 column fields each with 8 bits, storing P, L, and R, respectively. Row 0 (or node index 0) should not be used for valid node indexing. The unsigned 8-bit field then implies that a maximum node index of 2^8=196 can be referenced. The depth of the RAM/RAM should be $2n$ for an n-taxa tree. For a general number of taxa, the size of the ROM/RAM is $6n \log_2 n$.

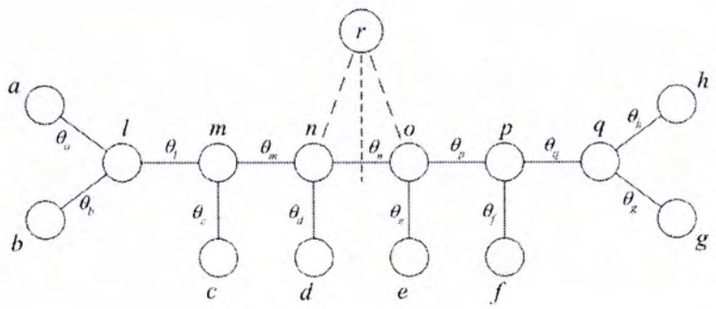


Figure 3.3 A 8-node unrooted tree, which is used to illustrate the pseudo-binary tree data structure. The root r is an arbitrary root adding between the node n and o. The edge between n and o is replaced by the edge between r and n. The edge between r and o is an imaginary branch.

Node	Address	P	L	R
Null	0	0	0	0
a	1	9	0	0
b	2	9	0	0
c	3	10	0	0
d	4	11	0	0
e	5	12	0	0
f	6	13	0	0
g	7	14	0	0
h	8	14	0	0
l	9	10	1	2
m	10	11	9	3
n	11	15	10	4
o	12	15	5	3
p	13	12	6	14
q	14	13	8	7
r	15	0	11	12

Figure 3.4 An example of tree represented in a ROM/RAM that the tree is in Figure 3.3

3.3.4 Binary Tree Traversal

The tree traversal is required to determine the correct operation sequence in evaluating tree T. Pre-order traversal generates a Polish expression from the tree, while a post-order traversal generates a reverse-Polish expression. Pre-order traversal is defined recursively, starting from the root r of T with its left subtree T_1 and right subtree T_2, as follows:

$$\text{Pre-order}(T) = r, \text{Pre-order}(T_1), \text{Pre-order}(T_2) \tag{3.8}$$

Example: 8-taxa unrooted phylogenetic tree. The given data structure of a RAM/ROM table for the example tree is assumed. Pre-order tree traversal will generate the node sequence [15, 11, 10, 9, 1, 2, 3, 4, 12, 5, 13, 6, 14, 8, 7].

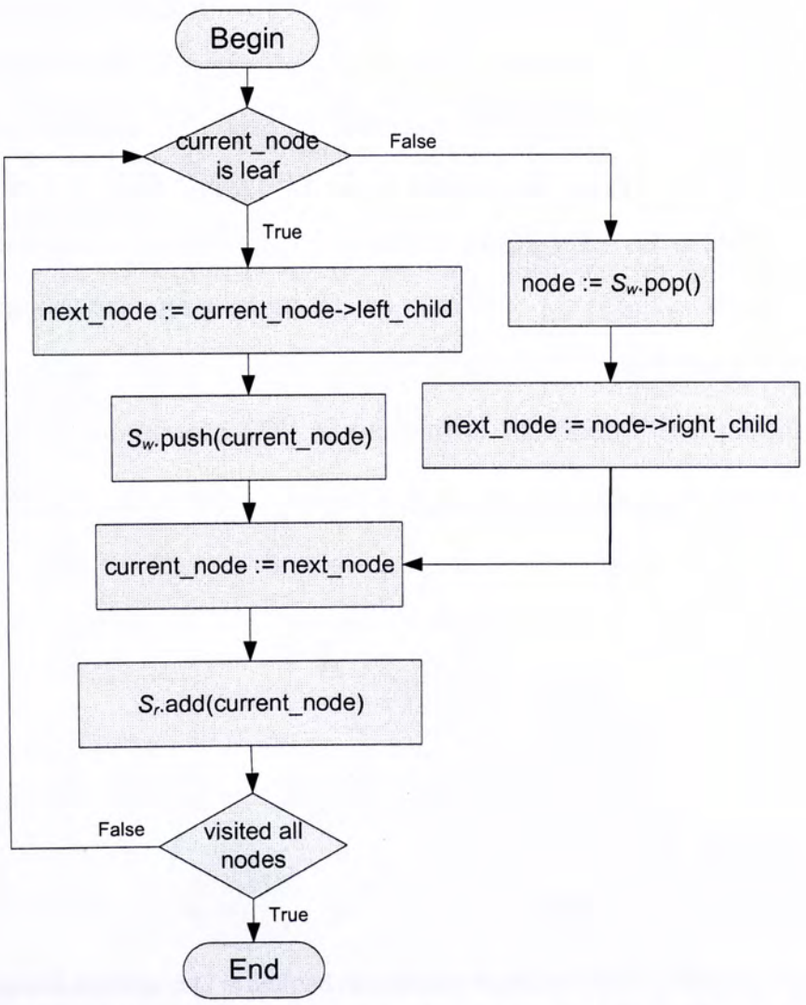


Figure 3.5 The flowchart for pre-order tree traversal with using stack

Recursive algorithm is not straightforward in hardware implementation while it can be implemented using software in a straightforward way. The Pseudo code for finding pre-order sequence for a binary tree T with root r is presented in a flowchart in Figure 3.5, which can be implemented in hardware. The code is developed using operation on two stacks, *working stack* (S_w) and *result stack* (S_r). The working stack supports the recursion while the results stack stores the polish expression.

3.3.5 Maximum-Likelihood Evaluation Algorithm

Likelihood probability evaluation takes exponential time in the number of unknown nodes, as the number of terms in Eq. (3.4) increases exponentially in the unknown variables. However, there exist economics way for computing these probabilities based on variable elimination (Horner's rule) that rearrange the individual terms in Eq. (3.5). One of the realization can be referring to the "pruning" algorithm proposed by Felsenstein. It is based on a reverse-polish expression to evaluate the likelihood of a given tree. In order to evaluate the likelihood value of a tree based the Pseudo-Binary tree data structure, the likelihood function can be formulated as a recursive function. In line with the works of (Adachi and Hasegawa 1996), *partial likelihood* is defined as the probabilistic likelihood of a node conditioned on a state. We can express the partial likelihood values for site pattern D_s as a matrix $Q=[Q_{ik}]_{c\times d}$, where c is the number of states and d is the nodes index. Following the definition of a binary tree, which is defined recursively as Eq. (3.8), we denote $Q_i^{(P)}$ as the partial likelihood of a parent node and $Q_i^{(L)}$ and $Q_i^{(R)}$ as the partial likelihood of left and right child nodes respectively (See Figure 3.6).

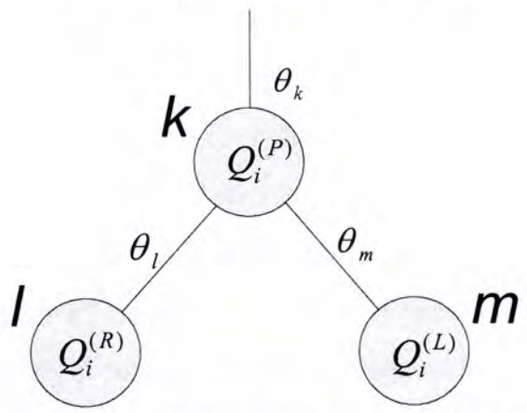


Figure 3.6 The partial likelihood Q is defined recursively based its parent-children relationships, where the superscripts (P) is the parent, (R) is the right child and (L) is the left child. The subscript of Q is the conditional state. θ_i is the branch length with respect to node i.

The partial likelihood can be computed recursively based on the reverse-polish expression (Eq. (3.9)-(3.12)). The recursion starts from one of the leaf node, where the partial likelihoods for this node are computed by using Eq. (3.12). As this is a leaf node, the partial likelihood simply equals to the corresponding transition probability of the DNA state of the node (i.e. D_s of the node). Partial likelihoods of any internal nodes are computed based on Eq. (3.11). Note that the formulation of Eq. (3.9) is in a similar form of Eq. (3.11), in which defines a binary tree. Based on the polish expression given by the Eq. (3.8), the $Q_i^{(L)}$ and $Q_i^{(R)}$ has been computed based on previous visit. As we are using a Pseudo-binary tree and one of the branch is imaginary branch that the partial likelihood value in corresponding to this node should be using Eq. (3.11) to compute. Finally, when visiting the root node, the overall likelihood probability is evaluated based on Eq. (3.9).

$$P(D_s \mid T, M) = Q^{(Root)} = \sum_{i=1}^{c} \pi_i \cdot Q_i^{(L)} \cdot Q_i^{(R)} \tag{3.9}$$

$$Q_i^{(P)} = Q_i^{(L)} \cdot Q_i^{(R)}, \qquad \text{if } P \text{ is the root node} \tag{3.10}$$

$$Q_i^{(P)} = \sum_{j=1}^{c} P_{ij}(\theta_k) \cdot Q_j^{(L)} \cdot Q_j^{(R)}, \qquad \text{if } P \text{ is the internal node} \tag{3.11}$$

$$Q_i^{(P)} = P_{ix}(\theta_k), \qquad \text{if } P \text{ is the leaf node} \tag{3.12}$$

An algorithm is introduced to implement the tree evaluation with input the reverse-polish expression, where we introduce four kinds of operations that working on pseudo-binary tree data structure. They are given as: 1. Terminal Node Operation (TNO) for Eq. (3.12), 2. Internal Node Operation (INO) for Eq. (3.11), 3. Imaginary Branch Node Operation (IMBO) for Eq. (3.10) and 4. Root Node Operation (RTO) for Eq. (3.9). There are c working stack (S_i) are introduced to accommodate the recursive computation, where i is the index of the stack. As we are working on the DNA, there are 4 stacks are prepared and each stack is for each state.

The four kinds of operations becomes the basic function on the tree evaluation application. We propose a dedicate architecture that provides the four basic instructions (TNO, INO, IMNO and RNO). The processor is designed, so that the computation speed is optimized providing parallel execution. Also, block floating point number representation is adopted that a high accuracy can be obtained.

1. Terminal Node Operation (TNO), with node h

For each state i
  If($D_s = i$)
    Push ($P_{ii}(\theta_h)$) into S_i
  Else
    Push ($P_{ij}(\theta_h)$) into S_i
  end
End

2. Internal Node Operation (INO), internal node q with children nodes g and h

For each state i
  Q_left[i] = Pop S_i
  Q_right[i] = Pop S_i
  Q_product[i]= Q_left[i] × Q_right[i]

  Q_parent[i]=0
  For each state j
    If(i=j)
      Q_parent[i] += Q_product[i] × $P_{ii}(\theta_h)$
    Else
      Q_parent[i] += Q_product[i] × $P_{ij}(\theta_h)$
    endif
  endfor
End

3. Imaginary Branch Node Operation (IBNO), internal node o with children nodes l and p

For each state i
  Q_left[i] = Pop S_i
  Q_right[i] = Pop S_i
  Q_product[i]= Q_left[i] × Q_right[i]
  Push (Q_product[i])
End

4. Root Node Operation (RNO), root node r with children nodes n and o

Q_parent[0]=0
For each state i
  Q_left[i] = Pop S_i
  Q_right[i] = Pop S_i
  Q_product[i]= Q_left[i] × Q_right[i]
  Q_parent[i] += Q_product[i] × π_i
End

Figure 3.7 The pseudo-code of four basic routines that is the realization of Eq. 9-12 of four basic routines from Figure 3.7.

```
Function Comp_likelihood (r_polish, root, imaginary_node, N)
- root stored the root node address
- imaginary_node is the imaginary branch node address
- r_polish is the stack storing the reverse polish expression
- N is the total number of nodes

begin
    while(r_polish is not empty)
        n = pop(r_polish)
        If n > N
                If (n = root)
                        RNO
                Else if (n = imaginary_node)
                        IBNO
                Else
                        INO
                End
        Else
                TNO
        End
    End
end
```

Figure 3.8 The pseudo-code of the recursive maximum-likelihood evaluation algorithm based on the four basic routines from Figure 3.7

3.4 System Architecture

The data path diagram for maximum-likelihood tree evaluation is shown in figure 3.9. The transition probability unit evaluates the probability from the tree branch length and model parameters. The probabilities will be stored in RAMs that is to be retrieved by the state parallel computational unit. The tree traversal unit outputs the reverse polish expression of the pseudo binary tree. As the probability computation and the tree traversal are independent tasks, they can be executed in parallel with independent hardware. The state parallel computation unit implements the recursive maximum-likelihood evaluation algorithm with input the reverse polish expression and the transition probability and output the likelihood value. As likelihood value of sites are assumed independent, we can introduce an array of state-parallel computational units and each responsible for a sub-sequence of data. The overall likelihood is just summed over all computational units after taking the logarithm[4] and accumulated. The designs of these computational units are presents in the following sections.

[4] Multiplicative normalization approach is adopted on the logarithm implementation [23].

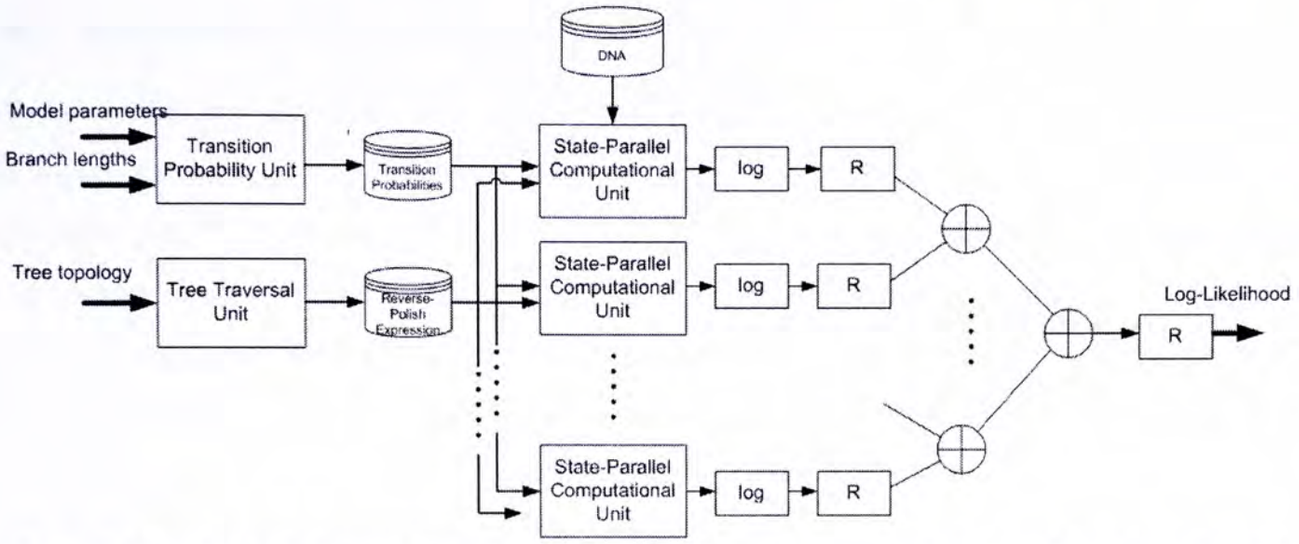


Figure 3.9 The data path diagram of likelihood evaluation for a given tree topology, branch lengths and model parameters. The state-parallel computational unit (SPCU) realize the recursive ML evaluation algorithm. Based on the site independent assumption, each SPCU evaluates a partial sequence and the overall likelihood can be obtained by adding up all the results from each SPCU.

3.4.1 Transition Probability Unit

The transition probability unit computes the state transition probability based on the input branch length and the model parameters. For simple models, such as Juke-Canter, the computation is realized using hardware. For instance, if we assumed that the sequences are evolving according to the Jukes Cantor model (Jukes and Cantor 1969). Following (Strimmer and Haeseler 2003), the probability can be computed based on two functions. We apply the addictive normalization unit to approximate the exponentiation value in the function (Ercegovac and Lang 2004.).

For more complicated modal, like HKY (Hasegawa-Kishino-Yano modal), spectral factorization is required (Hasegawa, Kishino et al. 1985). It is complicated and can be expensive in hardware implementation in terms of hardware resources. A hardware/software partitioning approach can be adopted to deal with the computation. The transition probability evaluation is then completed by the software (i.e. microprocessor). After obtaining the transition probability, FPGA read back the data from a commonly shared RAM. As this the transition probability values are the same for all nucleotide sites, this computation step only required in the initialization step. The probability values can be stored in the internal RAM of FPGA for the remaining computation.

3.4.2 State-Parallel Computation Unit

Consider the evaluation of a fixed-topology phylogeny as a recursive evaluation of the partial likelihood matrix Q, where the columns are nodes and rows are states. There exists data dependency on the columns, as the evaluation following a reverse-polish expression. But, entries with respect to each state are independent. In other words, the evaluation of each row of the matrix Q is independent. Therefore, parallelization can be applied on computing the partial likelihood matrix Q. Figure 3.10 shows the sequential and parallel ways to obtain the partial likelihood. The sequential and parallel implementations are shown on the left and right respectively. The calculation of partial likelihood in Figure 3.10 refers to the tree in Figure 3.1, which consists of 4 taxa. The nodes in the figure represent the process of computing the corresponding entry in matrix Q. Each of these nodes is evaluated by Eq. ((3.9)-(3.12)).

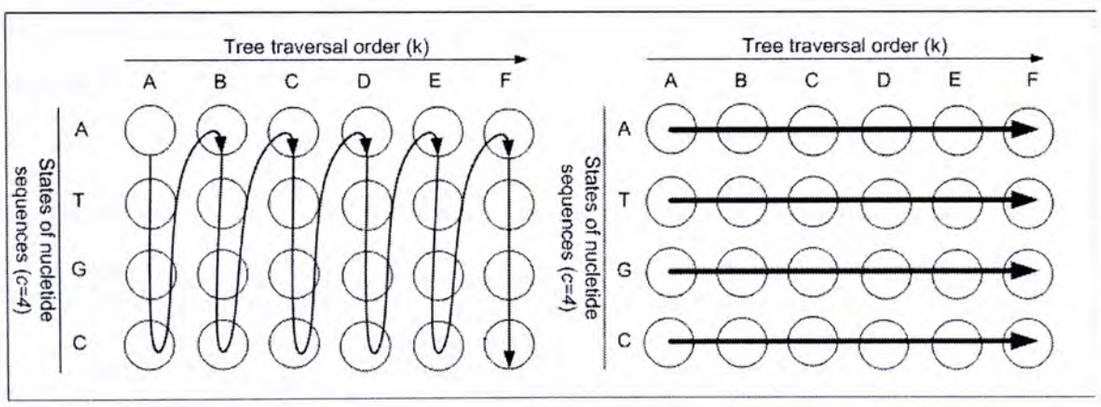


Figure 3.10 Comparison of the execution order for the partial likelihood matrix entries between software and hardware implementation. The left diagram is the software implementation that one entry of the partial likelihood matrix is computed each time. The right diagram is the hardware implementation that parallelism offers four entries are evaluating each time.

As shown in Figure 3.10, four independent processors can execute in synchronized to compute each entry of the partial likelihood. The realization is shown in Figure 3.11 that the computational unit is called State-Parallel Computational Unit. As its name implied, the partial likelihood of different states are evaluated in parallel. The basic building blocks of the unit are multiply-and-accumulation block (MAC), stack, multiplexer (MUX), registers (R) and the probability matching logics (P). The figure shows four identical state processors working in parallel and only the detail of the first one is shown for the sake of space saving. The

computational unit realizes the recursive maximum-likelihood computation algorithm with four different instructions are used repeatedly. There are n External Node Operation (ENO), n-3 Internal Node Operations (INO), one Imaginary Node Operation (IMNO) and one Root Node Operation (RTO).

Each processor is responsible for the partial likelihood of one state. The ENO requires only simple matching of the states that is realized by the probability matching logic. The comparator controls the selection of probability value based on whether the incoming DNA state is equivalence to the state index (S). Each processor has different S for distinguish the state they corresponding for. The Internal Node Operation (INO) is more time consuming as it requires addition and multiplications. The MAC unit is specially designed that not only outputs the multiply-and-accumulation results, but also output the product of two inputs. This design is dedicated for the evaluating of internal node partial likelihood and the imaginary branch node as which requires the only the product. The stack is for the recursive computation that four stacks are implemented for parallel data retrieval. The MUX block is the multiplexer, which is for the purpose of control. There are three registers in each processor to store the intermediate values.

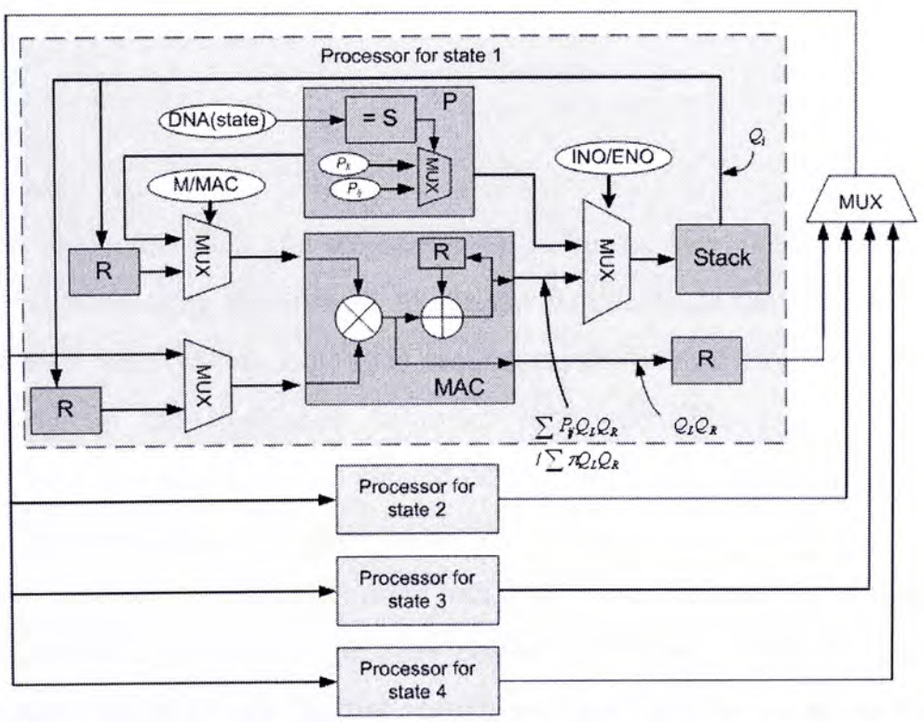


Figure 3.11 FPGA architecture of the State-Parallel Computational Unit (SPCU), which implements the recursive maximum-likelihood evaluation algorithm. There are four identical processors in the unit and each processor is responsible for computing the partial likelihood value conditioned on one state.

The likelihood probabilities evaluation requires $(2n-3)\cdot 4^{n-2}$ multiplications (See Eq. (3.4)). By applying the recursive routine with "pruning" algorithm, the number of multiplications are greatly reduced. There are 8 multiplications for evaluating each Q_i and there are 4 states, thus 32 multiplications in total for one internal node. Therefore, the total number of multiplications becomes to 32(n-3)+12. For the above evaluation of number of multiplications, single processor architecture (sequential realization) is assumed. Suppose an architecture, which is presented in Figure 3.11, with 4 parallel multipliers, the overall time is reduced and only 8(n-3)+3 is required. Table 3.2 shows that the number of multiplications required for case of 8 and 16 multipliers parallel implementation also.

Table 3.2 Timing analysis based on the number of multiplications

Number of parallel multipliers	Total time of multiplication for direct implementation of Eq. (3.4)	Total time of multiplications with implementation of Pruning algorithm	Number of RAMs required in the architecture
1	$(2n\text{-}3)\cdot 4^{n-2}$	$32(n\text{-}3)+12$	1
4	$(2n\text{-}3)\cdot 4^{n-1}$	$8(n\text{-}3)+3$	4
8	$(2n\text{-}3)\cdot 4^{n}$	$4(n\text{-}3)+2$	8
16	-	$2(n\text{-}3)+2$	16

Parallelism on sites

The likelihood values for each site are adding up to be the overall likelihood. Parallelization can be applied on computing the overall likelihood value. Since the sites are assumed to be independent[5] to each other in the nucleotide sequences, the overall likelihood value is obtained by simply adding up the likelihood value of each site (See Eq. (3.3)). In sequential implementation, only one site can be evaluated each time. In the parallel implementation, more than one site can be computed at the same time.

More than one *state-parallel computational unit*, which can be used to calculate the likelihood values of the sequences with any length (See Figure 3.9). The parallelism idea is to slice the input data into many equal length "partial sequence". Each "partial sequence" is computed by one computational unit, where they are running in parallel. The speed-up factor (i.e. the ratio of

[5] We are woking on homogeneous model that sites are assumed to be independent to each other.

sequential computation time to parallel computation time) equals to the number of computation units implemented. Certainly, the speed-up factor is limited by the availability of FPGA resources, where the resources consumption is also directly proportional the speed-up factor.

Pipelining

The partial likelihood evaluation and the logarithm computation (See. Figure 3.9) can be regarded as two independent tasks. There are registers for storing the results from the partial likelihood evaluation (first module) that will be used by the logarithm taking (second module). The two modules can be working independently. Therefore, pipelined can be applied on computing the likelihood and log-likelihood value of every site to achieve higher throughput.

3.4.3 Error Computation

The precision of the likelihood value computation obtained by the recursive maximum-likelihood algorithm can be calculated taking into account the two main error sources: 1.) the error in the transition probability representation that is due to the finite word length representation in digital logics and 2.) Truncation error accumulates at the multiplication of internal node operation and addition operations of those products with the limited bit length m of mantissa. The first source of error, which can be considered as a number representation problem, can be overcome by simply assigning more bits in mantissa. The maximum error on the transition probability number representation can be estimated as 2^{-m-2^p}, where m is the mantissa bit length and p is the exponent.

The second source of error correlates to the problem size that is more important to understand. The analysis is discussed as follows. Suppose real number A and B on the interval of zero and one are represented as $A' \cdot 2^{-r}$ and $B' \cdot 2^{-s}$ where r and s are non-negative integer and A' and B' are real number between 0.5 and one. Suppose there are m bits on representing the mantissa that A' and B' can be given as $\sum_{i=1}^{m} a_i 2^{-i}$ and $\sum_{j=1}^{m} b_j 2^{-j}$ where a_i and b_i is either zero or one. As A' and B' are number between zero and one, thus a_1 and b_1 must be one. The product of A and B can be given as:

$$
\begin{aligned}
A \times B &= A^{'} \cdot B^{'} \cdot 2^{-r-s} \\
&= \left(\sum_{i=1}^{m} a_i 2^{-i} \right) \cdot \left(\sum_{j=1}^{m} b_j 2^{-j} \right) \cdot 2^{-r-s} \\
&= \left[\left(\sum_{i=1}^{m-1} \sum_{j=1}^{m-i} a_i b_j 2^{-j-i} \right) + \left(\sum_{i=1}^{m} \sum_{j=m-i+1}^{m} a_i b_j 2^{-j-i} \right) \right] \cdot 2^{-r-s} \\
&= \left(| A^{'} \cdot B^{'} |_{trun}^{m} + \varepsilon_{\otimes} \right) \cdot 2^{-r-s}
\end{aligned}
$$

where the $||_{trun}^{m}$ is the number can be represented by the mantissa and $\varepsilon_{\otimes}$ is the truncation error due to the limited bit range. Suppose all the truncated bits are one, the maximum error for $\varepsilon_{\otimes}$ is given as

$$
\begin{aligned}
\varepsilon_{\otimes} &\leq \sum_{i=1}^{m} i \cdot 2^{-2m+i-1} \\
&= 2^{-m} (m + 2^{-m} - 1)
\end{aligned}
$$

Since there are $2n$-3 multiplications for each internal node, the overall error attributes to the mantissa during multiplication can be estimated as

$$
\begin{aligned}
error_{\otimes} &= \varepsilon_{\otimes} \left(1 - \sum_{i=1}^{2n-5} \prod_{j=1}^{i} \mathrm{P} \right) \\
&\leq \varepsilon_{\otimes}
\end{aligned}
$$

where P is general term referring to the transition probability values.

Besides the multiplications, additions operation on these products are considered. For each addition operation of A and B, that is represented as $A^{'} \cdot 2^{-s}$ and $B^{'} \cdot 2^{-t}$ the error can be estimated as follows with assuming $|s| \leq |t|$:

$$
\begin{aligned}
A + B &= (A^{'} + \varepsilon_{\otimes}) \cdot 2^{-s} + (B^{'} + \varepsilon_{\otimes}) \cdot 2^{-t} \\
&= 2^{-s} \cdot (A' + \varepsilon_{\otimes} + B' \cdot 2^{-t+s} + \varepsilon_{\otimes} \cdot 2^{-t+s}) \\
&= 2^{-s} \cdot (\left| A' + B' \cdot 2^{-t+s} \right|_{trun}^{m} + \varepsilon_{\oplus} + \varepsilon_{\otimes} \cdot (1 + 2^{-t+s}))
\end{aligned}
$$

Suppose $s = t$, so that the worst case of the error on the addition operation of two terms can be known as $\varepsilon_{\oplus} + 2\varepsilon_{\otimes}$. There are $4n$-8 additional operations, the overall error can be known as

$$\begin{aligned} \varepsilon_{likelihood} &= (4n-9)\varepsilon_{\oplus} + (4n-8)\varepsilon_{\otimes} \\ &\leq (4n-9)2^{-m} + (4n-8)\cdot(m+2^{-m}-1)\cdot 2^{-m} \end{aligned} \tag{3.13}$$

The number of significant decimal digits for the likelihood computation in this system can be estimates using the results in Eq. (3.13) as follows:

$$\text{significant digits} \approx \left| \log_2\left((4n-9)2^{-m} + (4n-8)\cdot(m+2^{-m}-1)\cdot 2^{-m}\right)\right| / 3 \tag{3.14}$$

Suppose we assign 32 bits for the mantissa representation, where m=32, and there are 8 taxa, where n = 8. Then we have the error bound equals 1.7858e-7 and the significant number of valid digits is estimated as 7.47 ≈ 7 digits are valid. For using 32-bit mantissa assignment, we can still have valid significant digits larger than 6 for 128 taxa.

Note that only error of computing likelihood for one site pattern is estimated. The probability likelihood is taking the logarithm and accumulated with number of sites. Suppose there is number loss of precision on logarithm and we assume that the precision error accumulates over site likelihood accumulation, the over error can be the accumulated with the number of sites, in which only four decimal significant digit can be found for one thousand sites. The precision is greatly reduced in our number system. In order to reduce the precision loss in this step, we suggest adopting the fix-point number system with a very long word length, such as 48-bit or 64-bit, at this step. Once the site likelihood taking the logarithm, the number is naturally converted the result into a fixed-point number representation. The number can be simply represented I fixed-point without extra effort. In contrast, extra logics are required to convert the number into floating point representation. Also, only addition operations are applied on this fixed-point number, the precision loss can be reduced to 2^{-q} simply increasing the word length to q, where q is number much larger than m, says 64.

3.5 Discussion

3.5.1 Hardware Resource Consumption

There are three different kinds of hardware resources, which are 1.) Logic slices, 2.) Internal Block RAM (BRAM) and 3.) Embedded multipliers, available within current FPGA[6] technology.

[6] We refer current FPGA as Xilinx Virtex-II FPGA

These are called on-chip hardware resources that are particularly more efficient in data communication and operation when comparing with the off-chip hardware. As different FPGA technology offers different combination of number of embedded multipliers, BRAM and slices. To our design, the overall number of memory consumption is correlated to the problem size (i.e. number of taxa), as the maximum depth of the working stack is n-1. Also, the DNA sequences are stored in internal RAM. The number of RAM can be estimated as equals to $0.125n+19$ that is linearly proportional to the problem size. (See Table 3.3, the number RAM required is linearly proportional to the number of taxa) Only 4 multipliers are required, as there are 4 MAC units in one computational unit that is independent to the problem size. (See Table 4, the number of multipliers are always constant) The adder, multiplexer, comparators, registers and controls are consuming logics in the FPGA. The logic consumption can be regarded as constant in our hardware model, as increasing the problem size will introduce little extra logic to the recursive algorithm. (See Table 3. The logic increase 0.35 percent in average when the number of taxa is double)

Table 3.3 Hardware resource consumption with increasing number of taxa

Number of taxa	RAM	Logic	Multiplier
8	20	4512	20
16	21	4535	20
32	23	4542	20
64	27	4561	20
128	35	4576	20

3.5.2 Evaluation Delay

The FPGA design computational time can be estimated based on the product of total number of clock cycles and the minimum clocking period. The total number of clock cycles depends on the design that more parallelism applied fewer clock cycles required. In our proposed design, the total estimated computational time is the sum of the transition probability computation (2n-2), recursive ML evaluation ($33n$-27). The recursive ML evaluation is repeated l+1 times for the l nucleotide sites and the extra cycle is for completion the logarithm as which is computed in pipeline. On the other hand, the DNA sequences can be equally partitioned into K subsequences and distributed to the K independent SPCUs to provide speed-up of K times. The FPGA-based

system evaluation ML evaluation time is given as:

$$\text{Evaluation Time (FPGA)} = [107n\text{-}101+l(33n\text{-}27)] \times 1/\text{frequency} \times 1/K \qquad (3.15)$$

Comparison of computational time between the software and FPGA implementation is presented in Figure 3.12. The time is measured by varying the number of taxa on a fixed length (l=500) sequence. Hence, we can find out the relationships between the computation time and the problem size (n). The X-axis is the dimension of the number of taxa while the Y-axis is the time measured in seconds. The time scale on the left is for the FPGA-based implementation while the time scale on the right is for the software implementation. The line is the result from the software while the bars are the FPGA results. There are two different cases for FPGA are presented which are differs from the number of computation units used. FPGA with single computation unit offers around 20 times speed-up when compared to the software computation. For higher degree of parallelism, five computation units are implemented in the FPGA design and 180 times acceleration can be achieved. In average, 21.9 and 175.6 speed-up can be found for using one and eight SPCUs for evaluating the ML phylogeny respectively.

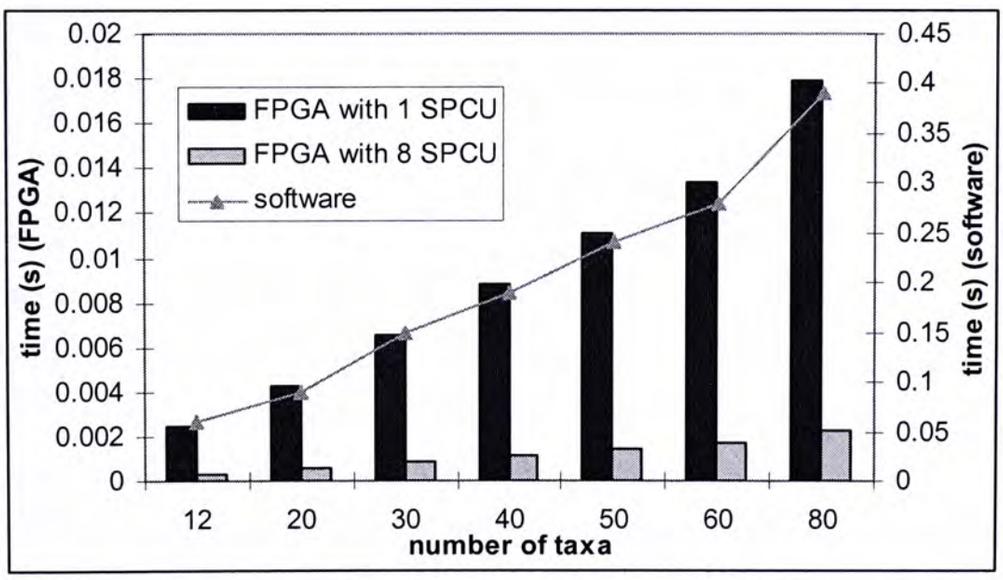


Figure 3.12 Comparison of the computation time between software and FPGA implementation. The FPGA implementation result is referring to the left time scale while the software results are referring to the right time scale. Two cases, within one FPGA processor one and eight SPCU are implemented, are shown for the FPGA implementation.

The FPGA-based system provide significant speed-up on the maximum-likelihood phylogeny evaluation. It can be regarded as a coprocessor in accelerate the phylogenetic tree evaluation while the software (microprocessor) is searching the tree topology with heuristics. However, there is communication cost between the software and hardware, which might decrease the speed-up ratio that the hardware provided. The latency in communication is technology dependent. In (Mak and Lam 2003), we have shown that using HW/SW (Software/Hardware) codesign for GAML implementation can provide significant speed up when compared with software-only implementation. The HW/SW approach is benefited from high performance hardware and the flexibility of software. In our preliminary design, the GAML is divided into two parts: the genetic algorithm (excluding the fitness evaluation) and the ML tree topology evaluation. Since the tree evaluation process is repeated extensively, which has been speed-up by using FPGA, the overall GAML runtime is reduced tremendously. It is also shown that our HW/SW system acquires a high potential for scaling up the problem domain for real applications.

In (Mak and Lam 2004), the HW/SW co-design system been extended to a more powerful embedded computing platform. In this platform, a microprocessor is immersed into FPGA fabric for realizing an effective environment for HW/SW co-design implementation. Significant improvements in data transmission between hardware and software and higher clock frequency of FPGA have been realized when compared to the interface in (Mak and Lam 2003).

3.6 Conclusion

This was the first attempt to investigate the problem of dedicated hardware realization for maximum-likelihood DNA phylogenetic tree evaluation. Hardware architecture for the evaluation algorithm has been proposed. FPGAs realization can provide significant acceleration, from 20x to 180x times, when comparing to software implementation. The hardware acceleration for the maximum-likelihood computation is attributed to the fine-grained parallelism based on the DNA state and nucleotide site independency, and the parallel recursive scheme proposed in this chapter. We also found that the logic utilization increases only 0.35 percent in average when the number of taxa is double. The number of multipliers used is always a constant. It implies that the design can be applicable to handle problem of larger scale. However, the internal memory requirement is linearly proportional to the problem scale. Therefore, FPGAs with larger internal memory is needed for handling large scale phylogeny analysis. In addition, we developed a

simplified floating-point number representation scheme, which is well adaptable for the precision demanding probabilistic computation. We also derived a formula on the error computation dedicating for the maximum-likelihood evaluation. The relationship between the number of significant decimal digits and the bit length assignment has been shown.

Chapter 4

Field Programmable Gate Array Implementation of Neuronal Ion Channel Dynamics

Neuron-machine interfaces such as dynamic clamp and brain-implantable neuro-prosthetic devices require real-time simulations of neuronal ion channel dynamics. Field programmable gate array (FPGA) has emerged as a high-speed digital platform for application-specific computation devices. Here, we present an FPGA design of a neuromorphic Hebbian synapse which mimics the NMDA and non-NMDA ion channel dynamics observed experimentally in hippocampal neurons. The proposed design can be readily extended to high-speed implementations of dynamic clamp and neuro-prosthetics as replacements for damaged neurons in the brain.

Keywords – Ion channel dynamics; dynamic clamp; brain-machine interface; brain-implantable neuro-prosthesis; neuromorphic synapse; Field Programmable Gate Array

4.1. Introduction

Real-time simulation of neuronal ion channel dynamics is an important step in the implementation of neuron-machine interaction, which is fundamental to several emerging neuromorphic and biomimetic applications. For example, in electrophysiological studies of neuronal membrane properties using the dynamic clamp technique (Sharp 1993; Butera 2004) , a digital computer is used to generate virtual ion channel conductances which continuously interact with a biological neuron in real time. Such software-based experimental applications are highly computation-intensive and often require judicious choice of operating system (Sharp 1993) and numerical procedures (Butera 2004) to improve the computational speed and flexibility. A hardware-based, application-specific implementation of the dynamic clamp technique would circumvent the limitations of general-purpose computers.

Another example of real-time neuronal ion channel dynamics computation is found in neuro-prosthetic devices using brain-machine interface (BMI). For example, a robotic arm controlled by central brain activity has been shown to be capable of generating complex motions (Taylor 2002), and such capability may find important applications in patients with Parkinson's disease, Essential Tremor, and dystonia (Isaacs 2000). Future applications of such neuro-prosthetic devices might incorporate brain-implantable biomimetic electronics as chronic replacements for damaged neurons in central regions of the brain (Berger 2001). One such technology is neuromorphic analog VLSI circuits (Mead 1990). Towards this end, we have previously proposed a neuromorphic Hebbian synapse design using analog CMOS circuits operating in subthreshold regime (Rachmuth and Poon 2003; Rachmuth 2004). However, the relatively long design and fabrication cycles for analog CMOS circuits can be a bottleneck for the development of such devices.

In recent years, Field Programmable Gate Array (FPGAs) technology has emerged as a high-speed digital computation platform. The flexibility of the FPGA's programmable logic combined with its high-speed operation potentially allows it to control neuro-prosthetic devices and dynamic clamp systems in real time. Additionally, FPGAs can be used to accelerate prototyping of analog hardware models of brain processes by quickly building a simulation platform to study the functional behavior of the proposed model in a much shorter design cycle.

In this chapter, we show that FPGA is an effective prototyping or permanent platform for hardware modeling and simulation of neuronal ion channel dynamics. Specifically, FPGAs offer an advantage of high-speed signal processing which could be orders of magnitude faster than software-based approaches to interfacing biological neuronal signals.

4.2 Background

4.2.1 Analog VLSI Model for a Hebbian Synapse

Previously, we described a biologically isomorphic (or 'neuromorphic' (Mead 1990)) implementation of various biophysical models of synaptic adaptation, using MOS transistor circuits designed in the analog (subthreshold) regime (Rachmuth and Poon 2003; Rachmuth 2004). The system design is based on a biophysical model of Hebbian synapse in area CA1 area of the hippocampus (Kitajima and Hara 1990; Zador, Koch et al. 1990).

The model features two types of ionotropic glutamate receptors: N-methyl-D-aspartate (NMDA) and non-NMDA (AMPA and kainate) receptors (See Figure 4.1). The current flow through an ion channel can be thought of as an ohmic relationship with a time varying conductance level, which is modeled as an alpha function. Non-NMDA channels, which carry the majority of the excitatory synaptic current, are modeled as:

$$I_{non-NMDA}(t) = g_{syn}(t)(V_m(t) - E_{syn}) \tag{4.1}$$

$$g_{syn}(t) = const \cdot te^{-t/t_{peak}} \tag{4.2}$$

where g_{syn} is the ligand-dependent conductance of the channel, V_{m} is the membrane potential, and E_{SYN} is the reversal potential of the synapse.

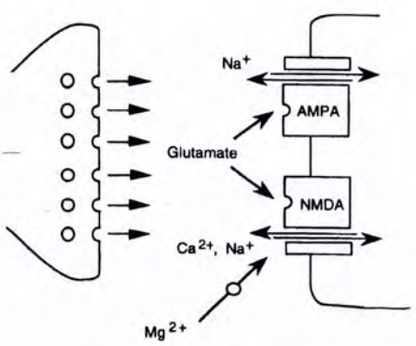


Figure 4.1 Biological synapse with glutamate activating AMPA and NMDA channels, and generating an EPSP.

NMDA channels are the major source of Ca^{+2} influx into the postsynaptic cell. They have the interesting property that their gating is jointly controlled by neurotransmitter binding and by a voltage-dependent blockage of the channel by Mg^{+2} ions. This bi-variate gating function is modeled by the following equation:

$$I(t)_{NMDA} = (E_{syn} - V_m(t))g_n \frac{e^{-t/\tau_1} - e^{-t/\tau_2}}{1+\eta[Mg]e^{-\gamma V_m(t)}} \tag{4.3}$$

where g_n is the maximal channel conductance, τ_1 and τ_2 are the channel's activation and deactivation time constants, [Mg] is the extracellular magnesium concentration, η and γ are constants, E_{syn} is the membrane reversal potential, and V_m is the synaptic membrane potential.

Because of their unique dependence on both presynaptic and postsynaptic activation, NMDA channels are generally thought to be a biophysical implementation of the Hebbian adaptation rule (Hebb 1949).

4.2.2 A Unifying Model of Bi-directional Synaptic Plasticity

Hippocampal long-term potentiation (LTP) and depression (LTD) can be induced experimentally by varying: (1) the membrane potential of the postsynaptic neuron during presynaptic stimulation at a constant frequency (Crair and Malenka 1995); (2) the rate of presynaptic stimulation (Bliss and Lomo 1973; Dudek and Bear 1992; Mulkey and Malenka 1992); and (3) the relative timing of presynaptic action potentials and postsynaptic backpropagating action potentials while holding stimulation rate constant (Bi and Poo 2001). Recently, a model proposed by Shouval et. al. (Shouval, Bear et al. 2002) has attempted to unite the various induction protocols of synaptic plasticity into a single Hebbian mechanism that is controlled mainly by intracellular calcium dynamics. This biophysical learning rule is represented by the following equation:

$$\frac{dW}{dt} = \eta([Ca])(\Omega([Ca]) - W) \tag{4.4}$$

where W represents the synaptic strength, η is a calcium-dependent learning rate, and $[Ca^{+2}]$ denotes the calcium level in the postsynaptic neuron. The function Ω, which represents the Bienenstock, Cooper, Munroe (BCM) sliding threshold adaptation model (Bienenstock, Cooper et al. 1982), defines the sign and magnitude of synaptic plasticity.

4.2.3 Non-NMDA Receptor Channel Regulation

Biologically, synaptic strength is often measured as the amplitude of excitatory postsynaptic current (EPSC) that is elicited by a presynaptic spike. Synaptic current is mediated predominantly by multiple non-NMDA receptor channels acting in parallel, each opening and closing randomly in an all-or-none fashion. The total synaptic current is determined by the average number of channels that are active at any instant. Hence, synaptic weight changes can be thought of as an average increase or decrease in the number of non-NMDA receptor channels activated for a given presynaptic stimulus.

3. FPGAs Implementation

4.3.1. FPGAs Design Flow

Our logic-level design makes extensive use of Xilinx's System Generator (SG) that works under MATLAB's Simulink environment (J. Hwang 2001). Simulink provides a schematic design environment of logic gate blocks with which to implement the FPGA model. SG then automatically converts the simulink code from the schematic design to a bit stream file to configure the FPGA hardware.

FPGA as a standalone simulation device is properly interfaced with different kind software, such as Matlab and LabView. Analog signal can be converted to digital by using an external Analog-to-Digital conversion (ADC) chipset before inputting to FPGA.

4.3.2 Digital Model of NMDA and AMPA Receptors

The NMDA and non-NMDA ion channels dynamics can be modeled by using FPGA that makes use of digital logic blocks to evaluate the characteristic equations of receptors, and a differences equation for the membrane voltage dynamics. The dynamic membrane potential can described as a differences equation as follows

$$V_{MEM}(t+1) = V_{MEM}(t) - \frac{g_{leak}}{C_{MEM}} \cdot (V_{MEM}(t) - V_{rest}) + \frac{\sum I_{EXE}(t)}{C_{MEM}} \quad (5) \qquad (4.5)$$

where the C_{MEM} is the membrane capacitance, g_{leak} is the leak conductance of the membrane patch, and I_{EXE} is the excitatory current sources, which are the I_{NMDA} and $I_{non\text{-}NMDA}$ described in Eq. (4.1) and Eq. (4.3).

In the biological system, the post-synaptic activities are initiated by a presynaptic pulse. However, in the functional description of the NMDA and non-NMDA ion channel dynamics, the input is implicit. The function takes two inputs, the membrane voltage V_m and time t. In our design, time is emulated by using a counter, which is reset to zero at each presynaptic pulse, and then outputs integers from zero to a large integer as a function of internal clock frequency. The other input of the receptors function is the membrane voltage V_m. It is initially set to the resting membrane potential, V_{REST}.

Following (Rachmuth 2004), an input pulse triggers NMDA and non-NMDA currents. The output currents are integrated on C_{MEM}, which depolarizes membrane potential. The resulting V_m value will be used as an input for evaluating the NMDA and non-NMDA channel currents in the next time step.

In our proposed design, we are using a register to store the membrane voltage and the value in the register is updated with a feedback loop from the output of the NMDA and non-NMDA functional block. Also the leak conductance of the membrane can be regulated by the differences between the membrane potential and the rest potential.

Figure 4.2 shows the schematic of an FPGA design of a synaptic compartment containing NMDA and non-NMDA receptors. The implementation also has a control block to enable the counter, and a register R to store V_m. R is initially set to V_{REST}, and is then updated accordingly throughout the iteration loping. The NMDA and non-NMDA blocks implement Eq. ((4.1)-(4.3)) in parallel and generate output currents with respect to the input time and membrane voltage. The accumulator converts the charge into a voltage, acting as an RC circuit modeling the effects of C_{MEM} and g_{leak}.

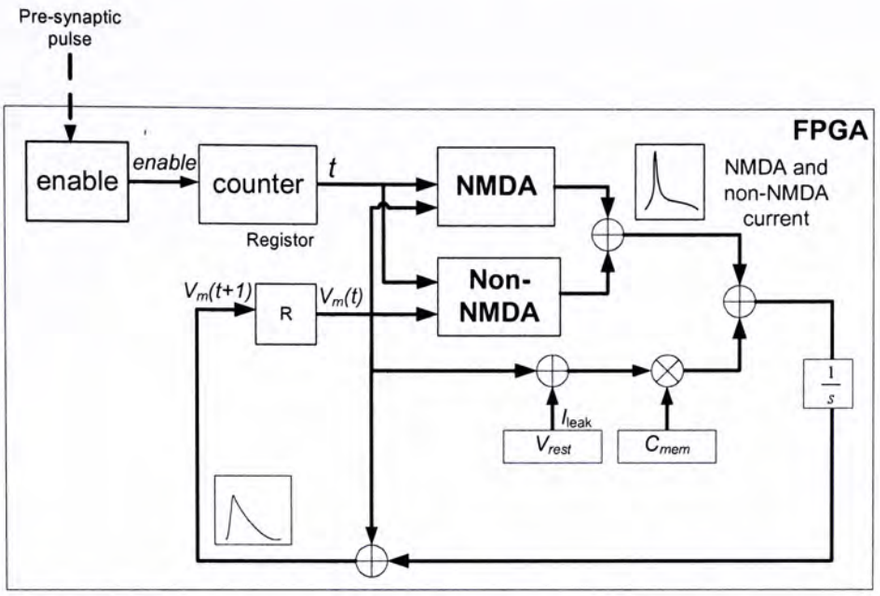


Figure 4.2 Schematic design of the NMDA and non-NMDA synapse using FPGA.

The implementation of the NMDA and non-NMDA channel function using digital logic is nontrivial because of the presence of exponential functions and divisions in the equations. We studies several possible methods to implement these function evaluations. We chose to implement the functions using direct table look-up for simplicity. The basic NMDA and nonNMDA values are pre-computed and stored in the memory. The two inputs, time and V_m, are mapped to the RAM address, and looked-up as needed.

Future implementations that need to minimize RAM consumption require alternative implementations. For example, the Coordinate Rotation Digital Computer (CORDIC) algorithm provides an effective solution for basic function evaluations, such as logarithms and division. It evaluates the basic function with using one adder, shifters, and a small look-up-table (Andraka 1998). For greater precision, the gradient of the function can be stored for interpolation.

4.3.3. Synapse modification

Following (Kitajima and Hara 1990), we considered synaptic weight changes as an average increase or decrease in the number of active non-NMDA receptor channels. The digital implementation for this model is shown is Figure 4.3. The digital representation uses n ON/OFF gated channels. A n-bit synaptic weight vector W controlled their gating, with "1" representing the ON state and "0" representing the OFF state. Thus, the sum of all n channels bits was the amplifying factor of the non-NMDA current.

The synaptic weight is controlled by the calcium current which generated by the NMDA block. I_{Ca}^{+2} is accumulated as V_{Ca}^{+2} which is then used to drive synaptic plasticity learning rule described by Eq. (4.4). The updated weight is multiplied with a unitary non-NMDA current to magnify the non-NMDA current according to the learning rule.

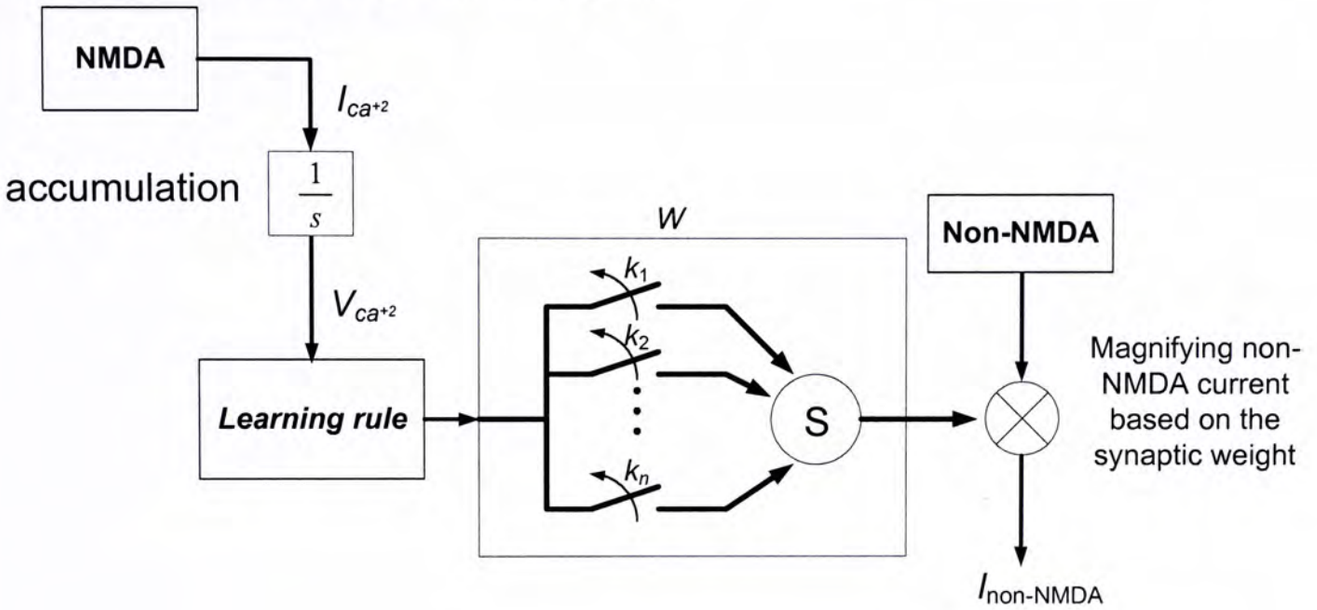


Figure 4.3 Schematic design of the NMDA and non-NMDA synapse using FPGA.

4.4 Results

4.4.1. Simulation Results

Figure 4.4 shows the comparison between biological recordings and FPGA simulation of individual NMDA channels, and miniature EPSCs, which sums AMPA and NMDA currents. The behavior of NMDA and AMPA current from the FPGA simulation are shown to be qualitatively similar to the experimental data in (Renger, Egles et al. 2001) in a.). Ion channels and the resulting membrane potential dynamics modeling using FPGA leads to a real time response to the input spikes and the membrane voltage modification.

The FPGA is driven by a global clock signal. Therefore, the computational speed of the model is dependent on the frequency of the digital clock f and the complexity of the model equations. The amount of time it takes to evaluate our model equations can be expressed as $p \cdot 1/f$, where p the number of clock cycles needed to compute an answer. In our design, the maximum

frequency is 57MHz. The model requires 4950 clocking cycles to generating an output, which means that each presynaptic pulse generates postsynaptic signals in 87 μs.

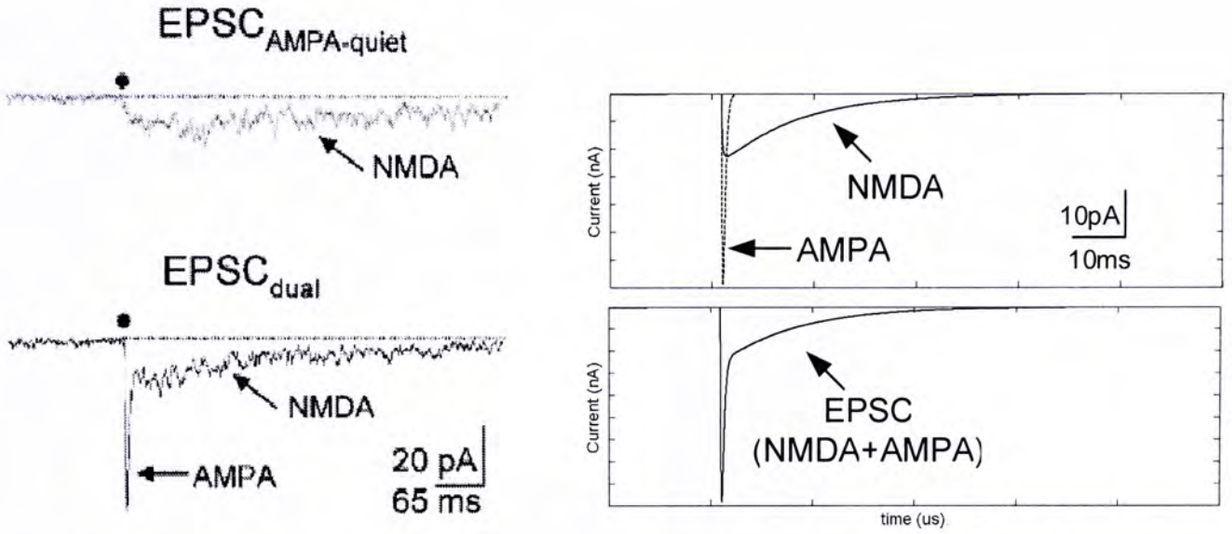


Figure 4.4 a.) Biological Recordings of individual NMDA channels, and miniature EPSCs which sum AMPA and NMDA currents, adapted from (Renger, Egles et al. 2001). b.) Simulation of FPGA circuit for the AMPA and NMDA current (upper) and the EPSC (below), which are qualitatively similar to the experimental data in (Renger, Egles et al. 2001) in a.).

In addition, we implemented the design using a Xilinx-Virtex-XC2V2000 FPGA prototyping board with ADC/DAC on the board. Figure 4.5 shows the screen capture from the output of an oscilloscope. The spikes is generating from the FPGA in real-time with the square train input. We found that the spikes are qualitatively similar to the biological measure.

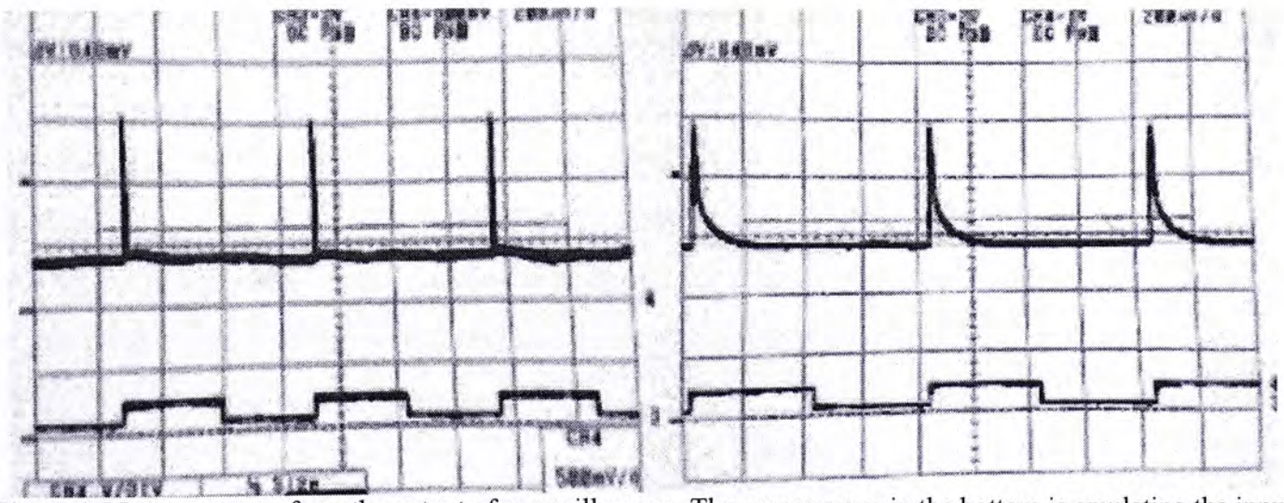


Figure 4.5 Image capture from the output of an oscilloscope. The square wave in the bottom is emulating the input signal before entering the ion channel, while the spike is emulating the action potential of from the ion channel (left) EPSC (NMDA+AMPA) spike, (right) membrane potential

Figure 4.5 shows the comparison between software and FPGA simulations of ion channel dynamics. Software computation of the model equations took 4.7 milliseconds, making FPGA faster by two orders of magnitude. This speed enhancement suggests that an FPGA system is capable of interacting with millisecond neuronal signals in real time. Additionally, the FPGA system can be configured as a stand-alone computation system without consuming any computational resources of the host computer.

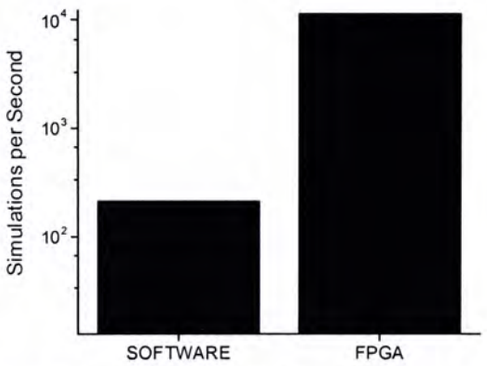


Figure 4.6 Comparison between software and FPGA for the post-synaptic ion channel simulation

4.5 Discussion

Digitally, a number is represented as bit string with fixed length and binary position. This is called fixed point representation. Some operations may cause overflow, a situation where the bit stream is longer than the allocated fixed length. This situation causes bit truncation leading to bounded numerical errors. In FPGA design, flexible bit length assignment allows us to overcome this problem and capture the output signals with any desired resolution.

The synaptic weight vector W in our proposed model is used to control the non-NMDA current magnitude. It is represented by digital bits. The FPGA allows a simple W-vector implementation because the fixed-point representation is matched with the k non-NMDA channels. Moreover, the synaptic weight can be stored indefinitely using either registers or using on-chip non-volatile Block RAM, leading to long-term memory.

Table 4.1 Comparison of resource utilization on different FPGAs

FPGA models	Logics (slice)	Memory (RAMs)	I/O	Embedded multipliers	Number of ion channels
XC2V1000	1848 (5120)	36 (40)	180 (432)	20 (40)	4
XC2V2000	2772(10752)	54 (56)	190 (624)	30 (56)	6
XC2V4000	6006 (23040)	117 (120)	195 (912)	65 (120)	13
XC2V6000	7392 (33792)	144 (144)	240 (1104)	80 (144)	16

Hardware resource (i.e. Multipliers, logic gates and RAM) in a single FPGA chip is limited, leading to tradeoffs between the resources, computational speed and accuracy. In our design, a single post-synaptic neuron consumes 26% of logic resources and 96% of the internal RAM[1]. By judiciously assigning resources, and implementing more efficient algorithms, a larger model may be designed.

4.6 Conclusion

An FPGA-based architecture for the numerical computation of NMDA and non-NMDA receptors activities and resultant synaptic plasticity has been presented. The FPGA realization computes with comparable accuracy to software implementations while operating at a much higher speeds. Additionally, the programmability of the FPGA system allows it to prototype analog circuit designs with a much shorter design cycle. Therefore, FPGA technology can be used to fill the gaps between software and hardware simulations. This technology can potentially be used as a tool suitable for dynamic clamp experiments, or to control neuro-prosthetic devices for chronic replacement of damaged neurons in central regions of the brain.

[1] The data is reference to a Xilinx Virtex-II FPGA with 24K logic resources and 1Mbits memory

Chapter 5

Continuous-Time and Discrete-Time Inference Networks for Distributed Dynamic Programming

Dynamic programming is a step of crucial importance in real-time decision making reinforcement learning problem where problem solution time can be an implementation bottleneck. We consider a distributed computational framework for continuous-time inference network for solving dynamic programming problems. Interconnected computational units, forming a network, participate simultaneously in the computation while maintaining coordination by information exchange via continuous communication link. The implementation of the value-iteration algorithm of dynamic programming for the expected average cost criterion is presented. We show that the inference network converges to the optimal Bellman optimality condition, for which the convergence rate can be made arbitrarily fast and is practically independent of the discounting factor and the number of states. Numerical simulation of using such continuous-time inference network for random-walk and stochastic shortest path problems are also described and compared. We also derived a discrete-time version of the inference network that the network can be well mapped to FPGAs for solving dynamic programming.

Nomenclatures and definitions

MDPs	Markov decision process
s	state at MDPs
a	action
a_{ik}	taking action at current state i and state k is the desirable state at the next transition
s'	next state, after taking action a
Σ	a set of state
A	a set of actions
$R(s, a)$	reward function, given set of states and action
$T(s, a, s')$	the probability of making a transition from state s to state s' using action a
γ	discount rate for the long-term reward
$V(s)$	expected reward at state s
π	policy, the rule to infer action from the value $V(s)$
BRIN	Binary relation inference network
$U(s)$	computational unit corresponding to the state s at the original problem
$g(i,j)$	output of the unit function at each computational unit, which is the binary relation for objects i and j.
$g(s)$	output of the computation unit for value at state s with an implicit reference state s'.
$G(.)$	The unit function, which is the "min" operator, designated for the MDPs problem

5.1 Introduction

Dynamic programming is an important step in time-critical decision making, such as real-time path planning and route finding. Autonomous robots and vehicles are assigned to perform missions in highly hazardous and extreme environments. In most of the time, good path planning and quick reaction to avoid dangerous spots can increase the chance to reach the target or accomplish a mission, as in the cases of gathering scientific information from a distant planet or searching and rescuing life from a mass casualty incident site (Rover Team 1997; Volpe, Estlin et al. 2000; Casper and Murphy 2003). On the other hand, exploration of the surface of a distant planet by networks of autonomous cooperating vehicles would be an effective alternative approach. Collective information from a distributed environment would increase the content of the information and increase the chance of survival of the robots. Given this option, distributed approaches for rapid learning in an uncertain environment and for making real-time decision is in great demand (Williams, Kim et al. 2001).

Among the examples of real-time path planning and decision making is the development of intelligent Unmanned Aerial Vehicles (UAVs) for future combat to reduce human causalities. The major challenge for intelligent UAVs development is path planning in uncertain and even adversarial environments, for which the objective is to complete the given mission, to arrive at the given target within a pre-specified time, while maximizing the safety of the UAVs. The problem can be modeled as a typical stochastic learning and sequential decision problem (Jun and D'Andrea 2003). However, in practice, the UAVs path planning is difficult because of two main reasons. Firstly, in the adversarial environment, information is always incomplete and is highly uncertain. It is difficult to acquire knowledge to decide on a reasonably well trajectory of the flight. Secondly, the computational load grows quickly as the number of radar sites increases. Delay in the decision making, due to heavy computational burden, would be disastrous. The distributed and parallel computational learning architecture could circumvent the limitation of the sequential computation machineries (Tin 2004).

The optimal decision making problem is formulated based on a well defined Markov decision process, in which the set of states, actions, rewards and probability distributions are given. It is aimed to find a value function, which can correctly give the expected reward at each state. In theory, the optimality condition is defined by the Bellman equation. It has been shown that the guarantee of optimality and efficiency for complex decision making problems using dynamic

programming. However, in many practical scenarios, the dynamic programming (also known as *Value Iteration)*, which works by producing successive approximations of the optimal solution, suffers from the high computational load. Not only is it computationally intensive within each iteration, unfortunately, the number of iterations required can also grow exponentially (Condon 1992). Modification of algorithm looking for computational trade-off between cost per iteration and the number of iterations was proposed, along with boundary improvement for some problems.

Besides, by applying linear programming and conic optimization techniques, the problem can be solved by general-purpose linear programming packages (D'Epenoux 1963; Hoffman and Karp 1966; Derman 1970). An advantage of this approach is that commercial-quality linear programming packages are available. From a theoretic perspective, linear programming is the only known algorithm that can solve the optimal decision problem in polynomial time, although the theoretically efficient algorithms have not been shown to be efficient in practice.

Despite decades of study, stochastic optimization problems remain far more computationally challenging than their deterministic counterparts. For the problem with high dimension and large number of states, it takes a relatively long time to obtain the optimal solution. The high computational load delays the decision making process to the extent that time-critical decision application becomes infeasible. A distributed and parallel approach can potentially circumvent the computational obstacle as an efficient alternative.

The structure of dynamic programming naturally lends itself well to distributed computation. The former involves calculations and so can be carried out in parallel to a great extent. In (Bertsekas 1982), Bertsekas proposed a model of asynchronous distributed computation for dynamic programming. The proposed model considered a broad range of problems and includes shortest path problems, finite and infinite horizon stochastic optimal control problems. Later, Jalali and Ferguson proposed a distributed computational model for Value Iteration algorithm (Jalali and Ferguson 1992). In line with Bertseka's work, they derived a distributed version of the algorithm, which compute the optimal expected average cost based on the Bellman optimality criterion. Theoretical analysis showed that the distributed approaches gain speed-up by taking advantage of the intrinsic independencies from the Bellman optimally criterion. Among the previous analytical works of distributed dynamic programming, which was assumed to have unpredictable delay in communication between different processors, practical

applications are sparse. While the work in [11] is dedicated and limited to the network of general purpose computers and assumes loose and delayed communication, computationally inefficiency might be expected in real time application.

Alternatively, heuristic approaches concentrate computational effort on the areas of the state-space that is most likely to be visited. This approach was proposed as a way of approximately solving the Bellman optimality equation. In (Barto, Bradtke et al. 1995), Real-Time Dynamic Programming (RTDP) was proposed. It is specific to problems in which the agent is trying to achieve a particular goal state and the reward everywhere else is zero. Further, implementation of RTDP using embedded dedicated processor provides further speed-up for the optimization of the dynamic programming. However, the optimality of the solution is not always guaranteed and the approach is limited to a fine scope of problems. In general, the approaches mentioned above all fall into the category of digital computation, in which the intrinsic difficulties and limitations are largely inherent in the digital and sequential computation paradigm.

When comparing to the digital computation paradigm, natural biological and physical systems provide an effective and collective environment to solve difficult computational problem in high speed. The computational powers routinely used by biological nervous system to solve perceptual problems must be truly immense, given the massive amount of sensory data continuously being processed, the inherent difficulty of the recognition tasks to be solved and the high performance speed, for which answers must be found (Mead 1989). Parallel processing in the nervous systems provides computational speed and power, which allows mammalian visual system perform elementary feature recognition massively in parallel (Ballard, Hinton et al. 1983).

The major feature of neural organization can act synergistically with parallel feedback and connectivity to greatly enhance computational power. This feature enables the biological system to operate in a collective analog mode (Hopfield 1982; Hopfield 1984), with each neuron summing the inputs of hundreds or thousands of others in order to determine its graded output. The parallel analog computation in a network of neurons is thus a natural way to organize a nervous system to solve optimization problems in continuous-time processing. However, the basic formulation of Bellman optimally criterion is in discrete-time. There is still an opening between the basic Bellman equation formulation and the continuous-time collective network solution which requires extra effort and formulation. In contrast, continuous-time network has shown to be an effective solution for deterministic shortest path problem (Lam 1991).

Optimization of shortest distance between cities are considered and mapped to a continuous-time dynamic network. It can be readily realized using analog design and drastic acceleration was found when compared to the conventional sequential dynamic programming approach (Lam and Tong 1996).

In this chapter, we present a distributed continuous-time inference network for dynamic programming problems. We show that the inference network converges to the optimal Bellman optimality condition, for which the convergence rate can be made arbitrarily fast and is practically independent of the discounting factor and the number of states. Further, an analog VLSI CMOS circuit has been developed to realize the inference network and to demonstrate its optimization for the dynamic programming problem. The following two sections discuss the theoretical foundation of such network and the numerical simulation results are discussed in section 4. When comparing to the counterpart of the discrete-time asynchronous dynamic programming (Bertsekas 1982; Jalali and Ferguson 1992), the continuous-time inference network provides a general concurrent mechanism to find the optimal solution based on the Bellman equation.

5.2 Background

Reinforcement learning (RL) is learning from interaction with an environment, from the consequences of action, rather than from explicit teaching. The learning algorithm is inspired by observing animal learning and being formulated within the mathematical framework of Markov decision process. RL algorithms are methods for solving this kind of problem, that is, problems involving sequences of decisions in which each decision affects what opportunities are available later, in which the effects need not be deterministic, and in which there are long-term goals. RL methods are intended to address the kind of learning and decision making problems that people and animals face in their normal, everyday lives.

In the standard reinforcement learning model, the learning system (or agent) is connected to its environment via perception and action, as depicted in the Figure 5.1. The agent interacts with the environment by receiving cost and state after perform an action. The uncertainty, which might be noise, can be modeled as perturbation on the return of state form the environment. In other words, when taking an action a at state s, the descendent state measure from the environment would be

state s' with probability $T(s, a, s')$. Thus the stochastic model is powerful, as captures the variability of the uncertainty of the system dynamic using probability models, which is called Markov decision process[1] (MDP).

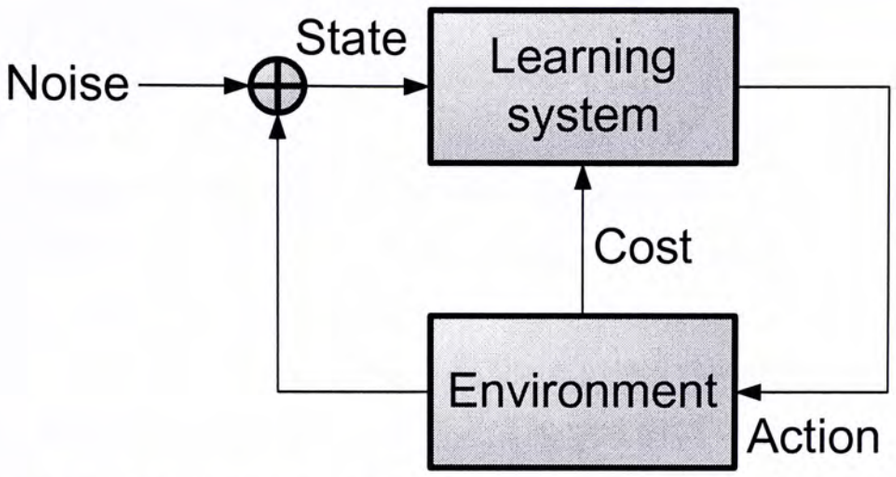


Figure 5.1 Learning system interacting with its environment. Action is applied to the environment while the corresponding cost and new state are feedback to the learning system afterwards. However, at the path from the environment to the learning system, noise is introduced resulting state perturbation. State can be regarded as measurement from the learning system. Thus, the measure can be sometime deviated.

5.2.1 Markov decision process (MDPs)

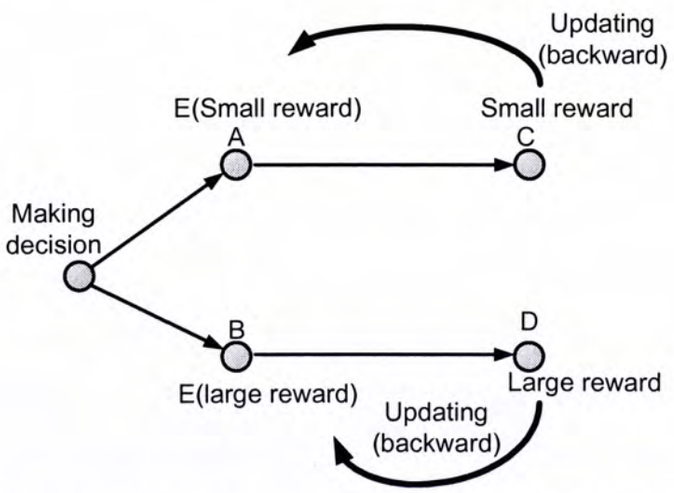


Figure 5.2 In reinforcement learning problem, the agent will get the reward after taking the action. Assume that the agent want to maximize the reward, the agent will have no idea in the beginning which decision would return a

[1] Markov decision process has more rigid definite in the perspective of optimal control. An MDP consists of states, actions, reward function and state transition function, which is a probability distribution over the state. It also follows the Markov criterion, in which the state transitions are independent of any previous environment states or agent actions.

larger reward. Fortunately, the agent can still make decision based on the expected reward of the node, which is an attribute or record attaching at the node. Agent will prefer the action with high expected reward. In the figure, following the pathway A, the agent will get a smaller reward while following the pathway B will get a larger reward. The expected reward of the node A and B can be computed by backward updating. The expected reward of previous state (A) is updated when the agent reach state (C) and obtain the reward. Thus, to update the expected reward in previous state requires information about reward in the next state. Always, the reward is uncertain, and modeled as Markov decision process.

The agent can making decision based on the *expected reward* (or value). Expected reward is the expectation of reward the agent will receive in a long run. Usually, the agent will be more willing to choose a state with high reward expectation. In Figure 5.2, an agent has to make decision to choose between pathway A and B. As the expected reward at node B is larger than node A, the agent will prefer to choose the node B. The next question would be how to evaluate the expected reward? It can be evaluated by backward updating based on the expected reward in the successor state. For example, in Figure 5.2, the reward at state D is larger than at state C. By using backward updating, the expected reward at state B is, therefore, larger than at state A. While more solid definition of backward updating is defined by Bellman, with his famous Bellman optimality equation (or simply Bellman equation). With iteratively applying the Bellman equation on all nodes repeatedly, the expected reward can be obtained.

The reinforcement learning problem with computation expected reward are well modeled as *Markov decision process* (MDPs). An MDP consists of

- a set of state Σ,
- a set of action A,
- a reward function $R(s, a)$, and
- a state transition function T , where a member of T is a probability distribution over the set Σ (i.e. it maps state and action pair to probabilities). We write $T(s, a, s')$ for the probability of making a transition from state s to state s' using action a.

The state transition function probabilistically specifies the next state of the environment as a function of its current state and the agent's action. The reward function specifies expected instantaneous reward as a function of the current state and action. There are many good references to MDP models (Bellman 1957; Bertsekas 1987).

5.2.2 Learning in the MDPs

In the MDP environment, the agent's goal is to maximize the reward it received in a long run. In previous chapter, we have mentioned that the uncertain environment is unknown. Reinforcement learning is primarily concerned with how to obtain the optimal policy when such a model is not known in advance. Therefore, interaction between the agent and the environment is necessity to obtain the stochastic distribution of the environment $\Pi(S)$ and to find an optimal policy. In other words, reinforcement learning consists of two tasks, which are 1.) to explore and learn a probabilistic model and 2.) to use the model to derive an optimal policy. This kind of reinforcement learning approach is called the model-based method (Kealbling, Littman et al. 1996; Sutton and Barto 1998).

Learning a probabilistic model from an unknown uncertain environment can be straightforward by exploring the environment and keeping statistics about the results of each action. While given the statistical model of an environment, derivation of an optimal policy is a computational intensive task and requires parallel processing to speed up. In the coming section, we will focus on the optimization of policy based on the probabilistic model (Kealbling, Littman et al. 1996).

5.2.3 Bellman optimally criterion

The objective of an agent is to maximize the received reward in a long run. Alternatively, it can be thought as to find a rule for the agent to take action that the long term reward can be maximized. The sequence of rewards receives after time step t is denoted r_{t+1}, r_{t+2}, r_{t+3}, ... while the total expected return at time t is $R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T$, where T is a final time step. In some aspect, the reward in the future is not as important as the immediate reward. This is the discounting of future reward, that the total expected return at time t can be written as

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{5.1}$$

Where γ is a parameter, $0 \le \gamma \le 1$, called the discount rate.

The received reward in a long run is considered as the expected return for the agent staying at the MDPs infinitely long. At each state, which associates with an expected reward (or value); the value is the expected infinite discounted sum of reward that the agent will gain if it starts in the

state and executes the optimal policy. Using π as a complete decision policy, the expected reward (value) at state s equals to the sum of the actual received reward and the best expected reward in the subsequent state.

$$\text{expected reward at state } s = \text{reward} + \min_{\pi}(\text{discount} \times \text{expected reward at state } s') \quad (5.2)$$

The equation 1.2 outlines the basic idea of the Bellman optimality equation, which defines the relationships of the expected reward between different states. Suppose the expected reward at state s is denoted by $V(s)$ and the actual received reward after taking action a from state s is denoted as $R(s, a)$. The Bellman equation for optimal value function $V(\cdot)$ is unique and can be defined as the solution to the recursive equation

$$V^*(s) = \min_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s') \right), \forall s \in S \quad (5.3)$$

which assert that the value of a state s is the expected instantaneous reward plus the expected discounted value of the next state, using the best available action. Given the optimal value function, we can specify the optimal policy as

$$\pi^*(s) = \arg\min_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s') \right) \quad (5.4)$$

The formulation of optimal policy depends on the optimal expected V^* reward in every states. After obtain the optimal expected reward, computing the optimal policy is straightforward, by applying equation 1.4 on all states. However, it is not trivial to evaluate the optimal expected reward. The equation 1.3 only specifies the interrelationships of the optimal reward between states. But equation 1.3 indeed is not an algorithm to optimize the value.

5.2.4 Value iteration

To find the optimal expected reward in an MDPs is a well-studied problem. Among the huge literatures on the discussion of the optimization algorithm, value iteration is simple and effective and can be shown to converge to the optimal V^* values (Bellman, 1957, Bersekas 1987). The value iteration is simply applying the Bellman equation (equation 1.3) for all possible state and

actions over and over again, until the value is converged. The algorithm can be expressed as three levels of iterative loops.

Algorithm 5.1 *Value Iteration*

Initialize V(s) arbitrarily
loop until policy good enough
    loop for $s \in S$
        loop for $a \in A$
            $Q(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')$
        end loop
        $V(s) = \min_a Q(s,a)$
    end loop
end loop

It is not obvious when to stop the value iteration algorithm. One intuitive approach is to stop when the maximum difference between successive value function is less than a small number. This could be an effective criterion to stop.

It can be noted that the value iteration algorithm consists of three iteration loops. For the innermost loops, the evaluation of the summation exhausting all possible states, thus, the computational complexity, per iteration, is quadratic in the number of states and linear in the number of actions. In addition, the number of the iterations is uncertain, and could be increasing exponentially with the number of states. The algorithm is computationally intensive and we are looking for distributed approaches to speed up the computation.

5.3 A Computational Framework for Continuous-Time Inference Network

Although the Bellman optimization algorithms were originally developed for sequential computation, most of the research efforts in their parallel implementation focus on the distributed discrete-time asynchronous design, based on the Bellman optimality criterion (Bertsekas 1982; Jalali and Ferguson 1992). In such a case, significant computational burden at processors and communication delays were expected to be encountered. Slow convergence rate might be

expected in large network. Connectionist approach with interconnected simple computational units provides distributive computational power. This kind of computational paradigm provides collective computational power could significant be better performance than sequential execution or computation.

For example, in the biological nervous system, signal propagation with high speed performance even the network is huge and complex, that the electrical and chemical signal transmitted in massively parallel and under the continuous-time physical dynamic. The continuous-time systems with collective analog computational circuits dynamically provide speed, needed for real-time processing. Pioneers, such as Hopfield and Tank, by mapping difficult combinatorial optimization problem, traveling salesman problem (TSP), to continuous-time analog circuit demonstrated collective computational abilities of power and speed, which digital computer would fail to provide (Hopfield and Tank 1985).

In the continuous-time network, the optimization problem was formulated as a set of differential equations, which is essentially being well mapped to the circuit differential equation and the circuit differential equation is essentially a program by which an answer to a question can be found. The converged results of the circuit following the differential dynamics would eventually become the solution of the optimization problem. The MDPs Bellman optimization can also be solved using the dynamic network approach to gain speed through the collective power of a network of computational units.

5.3.1 Binary relation inference network

Mapping Bellman recursive dynamic programming to a continuous-time computation paradigm can be realized and boosted with the introduction of a connectionist network architecture, called Binary relation inference network (BRIN). The network has a parallel architecture, and can be used to derive unknown binary relations through the simultaneous propagation of successive inferences. Originally, it provides an efficient platform for checking data inconsistancy due to results from different inference paths (Lam 1996). In (Lam and Tong 1996), with close resemblance to the deterministic type dynamic programming formulation on closed semiring, Lam and Tong introduced BRIN to solve a set of graph optimization problems with an asynchronous and continuous-time computational framework. In contrast to its close counterpart, discrete-time BRIN, in which some significant limitations about the network

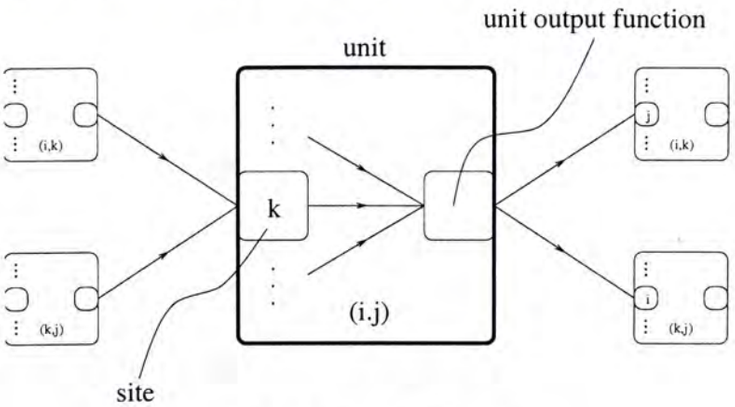
instability and oscillation under specific circumstances, and the slow convergence rate commonly observed in large network were found (Lam 1996). This new class of continuous-time inference network is inherently stable in all cases and it has been shown to be robust and with arbitrarily fast convergence rate (Lam 1991; Lam 1996). In addition, the continuous-time network is readily leads to real-time application using analog VLSI circuit (Ng 1996).

A binary relation inference network is formed by the interconnection of self contained computational units. Figure 5.3 shows the structure of a unit and the connections in a general inference network. Each unit is to represent a binary relation (i, j) between two objects i and j. In each unit, there are N sites to carry out the inference operations as defined in the site function $S(\cdot)$. The value of the corresponding relation between i and j is then determined by resolving the conflict among all of the site outputs. In essence, if $S_k(i, j)$ represents the site output at the k-th site and $g(i, j)$ stands for the unit output of unit (i, j), then

$$S_k(i, j) = g(i, k) \circ g(k, j) \tag{5.5}$$

$$g(i, j) = \bigcup_{\forall k} S_k(i, j) \tag{5.6}$$

where $\circ$ is the inference operator for the site function (which is usually the same at all of the sites) and $\bigcup$ is the conflict-resolution operator for the unit function. Also the computational unit $U(i, j)$ denotes the unit which resolves the binary relation (i, j).



$$1 \le i, j, k \le N; k \ne i, j$$

Figure 5.3 Unit interconnection in a general binary relation inference network

5.3.2 Binary relation inference network for MDPs

Markov decision process (MDPs) optimization based on the Bellman criterion is particularly well suited to be solved on the binary relation inference network. The optimal *value* for state s, $V(s)$ is the expected discounted sum of reward that the agent will gain if it starts in that state. Considered that a typical MDP to be a single-destination stochastic shortest path problem. Each state represents a city and there are m cities, where the m-th city is referred to the destination. The optimal value of each city, $V^*(s)$, is referring to the expected shortest distance from the city s to the destination city.

The MDPs shortest path problem can be mapped to the inference network. For the original problem graph, each node or state refers to a city. But in the inference network, each computational unit $U(i, j)$ represents the binary relation, thus the shortest distance between city i and city j. Thus, when the network has converged, the solution of the problem would be found at the output of each computational unit. For the stochastic shortest path problem, $g(i, j)$, the output of each computational unit, take the semantic meaning of the expected shortest distance between cities i and j. Since, the destination city is fixed, as we consider a multiple-source and single destination case, city j always refers to the destination city. Thus, $g(i, j)$ can be simply denoted by $g(i)$. Also the computational unit is simply denoted by $U(i)$. In general, if there are m states in the original graph, then the BRIN network (based on the Bellman-Ford single destination (or source) formulation) will have m-1 value function units $U(i)$ with reference to a designated reference state, the destination, which is "implicit" in the $U(i)$ notation.

Consider a specific unit $U(i)$, the output of $U(i)$ is denoted by $g(i)$, where each unit only has one and only one output function. Each unit has k sites, and each site has site function $f_p(i), p = 1,...,k$. The site function $f_p(i)$ computes the expectation of reward based on the value of at unit $U(p)$. Thus, for mapping the Bellman equation to the inference network, the site function, which defines the computational expectation of rewards for taking the action a can be stated as

$$f_a(i) = R(i,a) + \gamma \cdot E[g(p)], \ \forall a \in \mathrm{A}_i \tag{5.7}$$

where $E[.]$ is the expectation. Further, given the probability distribution, the site function $f_a(i)$ for each state i can be stated as

$$f_a(i) = R(i,a) + \gamma \cdot \sum_{i' \in S} T(i,a,i')g(i'), \ \forall a \in A_i \tag{5.8}$$

where $R(i, a)$ is the reward received when taking action a at state i, and $T(i, a, i')$ is the probability of arrived at state i' when taking action a at state i.

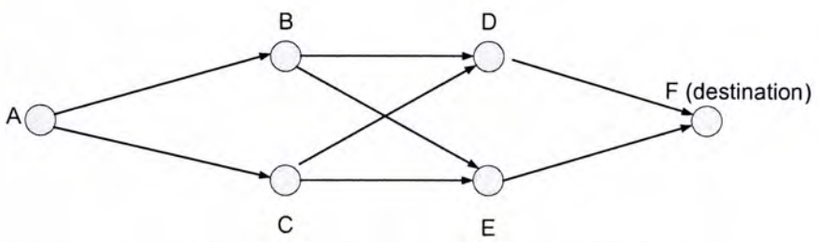
According to the Bellman optimality criterion, for the optimal decision would be the one with the least cost. Therefore, the unit function is defined as

$$g(i) = G\{f_1(i,j), f_2(i,j), ..., f_k(i,j)\} \tag{5.9}$$

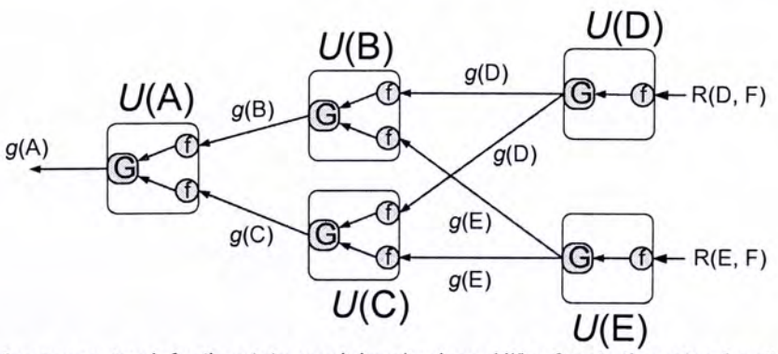
$$= \min\{f_a(i), a = 1, ..., k\} \tag{5.10}$$

$$= \min_{\forall a}\{f_a(i)\} \tag{5.11}$$

Considers a typical 6-state MDP example, where 6-th state F is considered as the destination or terminal. The state graph topology is shown in figure 5.4. Each node represents one state and the arc is the possible path between two nodes. If there is no arc in between two node, it means that the cost between these two nodes is infinite. It is aiming to determine the value of each node, which is the expected shortest path cost to the destination city.



a.) A typical 6-state MDPs problem, where state F is the regarded as destination and state A is regarded as start.



b.) Inference network for the state graph in a.), where $U(i)$ refers to the value function unit i and $g(i)$ refers to the output of the unit function at $U(i)$. $g(i)$ takes semantic meaning of the minimum expected cost between states i and the destination.

Figure 5.4 a.) Original problem state graph b.) inference network for solving graph in a.)

The corresponding inference network has only five computational units. A close resemblance can be found between the inference network and the original state graph. Computational unit $U(i)$ represents the expected shortest distance from city i to the destination. Therefore, the unit for city F, U(F), which is always zero, can be omitted. We have assumed that the distance between two cities are infinite if there is no directed path between them. Then each unit is not connected to all other units, but only connects to its neighborhood units, which is the decedent state from the original graph. For the units, U(D) and U(E), which the site function simply equals to the received reward to city F.

5.3.3 Continuous-time inference network for MDPs

Assume that the min operator requires an infinitesimal time (δt) for evaluation, then it seems to be straightforward to use

$$g_{t+\delta t}(i) = G_t(i) \tag{5.12}$$

Let's make a further assumption that each individual unit behaves dynamically as a first-order system, described by the differential equation

$$\frac{dg_t(i)}{dt} = -\lambda_i g_t(i) + \lambda_i \{\min_{\forall p} \{f_p(i)\}\} \tag{5.13}$$

$$= -\lambda_i g_t(i) + \lambda_i \min_{\forall a} \left\{ R(i,i') + \sum_{i' \in S} T(i,a,i') g(i') \right\} \tag{5.14}$$

where λ_i is the open-loop system pole for unit (i) which controls the rate of how $g_t(i)$ may change. If $\lambda_i = 0$ then $dg_{t(i,j)}/dt = 0$ and $g_t(i)$ is a constant the unit is said to be fully constrained and has a fixed memory. Whereas for a memoryless unit with $\lambda_i = \infty$, it has infinite power to change because $|dg_{t(i,j)}/dt|$ can be made arbitrarily large.

A computational framework for solving the optimal policy problem can be formulated as follows:

1. Construct a m-unit continuous-time inference network, where the dynamic behavior of each individual unit is governed by a parameterized differential equation of the form of equation (1.11).

2. The network has a complex but regular interconnection structure following the topological structure of the given by the problem or graph. Each unit $U(i)$ sends its output to a set of neighborhood units, which is the decedent state in the original problem graph, can reach the unit state i by taking proper action. Each unit received at the same time the outputs from the set of neighborhood units, which can be reached from state i by taking proper action.
3. Initialize the network units with randomized the output value of each unit $g_0(i)$.
4. The network will converge to a global optima arbitrarily fast, at a rate dependent on all distinct λ_i parameters. The converged output $g_\infty(i)$ from each individual unit (i) is the incurred minimum expected reward according to the definition based on the Bellmen optimality criterion.
5. Using a tag processor to keep track of the site functions at each individual unit (i), such that $\pi_\infty(i) = \arg\min_{\forall a}\{f_a(i)\}$, where π(i) is the optimal action at state i, can be obtained.

5.4 Convergence Consideration

There are two important considerations in using the inference network for dynamic programming. Firstly, will the network always converge to the desired solution? Secondly, what are the parameters or conditions that affect the convergence rate of the network? The answer to the first question is an affirmative 'yes', because it follows directly from the principle of Bellman optimality equation that all constituent optimal expected value of all states are optimal. The local minimization based on the Bellman equation performed at each distinct unit, in fact, is driving the network to a global optimal state, which is the desired solution. To measure the 'distance' of the network from this global minimum, we can define the following computational energy E(t), where For the continues time dynamic equation, which can be stated as

$$E(t) = \sum_i \left\{ g_i(s) - \min_{\forall a \in A} \left\{ R(i,i') + \sum_{i' \in S} T(i,a,i') g(i') \right\} \right\}^2, \ \forall i \tag{5.15}$$

To determine the convergence rate of the network, we need to derive an explicit expression for $dE(t)/dt$. Using Eq. ((5.13)-(5.14)) and noting that

$$\frac{dE(t)}{dt} = \frac{dE(t)}{dg_t(i)} \cdot \frac{dg_t(i)}{dt} \tag{5.16}$$

We have

$$\begin{aligned}\frac{dE(t)}{dt} = \sum_i -2\lambda_t \{g_t(s) - \min_{\forall a \in A}\left\{R(i,i') + \sum_{i' \in S} T(i,a,i')g(i')\right\} \\ \cdot \frac{d}{dg_t(s)}\left\{g_t(s) - \min_{\forall a \in A}\left\{R(i,i') + \sum_{i' \in S} T(i,a,i')g(i')\right\}\right\} \\ \cdot \left\{g_t(s) - \min_{\forall a \in A}\left\{R(i,i') + \sum_{i' \in S} T(i,a,i')g(i')\right\}\right\}\end{aligned} \tag{5.17}$$

In order to resolve Eq. (5.16), a necessary condition is required. In this case, we simply assume $i \neq i'$. The meaning of this assumption is that, if the current node is i, the next node should not be i itself. Then the above expression equals to one.

Thus, we hava

$$\begin{aligned}\frac{dE(t)}{dt} &= \frac{dE(t)}{dg_t(i)} \cdot \frac{dg_t(i)}{dt} \\ &\leq \sum_s -2\lambda_t \{g_t(s) - \min_{\forall a \in A}\{C(s,a) + \gamma \sum_{\forall s' \in S} P_{s,s'}(a) \cdot g_t(s')\}\}^2 \\ &\leq 0\end{aligned}$$

The non-negative parameters λ_i for all distinct units, can therefore be used to achieve arbitrarily fast convergence for the inference network.

5.5 Numerical Simulation

5.5.1 Example 1: Random walk

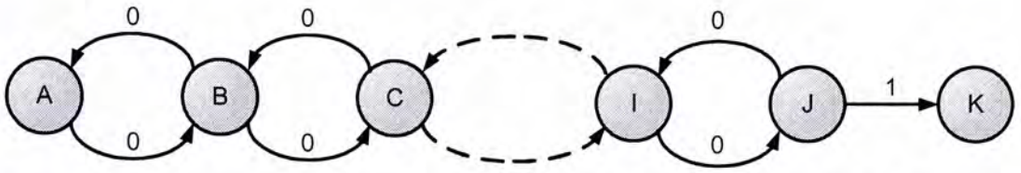


Figure 5.5 A small Markov process for generating random walks. The chain has 11 states in total, while the state 'K' is the terminal state. Transitions would result zero reward except the "right" action at state J would result reward of one, as shown in the arcs in the figure. The semantic meaning is that one would move to the left state with probability 0.8 given taking the action "left", or one will arrive at the states on the right with probability 0.2. The objective of dynamic programming is to evaluate the discounted expected reward at each state from A to J.

A 10-state random walk problem can be solved by a 10-unit continuous-time inference network. The 10 state indexed by {A, B, C, D, E, F, G, H, I, J}. The 10 units output of the network are described by a vector $\mathbf{x_t} = [g_t(A), g_t(B), g_t(C), g_t(D), g_t(E), g_t(F), g_t(G), g_t(H), g_t(I), g_t(J)]$, which has semantic meaning of expected reward of $V_x=[V_A, V_B, V_C,\ldots,V_J]$. The underlying transition probability distribution of the environment is defined as follows. For state i, the available actions is denoted by $\mathrm{A}_i = \{a_{ik}\}$, where the second subscript k in action a_{ik} taken by the learning system indicates the availability of more than one possible action when the environment is in state i and state k is the desirable next state. The transition of the environment from the state i to the new state j, for example, due to action a_{ik} is probabilistic in nature. The transition probability of the environment from state i to state j by taking action a_{ik} be defined as $T(i, a_{ik}, j)$. In our experiment, probability $T(i, a_{ik}, i') = 0.9$ if $k = i'$ and $\sum_{\forall k} T(i, a_{ik}, i') = 0.1$ if $k \neq i'$, where i is the current state, i' is the next state and a_{ik} is the action with k is the desirable next state.

The continuous-time inference network can be modeled by a set of differential equations on the 10 nodes *A,B, ..., J*. The expected rewards $V_A, V_B, \ldots, V_J$ evolve as first-order lag controlled by λ (a system or implementation related parameter which is non-problem related). γ and ρ are two problem-related parameters, defined as the discount factor for multistage reward and the transition probability, respectively.

$$\frac{dV_A}{dt} = -\lambda V_A + \lambda\gamma V_B \tag{5.18}$$

$$\frac{dV_C}{dt} = -\lambda V_C + \lambda \max\begin{cases} \gamma\left[\rho V_B + (1-\rho)V_D\right] & \text{for left;} \\ \gamma\left[\rho V_D + (1-\rho)V_B\right] & \text{for right.} \end{cases} \tag{5.19}$$

$$\frac{dV_J}{dt} = -\lambda V_J + \lambda \max\begin{cases} 1-\rho+\gamma\rho V_I & \text{for left;} \\ \rho+\gamma(1-\rho)V_I & \text{for right.} \end{cases} \tag{5.20}$$

Eq.(5.18) describes V_A of the boundary node A which has a single "right" action of zero reward. For nodes $B, C, \ldots, I$, they are having both left and right actions and can be readily shown to follow equations as typified in Eq.(5.19) for C. K is a goal state which should have its expected reward V_K defined as zero. A goal state has no associated actions and zero reward as the "goal" is reached. Eq.(5.20) describing V_J can thus be derived. Note that the non-zero immediate expected rewards from J to K and from J to I are obtained as $\left(\rho\times 1+(1-\rho)\times 0\right)$ and $\left(\rho\times 0+(1-\rho)\times 1\right)$, respectively.

Given arbitrary positive initial values of $V_A, \ldots, V_J$, the converged values of the respective differential equations (Eqs.(5.18)-(5.20)) can be verified to be identical with the optimal values $\left(V_A^*, V_B^*, \ldots, V_J^*\right)$ governed by the Bellman equations. Figure 5.6 shows the case for the problem defined by $\rho = \gamma = 0.9$, and using $\lambda = 0.9$. The converged values are found to be [0.3160, 0.3511, 0.3984, 0.4528, 0.5147, 0.5852, 0.6652, 0.7563, 0.8597, 0.9774]. For cases with ρ and γ close to 1, the optimal policy is quite straightforward as each state will take an action to the right. The Bellman equations then reduce to a much simpler set of linear equations, given by:

$$V_A^* - \gamma V_B^* = 0 \tag{5.21}$$

$$-\gamma(1-\rho)V_B^* + V_C^* - \gamma\rho V_D^* = 0 \tag{5.22}$$

$$-\gamma(1-\rho)V_I^* + V_J^* = \rho \tag{5.23}$$

For simplicity, only Eq.(5.22) is given for node C; similar equations can readily be obtained for nodes $B, D, \ldots, I$. By solving this tridiagonal equation set (Eqs.(5.21)-(5.23)), $V_A^*, \ldots, V_J^*$ are determined and found to be the same as the converged values of the differential equations.

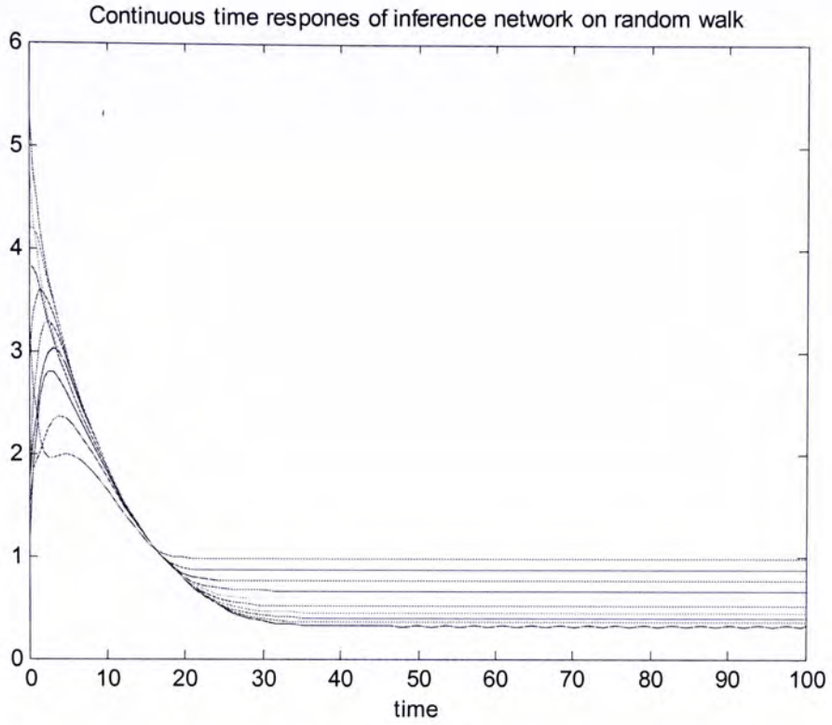


Figure 5.6: Convergence of the differential equations for $V_A,\ldots,V_J$

It is interesting to consider a slight variation of the problem by the inclusion of an arc (with zero reward) from node *K* to node *J*, and to find out how the continuous-time network adapts to the changes of optimal solution. Noting that *K* is now not a goal state with a possible left action, V_K is not necessarily zero. The Bellman equation for V_K^* is modified, together with the inclusion of a new equation for V_K^*.

$$-\gamma(1-\rho)V_I^* + V_J^* - \gamma\rho V_K^* = \rho \tag{5.24}$$

$$-\gamma V_J^* + V_K^* = 0 \tag{5.25}$$

The optimal values for $V_A^*,\ldots,V_J^*,V_K^*$ can be found from the modified tridiagonal equation set, to give [1.5169, 1.6854, 1.9122, 2.1735, 2.4708, 2.8089, 3.1933, 3.6302, 4.1269, 4.6916, 4.2224]. For the differential equations of the continuous-time inference network, the essential changes are given in Eq.((5.26)-(5.27)). Figure 5.7 shows the network adapts well to the problem change and gives converged values which agree with the new optimal values.

$$\frac{dV_J}{dt} = -\lambda V_J + \lambda \max\begin{cases} 1-\rho+\gamma\left[\rho V_I + (1-\rho)V_K\right] & \text{for left;} \\ \rho+\gamma\left[\rho V_K + (1-\rho)V_I\right] & \text{for right.} \end{cases} \tag{5.26}$$

$$\frac{dV_K}{dt} = -\lambda V_K + \lambda \gamma V_J \tag{5.27}$$

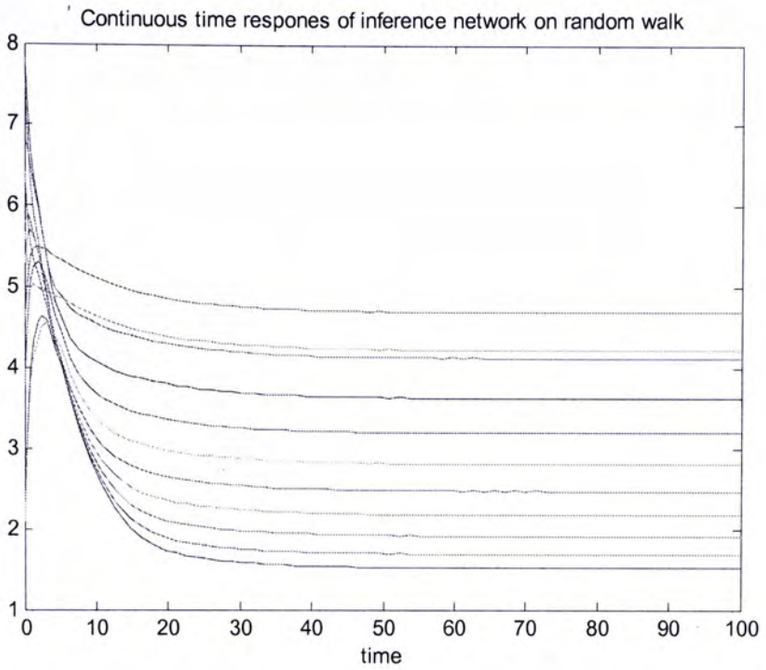


Figure 5.7: Convergence of the differential equations for $V_A, \ldots, V_J, V_K$

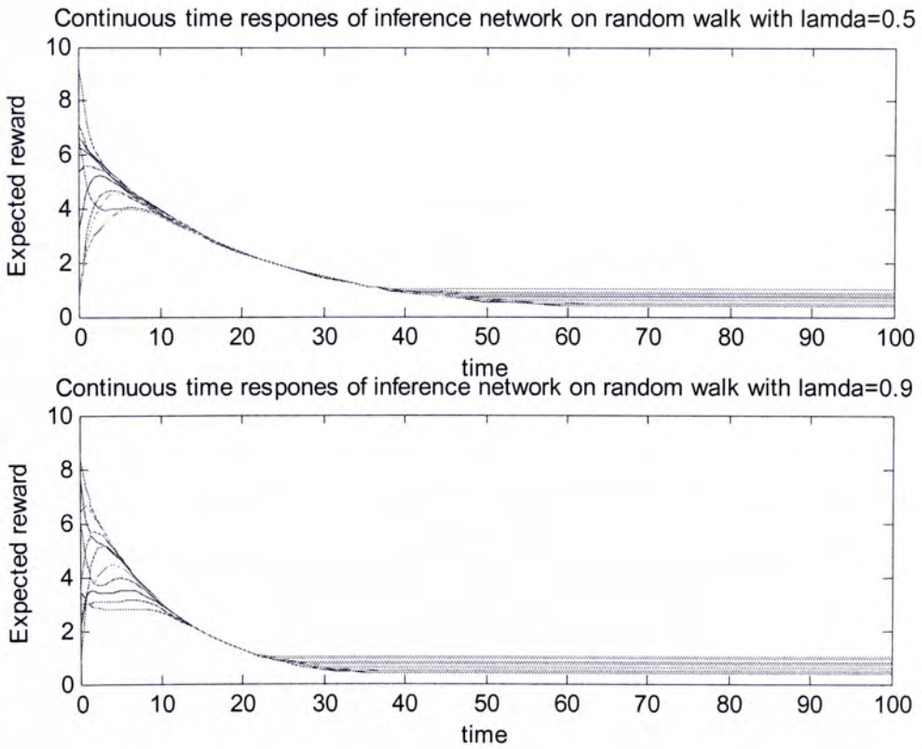


Figure 5.8 Simulation of a 10-unit inference network for 10-state random walk (upper: λ_i=0.5, γ=0.9, below: λ_i=0.9, γ=0.9,)

Further, the difference control parameter λ, the convergence of the network would be different. The λ is machine dependent, which is a parameter depends on the implementation methodologies. Figure 5.8 depicts the results of simulation of a 10-unit inference network for 10-state random walk with different parameter setup.

5.5.2 Example 2: Random Walk on a Grid

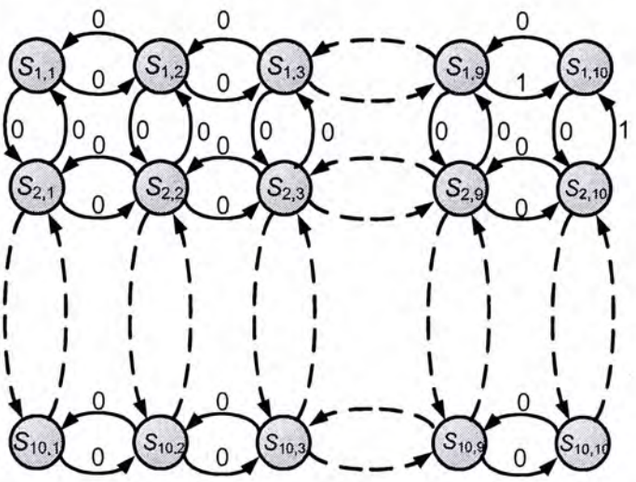


Figure 5.9 A small Markov process for generating random walks. The Markov chain has 100 states in total. The transition probability given $T(i,a_{ik},i')=0.8$ if $k=i'$ and $\sum_{\forall k}T(i,a_{ik},i')=0.2$ if $k\neq i'$, where i is the current state, i' is the next state and a is the action.

Consider random walk on a 100-state with 10 by 10 grid-world. Each state only connects to at most 4 adjacent states, while states at edge connect to three and states at corner connect to two. States are orientated as a perfect square. All transitions would result zero reward except transition to the state at the north-east corner would return reward of one.

Similar to the random walk example, the continuous-time inference network can be modeled by a set of differential equations on the 100 nodes $S_{i,j}$, $i,j = 1,2,\ldots,10$. The expected rewards $V_{i,j}$, $i,j = 1,2,\ldots,10$ evolve as first-order lag controlled by λ. When the network state is close to the optimal solution, presumably, we have the linear Bellman equations. The Bellman equations for 2-node (two adjacent nodes), 3-node, and 4-node:

$$V_{1,1}^{*}=\gamma\left[\rho V_{1,2}^{*}+(1-\rho)V_{2,1}^{*}\right] \tag{5.28}$$

$$V_{1,2}^{*}=\gamma\left[\rho V_{1,3}^{*}+\frac{1-\rho}{2}\left(V_{1,1}^{*}+V_{2,2}^{*}\right)\right] \tag{5.29}$$

$$V_{2,2}^{*}=\gamma\left[\rho V_{2,3}^{*}+\frac{1-\rho}{3}\left(V_{1,2}^{*}+V_{2,1}^{*}+V_{2,3}^{*}\right)\right] \tag{5.30}$$

For those boundary nodes

$$V_{1,n-1}^{*}=\rho+\gamma\left[\rho V_{1,n}^{*}+\frac{1-\rho}{2}\left(V_{1,n-2}^{*}+V_{2,n-1}^{*}\right)\right] \tag{5.31}$$

$$V_{2,n}^{*}=\rho+\gamma\left[\rho V_{1,n}^{*}+\frac{1-\rho}{2}\left(V_{2,n-1}^{*}+V_{3,n}^{*}\right)\right] \tag{5.32}$$

$$V_{1,n}^{*}=\gamma\left[\rho V_{1,n-1}^{*}+(1-\rho)V_{2,n}^{*}\right] \tag{5.33}$$

The network can be modeled by using differential equations. Differential equations for 2-node, 3-node and 4-node:

$$\frac{dV_{1,1}}{dt}=-\lambda V_{1,1}+\lambda\max\begin{cases}\gamma\left[\rho V_{2,1}+(1-\rho)V_{1,2}\right] & \text{for down;}\\ \gamma\left[\rho V_{1,2}+(1-\rho)V_{2,1}\right] & \text{for right.}\end{cases} \tag{5.34}$$

$$\frac{dV_{1,2}}{dt}=-\lambda V_{1,2}+\lambda\max\begin{cases}\gamma\left[\rho V_{1,1}+\frac{(1-\rho)}{2}\left(V_{1,3}+V_{2,2}\right)\right] & \text{for left;}\\ \gamma\left[\rho V_{1,3}+\frac{(1-\rho)}{2}\left(V_{1,1}+V_{2,2}\right)\right] & \text{for right;}\\ \gamma\left[\rho V_{2,2}+\frac{(1-\rho)}{2}\left(V_{1,1}+V_{1,3}\right)\right] & \text{for down.}\end{cases} \tag{5.35}$$

$$\frac{dV_{2,2}}{dt}=-\lambda V_{2,2}+\lambda\max\begin{cases}\gamma\left[\rho V_{2,1}+\frac{(1-\rho)}{3}\left(V_{1,2}+V_{2,3}+V_{3,2}\right)\right] & \text{for left;}\\ \gamma\left[\rho V_{2,3}+\frac{(1-\rho)}{3}\left(V_{1,2}+V_{2,1}+V_{3,2}\right)\right] & \text{for right;}\\ \gamma\left[\rho V_{1,2}+\frac{(1-\rho)}{3}\left(V_{2,1}+V_{2,3}+V_{3,2}\right)\right] & \text{for up;}\\ \gamma\left[\rho V_{3,2}+\frac{(1-\rho)}{3}\left(V_{1,2}+V_{2,3}+V_{2,1}\right)\right] & \text{for down;}\end{cases} \tag{5.36}$$

For those boundary nodes, the differential equations:

$$\frac{dV_{1,n}}{dt}=-\lambda V_{1,n}+\lambda\max\begin{cases}\gamma\left[\rho V_{2,n}+(1-\rho)V_{1,n-1}\right] & \text{for down;}\\ \gamma\left[\rho V_{1,n-1}+(1-\rho)V_{2,n}\right] & \text{for left.}\end{cases} \tag{5.37}$$

$$\frac{dV_{1,n-1}}{dt} = -\lambda V_{1,n-1} + \lambda \max \begin{cases} \dfrac{(1-\rho)}{2} + \gamma\left[\rho V_{1,n-2} + \dfrac{(1-\rho)}{2}\left(V_{1,n} + V_{2,n-1}\right)\right] & \text{for left;} \\ \rho + \gamma\left[\rho V_{1,n} + \dfrac{(1-\rho)}{2}\left(V_{1,n-2} + V_{2,n-1}\right)\right] & \text{for right;} \\ \dfrac{(1-\rho)}{2} + \gamma\left[\rho V_{2,n-1} + \dfrac{(1-\rho)}{2}\left(V_{1,n-2} + V_{2,n-1}\right)\right] & \text{for down.} \end{cases} \quad (5.38)$$

The computational units are arranged and connected to form a square matrix following the topology as shown in Figure 5.9. Consider λ_i=0.5, γ=0.9 we obtained the estimates of the state-value function shown in Figure 5.10. It showed that the parallel computation of the inference network. Value at each grid converges to the optimal solution in a parallel and synchronize way.

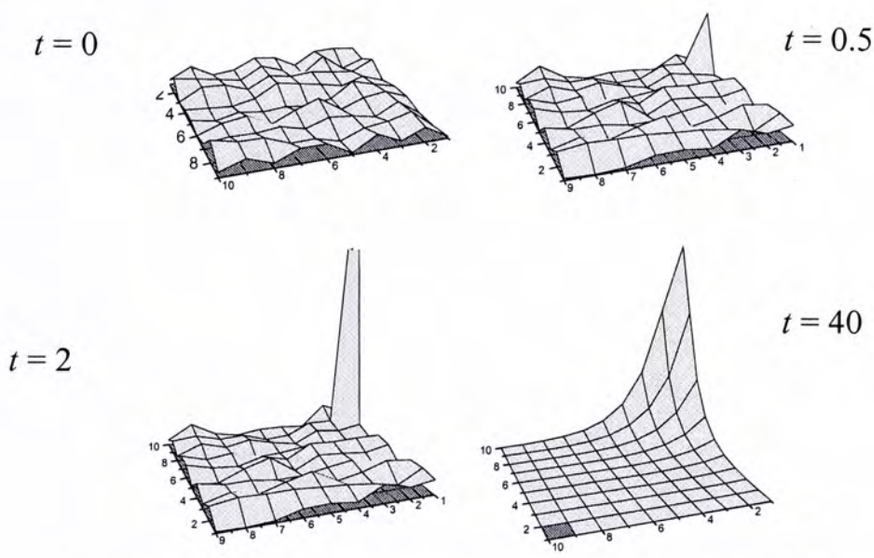


Figure 5.10 Computation of state-value functions for the random walk in the grid-world using the continuous-time inference network with 100 units

Convergence of the inference network in two dimensional grid-world problem is controlled by lambda. When comparing to the previous 10-state random walk problem, the network settles to the desired solution after $t \approx 40$, which is more or less the same as that for the 10-state problem. In addition, by increasing the value of λ, the time needed for the network settled decreased. Also, when λ increases to a large value (says 0.9), the network still converge to the optimal solution. It implies that the network converges to the optimal solution independent on the rate of convergence.

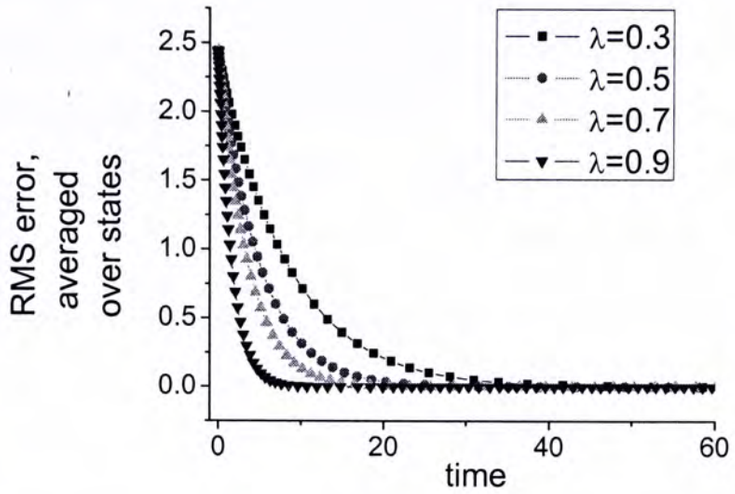


Figure 5.11 Learning curves for continuous-time inference network for 100-state random walk in the grid-world problem, for various values of λ. The performance measure shown is the root mean-square (RMS) error between the value function leaned and the true value function, averaged over 100 states.

5.5.3 Example: 3 Stochastic Shortest Path Problem

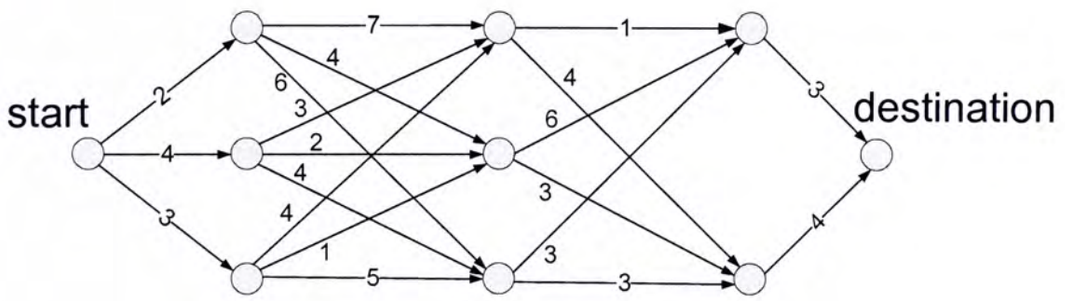


Figure 5.12 The flow-graph for stagecoach problem (Haykin 1999).

We consider a stagecoach problem. A fortune seeker in Missouri decided to go west to join the gold rush in California in the mid-nineteenth century. The journey required traveling by stagecoach through unsettled country, which posed a serious danger of attack by marauders along the way. The starting point of the journey (Missouri) and the destination (California) were fixed, but there was considerable choice as to which other eight states to travel through the route, as shown in Figure 5.12.

There is also cost of life insurance policy for taking any stagecoach run based on a careful evaluation of the safety of that run. The problem is to find the route from the starting point to the

destination with the cheapest insurance policy. To find the optimal route, we consider a continuous-time inference network with 10 units. The units connected following the topology given in the figure. Also the transition probabilities of the problem is given as $T(i, a_{ik}, i') = 0.8$ if $k = i'$ and $\sum_{\forall k} T(i, a_{ik}, i') = 0.2$ if $k \neq i'$, where i is the current state, i' is the next state and a is the action.

While similar convergence to the optimal expected cost at each state are obtained and shown in Figure 5.13. We varied the discounted factor, which signifies the importance of the long term reward when approaches to one. We found that though both cases converges to the optimal expected reward. But the convergence time of the continuous-time inference network decreases as the discounted factor approaches to one.

The discounting factor is significant parameters to the computational load of value iteration and other dynamic programming. For value iteration, the number of iterations required to obtain the optimal value increases exponentially when increases the discounting factor (Kealbling, Littman et al. 1996). The continuous-time network suffers more or less the computational burden of the discounting factor.

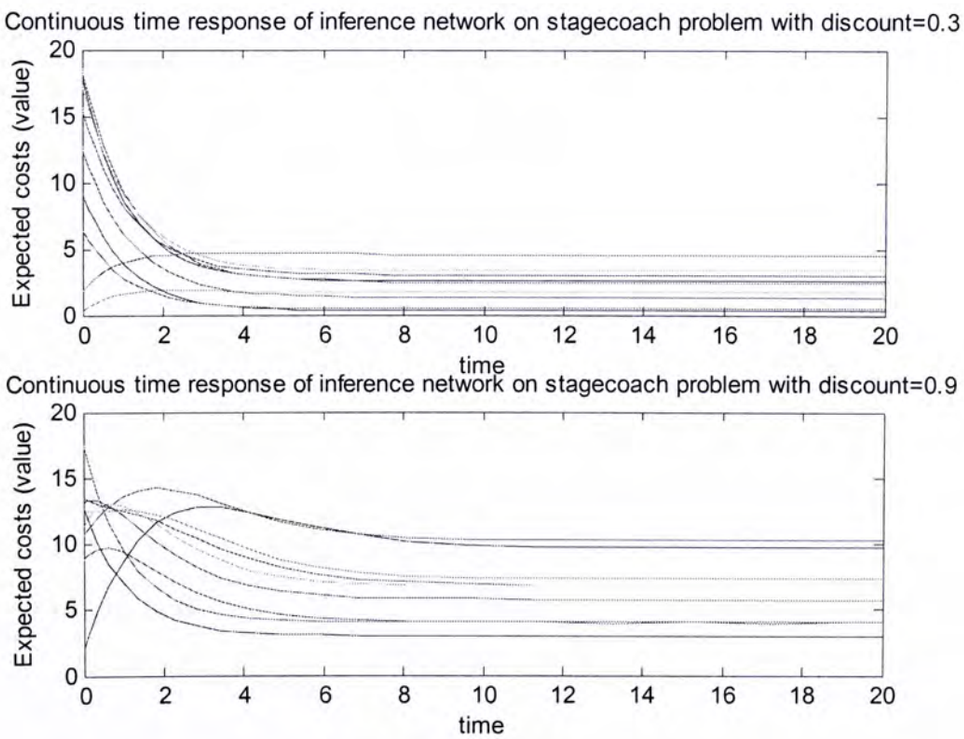


Figure 5.13 Simulation of a inference network for the stagecoach problem (upper: λ_i=0.9, γ=0.3, below: λ_i=0.9, γ=0.9,).

5.5.4 Relationship between λ and γ

Since λ is the open-loop system pole for the computational unit, λ increases leads to the increasing of the network convergence. Considers fixed discount factor γ for a typical random walk problem, we vary λ from 0.1 to 0.9. By measuring the averaged RMS error over all states at a fixed time, convergence time is exponentially decreasing while λ is increasing. By adjusting the discount factor from 0.3 to 0.9, the exponentially decreasing of convergence time still observed but with a slower rate (See Figure 5.14).

Figure 5.15 shows the plot of RMS error against λ and γ at the same time. It showed interesting relationships that RMS error decreases exponentially when λ increases while RMS error increases exponentially when γ increases. The results suggest that given the discount factor approaches to one, using an arbitrarily large λ would result in a fast convergence to compensate.

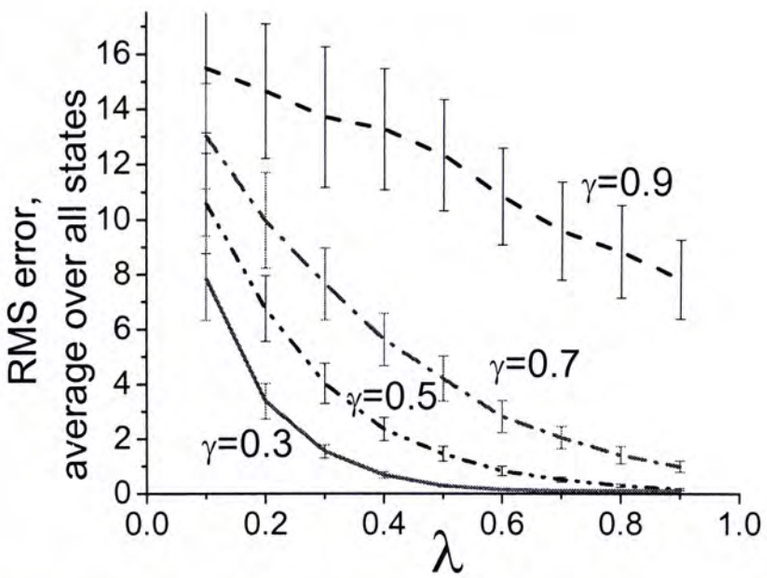


Figure 5.14 Comparison of convergence speed for continuous-time inference network for the random walk problem, for various values of λ and discount factor γ. The performance measure shown is the root mean-square (RMS) error between the value function leaned and the true value function, averaged over all states. It showed that exponentially decreasing of RMS when λ increases. By increasing discount factor γ, it decreases the rate of convergence.

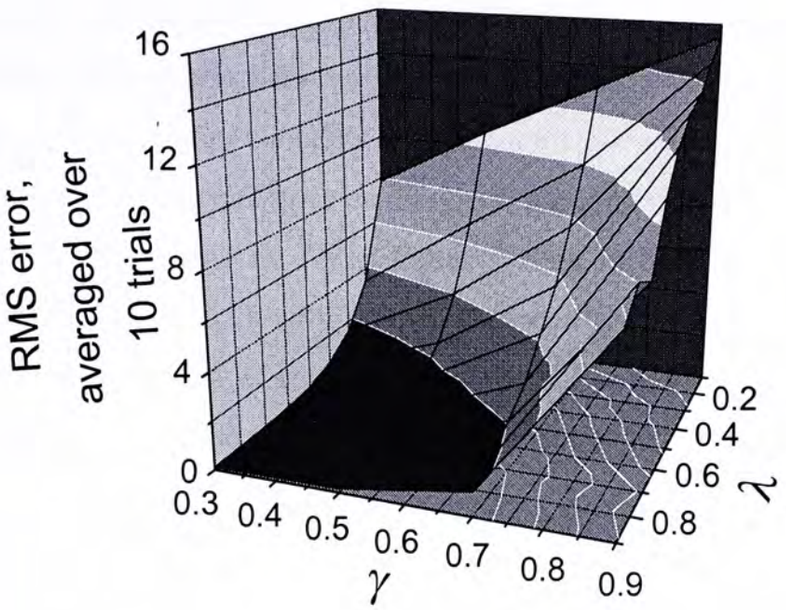


Figure 5.15 Comparison of convergence speed for continuous-time inference network for the random walk problem, for various values of λ and discount factor γ on an alternative view. Interesting relationships between λ and γ can be found. The RMS error shows relatively symmetric between large λ, large γ and small λ, small γ. Long convergence time for large γ (discount factor) and be compensated by using large λ in the continuous-time inference network.

5.6 Discrete-Time Inference Network

Modified Eq. ((5.13)-(5.14)) that consider δt is an arbitrarily small time interval. We have Eq. (5.39) as the differences equation for the output of each computational unit that is derived from the differential equations Eq. ((5.13)-(5.14)).

$$g_{t+\delta t}(i) - g_t(i) = \left\{ -\lambda_i g_t(i) + \lambda_t \min_{\forall a} \left\{ R(i,i') + \sum_{i' \in S} T(i,a,i') g_t(i') \right\} \right\} \cdot \delta t \qquad (5.39)$$

For simplicity, considered that δt equals to one, we have

$$g_{t+1}(i) = (1-\lambda_i) g_t(i) + \lambda_t \min_{\forall a} \left\{ R(i,i') + \sum_{i' \in S} T(i,a,i') g_t(i') \right\} \qquad (5.40)$$

and we can consider an arbitrarily fast system that λ_i equals to one for all i.

$$g_{t+1}(i) = \min_{\forall a} \left\{ R(i,i') + \sum_{i' \in S} T(i,a,i') g_t(i') \right\} \qquad (5.41)$$

Noticed that Eq. (5.41) has a similar form as the Bellman optimality equation Eq. (5.3), but in Eq. (5.41) discrete time relationships between the outputs of each computational unit are defined by t. Supposed that there is a memory elements at each computational unit that the output $g(i)$ at time t is registered, the discrete-time inference can be readily implemented by using digital logics.

In discrete time system, such as digital logics system or FPGAs, the time variable t can be implemented using a system clock with known or controllable frequency. Each clock tick can be considered as t+1. But in most of the time, it is not sufficient for only one clock tick to have finished all computation from Eq.(5.41). So, several clock ticks can be considered for t+1.

5.6.1 Results

The architecture can be readily realized using digital FPGAs. The convergence of output of computational units based on the FPGAs implementation is compared against the software[2] implementation. Figure 5.16 shows the results. From the figure, "*" represents the software results while "-" represents FPGAs results. Both software and FPGAs network converge to the optimal value within 10 iterations. The iteration in software refers that all states and all actions has been gone through exactly once. However, in the FPGA distributed implementation, all states value is updated simultaneously.

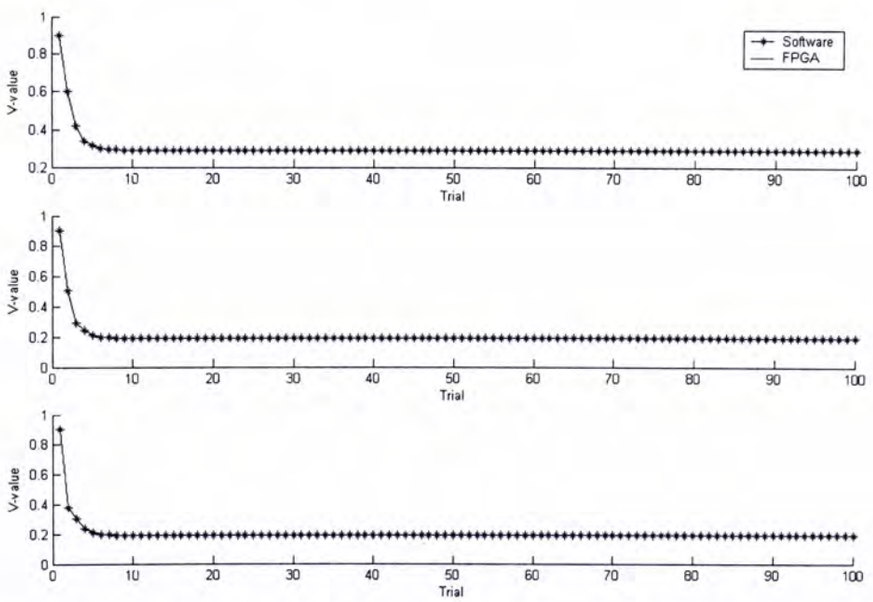


Figure 5.16 Verification of the network convergence on FPGAs implementation with referring to the software computation.

[2] Matlab is used as software programming

It is interesting to find out the hardware computation error when using software results as reference. The multiplication in hardware would be the major source of error. We have the absolute error against iteration loops in the figure 5.17. The error is small as in in the order of 10^{-6}, when 16-bit fixed-point is used. Also, it shows that when the network converges, the hardware would have less error.

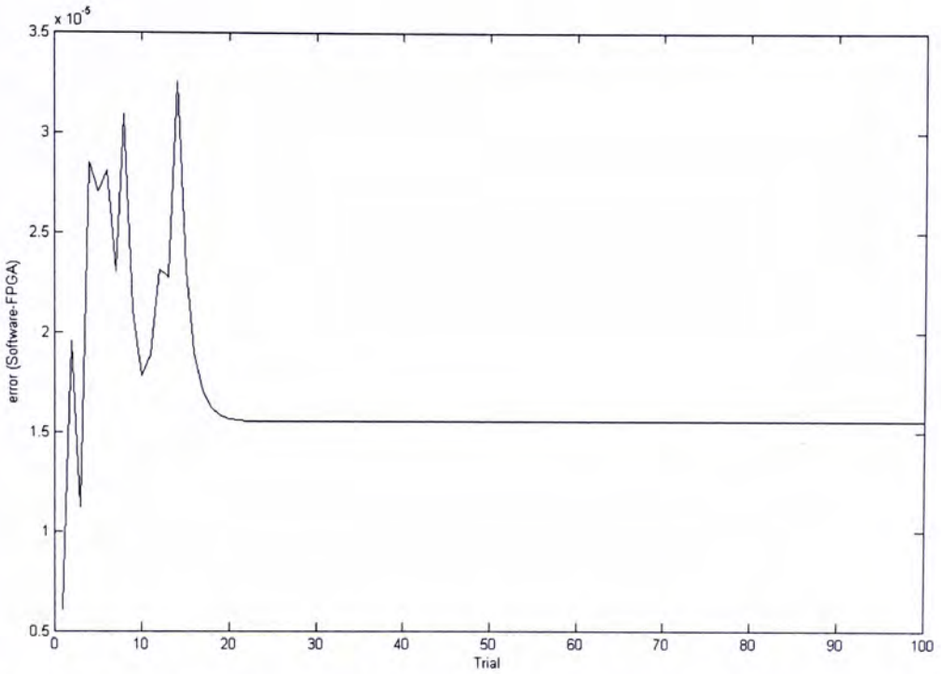


Figure 5.17 Bit truncation error in the 16-bit FPGA inference network computation

5.7 Conclusion

This chapter describes a novel connectionist approach for mapping the value iteration algorithm to a continuous-time inference network solving the dynamic programming in the continuous-time domain. Some of the limitations for the conventional heuristic search techniques, such as increased inefficiency for problem with large m, and the discrete time asynchronous distributed approach with requirement for the synchronization of control flags between processors, exponentially increasing of computational loads when discount factor approaches one, have been effectively overcome. Numerical simulations have been used to demonstrate the salient features of such type of continuous-time inference network, for which fully-analog parallel implementation can be made arbitrarily fast and its required computational time is practically independent of m. We also derived a discrete-time version of the inference network that the network can be well mapped to FPGAs for solving dynamic programming. This opens up the possibility for future realization of this class of optimization and learning circuits for real-

time decision making applications, such as rover path planning and trajectory design for UAVs for surveillance mission.

Chapter 6

On Distributed *Q*-Learning Network

In a Markov decision process, knowledge of the state transition probability function $T(s,a,s')$ and the reinforcement reward $R(s,a)$ is not always available. The agent must interact with its environment directly to obtain information, which by means of an appropriate algorithm, can be processed to produce optimal decisions. To circumvent the limitation of the dynamic programming, Q-learning, proposed by Watkins, as a simple yet striking powerful learning algorithm contributes the on-line reinforcement learning with assuming the probability and rewards are unknown. However, the slow sequential learning process under the conventional Q-learning algorithm would have delayed the solution time for many of the time-critical and real-time decision making problems. Based on the framework of the Bellman Inference Network, a novel Q-learning network architecture is proposed. We introduce the exploration and memory components to the original Bellman inference network to form a connectionist network with together the capability of parallel learning and optimization. We found that the Q-learning network outperforms significantly the conventional Q-learning algorithm under a distributed unknown environment. We also proposed two design alternatives for the realization of Q-learning network using Field Programmable Gate Array (FPGA) that the network would be dedicated to fit different applications.

6.1 Introduction

Dynamic programming can efficiently solve reinforcement learning problem, as optimal decisions can be found given prior information about the environment, such as rewards and probability distribution. The policy iteration or value iteration dynamic programming algorithm requires prior knowledge of the underlying Markov decision process. That is, for the computation of an optimal policy to be feasible, we require that the state-transition probabilities and the observed rewards to be known. However, considering the fact that prior information is not always available and the cost to obtain such information is exceptionally high under certain circumstances, the dynamic programming approach may not always be feasible or easily being adopted (Haykin 1999).

To circumvent the limitation of the dynamic programming approach, learning from on-line experience requiring no prior knowledge of the environment's dynamic would be striking (Sutton and Barto 1998). A simple yet powerful algorithm, called Q-learning, has been proposed by Wilkins (Watkins 1989). Q-learning uses simulation or experimental information to compute estimates of the expected value as a function of the initial state. It is an incremental dynamic programming procedure that determines the optimal policy in a step-by-step manner. It is highly suited for solving Markovian decision problems without explicit knowledge of the transition probabilities. It was found to be an effective endeavor with capability to deal with the learning under situations with incomplete information. It allows the learning agent to explore the environment by making trials, while performing optimization to find the best decision at the same time.

Q-learning has been extensively studied and applied across a broad domain, such as control and decision making. However, the original Q-learning formulation only considers a single learning agent exploring the environment and making a single decision at a time (Sutton and Barto 1998). Many interesting problems that have properties favoring the use of distributed solution can be solved by reinforcement learning. Whenever the state and action space is large, a distributed approach to perform the computation is desirable because it makes computational speedups from parallelism possible. In general, distributed learning could not only release natural constraints like the limited processing power of a single agent or the geographical distribution of data but also benefit from the inherent properties of distributed systems like robustness,

parallelism and scalability. Although Q-learning is fairly well understood, distributed reinforcement learning is a much less mature concept (Tsitsiklis 1994; Weib 1995; Schneider, Wong et al. 1999).

Theoretical study of the asynchronous mathematical model of Q-learning, developed by Tsitsiklis (Tsitsiklis 1994), forms the basis of parallel and distributed realization of Q-learning. He proposed a mathematical model to analyze and prove the convergence of Q-learning under asynchronous updating and delayed information. Though with the elegant theoretical analysis, there has been few realization and implementation for the distributed Q-learning until recent years. The needs for distributed approach emerge along with the rapid growth of internet and communication network technologies.

For example, the problem of routing packets efficiently in a communication network with an irregular topology and unpredictable usage patterns can readily adopt a distributed reinforcement learning model (Littman and Boyan). The Q-learning can adapt to changes in network traffic and is constructed from a distributed collection of learners, each of which is responsible for a partition of the problem. The distributed approach appears to be very effective in routing packets under high load. The other approaches with similar idea of distributed computational learning also apply on the network and communication systems (Caro and Dorigo 1998) to obtain higher throughput.

In contrast to computer network or communication problems, real-time decision making applications such as autonomous robots and UAVs, requires efficient computation that the software approach is not sufficiently fast enough to provide (Cauwenberghs and Bayoumi 1999; Williams, Kim et al. 2001; Jun and D'Andrea 2003). VLSI hardware allows efficient high speed on-chip communication between processing units which provides powerful collective computation and hardware realization. However, learning algorithms that are efficiently implemented on general-purpose digital computers do not necessarily map efficiently onto VLSI hardware (Cauwenberghs 1997). There are different design constraints imposed by the hardware VLSI that are absent in software implementation.

In (Cauwenberghs 1997), Cauwenberghs proposed a hardware perturbative model for model-free gradient decent implementation that reinforcement learning can be mapped onto analog VLSI based on the hardware model. The architectural VLSI design retains desirable properties of a modular and cellular structure, model-free distributed representation and robustness to noise

and mismatches in the implementation. The hardware architecture demonstrates the feasibility and effectiveness using silicon solution for real-time learning and decision making. In line with Cauwenberghs' work, the hardware architecture can be further extended and modified to handle more complicated and effective decision making model.

The connectionist architecture proposed by Sutton in the 80's, called the Critic and Actor architecture, first demonstrated the capability of reinforcement and exploration learning mechanism and had shown that the model could solve difficult control task (Barto, Sutton et al. 1983). The first remarkable success of the proposed algorithm, Critic and Actor architecture, is now regarded as a rather sophisticated approach, in which the performance might be limited by its high computational complexity (Kealbling, Littman et al. 1996). On the other hand, *Q*-learning emerges as an effective alternative because of its simplicity and effectiveness. Previous implementation of the perturbative model realized the Critic and Actor model based on a connectionist approach (Cauwenberghs 1997). However, in spite of the compact architecture, the implementation would inherit the limitation of the computational complication. Alternatively, realization of the *Q*-learning would inherit the compactness and efficient architecture which might improve from the original Critic and Actor architecture.

In the previous chapter, a distributed continuous-time inference network for dynamic programming problems has been proposed. It has shown that the inference network converges to the optimal Bellman optimality condition, for which the convergence rate can be made arbitrarily fast and is practically independent of the discounting factor and the number of states. Further, an analog VLSI CMOS circuit has been developed to realize the inference network and to demonstrate the inference network optimization for the dynamic programming problem. By adopting the collective computational paradigm of the inference network, in addition to the original architecture, we introduce the component of stochastic exploration. It would enable the learning capability of the connectionist network beside the optimization capability. We consider each computational unit as a learning unit, in which the architecture is defined by the *Q*-learning formulation and the stochastic elements for exploration.

In this chapter, we propose a novel architecture of distributed *Q*-learning network. In line with the spirit of the Binary relation inference network (BRIN), a collective connectionist network approach could solve the reinforcement learning problem by introducing the component of stochastic exploration. The *Q*-learning network would be shown efficient to solve benchmark

Markov decision process problem with unknown prior transition probability distribution and rewards. Also realization of the Q-learning network using Field Programmable Gate Array (FPGA) will be discussed with alternatives design approach compared.

6.2 Distributed *Q*-learning Network

The behavioral task of the reinforcement learning system is how to find an optimal policy after trying out various possible sequences of actions, and observing the costs incurred and the state transitions that occur. Q-learning defines a procedure of taking sample of the environment and associating the action with the observed reward. The formulation follows a well-known Bellman optimality equation that Q-learning use a simple sample to replacing the probability distribution from the original Bellman equation. In other word, Q-learning can be considered as a sampling approach that the learner directly taking samples from the environment instead of assuming a probability distribution is given.

Bellman equation defines the optimal criterion for the Markov decision process, such that the expected reward $V(s)$ at each state s can be found by applying the Bellman equation iteratively. Recalled that the optimal decision $\pi^*(s)$ at state s can be computed based on the expected reward $V(s)$. It seems that the decision variable π and expected reward V is two separated concepts while they are related due to Eq. 5.4. Along the introduction of the concept *state-action pair* (s, a), the value or the expected reward for taking action a at state s is quantitatively defined by the term $Q(s, a)$, or called Q-factor (Watkins 1989). Function $Q(s, a)$ is used to memorize the expected reward for the action a and the state s. Thus, there are direct measures of all possible actions at all states, while the best decision can be found simply the $Q(s, a_i)$ with highest score for all i. Evaluation of Q-factor would benefit the reinforcement learning approach, such that the complicated evaluation of decision π from V is not a necessity.

Let $Q^*(s, a)$ be the expected discounted reinforcement of taking action a in state s, then continuing by choosing action optimally. Note that $V^*(s)$ is the value of s assuming that best action is taken initially, so $V^*(s) = \min_a Q^*(s,a)$. Following Bellman equation, $Q^*(s,a)$ can hence be written recursively as

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \min_{\forall a'} Q^*(s',a') \tag{6.1}$$

Note that, since $V^*(s) = \min_a Q^*(s,a)$, we have $\pi^*(s) = \arg\min_a Q^*(s,a)$ as an optimal policy.

Because the Q function makes the action explicit, we can estimate the Q values on-line by taking samples of rewards from the environment. Following (Kealbling, Littman et al. 1996), we define $\langle s,a,r,s' \rangle$, which is called *experience tuple* summarizing a single transition in the environment. Here s is the agent's state before the transition, a is its choice of action, r the instantaneous reward it receives, and s' its resulting state. The Q-factor is learned using the Q-learning (Watkins and Dayan 1992) which uses the updating rule

$$Q(s,a) = Q(s,a) + \alpha \left(r + \gamma \min_{\forall a'} Q(s',a') - Q(s,a) \right) \tag{6.2}$$

$$V(s) = \min_{\forall a} Q(s,a) \tag{6.3}$$

where γ is the discount factor. Also, it has been shown that if each action is executed in each state an infinite number of times on an infinite run and α is decayed appropriately, the Q value will converge with probability 1 to Q^* (Watkins 1989; Watkins and Dayan 1992; Tsitsiklis 1994).

6.2.1 Distributed *Q*-learning network

As a parallel extension, sampling the environment can be executed in parallel or in a distributed process. It is no necessarily to keep trying each action in a sequential manner, given that it is feasible to the environment. In other words, the sampling and updating procedure of the Q-learning can be achieved in parallel by independent units, which represent agents at different states. In (Tsitsiklis 1994), Tsitsiklis showed that Q-learning would converge to the optimal solution under asynchronous updating and sampling. It provided a solid mathematical foundation for the formulation of the distributed Q-learning network.

In previous chapter, we have shown that the optimal policy or the optimal solution based on Bellman optimality criterion can be computed using a continuous-time inference network architecture. In the inference network, each computational unit represents a binary relation

between the corresponding state and the destination (or goal) state. The Bellman equation can be mapping to the site functions and unit function in the computational unit, and the network converges to the optimal solution. It has been proved that the energy function of the network is decreasing and will converge to the optimal solution.

In general, the Bellman inference network is performing optimization based on the Bellman optimality criterion. However, probability distribution and rewards are simply assumed to be known in prior. Following the idea of Q-learning, we can introduce the $Q(s, a)$ function into the Bellman inference network to memorize the expected reward for the action a and the state s. Also, stochastic exploration mechanism can be introduced in the network, so that the computational unit can interact with the environment to obtain information about the reward and state transition.

Q-learning algorithm is less computational intensive than the Bellman dynamic programming, as the intensive probabilistic expectation computation is omitted. However, as samplings from the environment, elements of memories are required in extra, in contrast, which is not necessary in Bellman inference network.

A Q-learning network is formed by interconnection of self contained computational units. Figure 6.1 shows the structure of a unit and the connections in a general learning network. Each unit is to represent a binary relation between the state i and the destination state d. In most cases, destination state d is simply omitted and implicitly assumed. So for each unit, which is denoted as $L(i)$ for state i. Each unit, there are N inputs for carrying out the Q-learning updating. For each epoch, only one input is selected, based on the state transition s' feedback from the environment, to execute the updating. The value function $S(\cdot)$ of the corresponding relation for state I is then determined by resolving the conflict among all of the $Q(s, a_i)$, for all i.

In addition, there is an interaction between the computational unit and the environment to be learnt. The unit $L(i)$ will generate an action a to the environment, which represents the agent (or learner) at state i taking action a. Consequently, reward r and state transition result s' would be received after taking action a. There is a close analogue between the learning network and the learning agent. Considered that there is a stationary agent at each of the state in the original problem graph. The agent keeps trying different actions at the same time and updating the corresponding Q-factor. At the mean time, the inference network is performing optimization via the interconnected computational unit architecture.

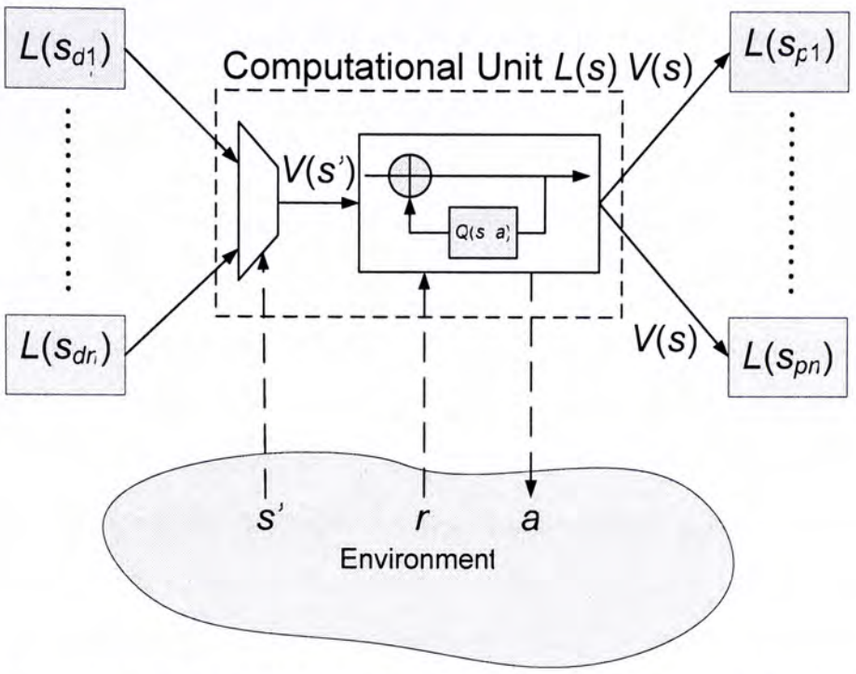


Figure 6.1 A typical computational unit in the learning network depicts the input and output of the learning network architecture. For computational unit L(s), which represents the binary relation between state s and the destination state, inputs the value V(s') from the decedent units. However, in contrast to the Bellman network, s' is determined by the environment. Therefore, all possible decedent units output V(s') to unit L(s), and is being selected by the environment. Within the unit L(s), Q-factor for all possible action are stored and being updated accordingly. The unit L(s) outputs the action, and inputs the reward for the Q-learning rule updating. The unit also outputs the V(s) to the predecessor units.

6.2.2 *Q*-learning network architecture

Considered a specific unit $L(s)$, the output is denoted $p(s)$, where each unit only has one and only one output function. Each unit has component of state exploration Ψ_1, Q-factor table Ψ_2, and exploitation logics Ψ_3.

To choose an action a in a state s, the exploration component Ψ_1 uses the action-value estimations $Q(s, a)$ and an exploration strategy called σ. The selected action is then:

$$a = \sigma(s, Q) \tag{6.4}$$

Following (Sutton and Barto 1998; Haykin 1999), σ is the ε-greedy searching strategy: most of the time, the greedy action is selected (the action for which $Q(s, a)$ is maximum) and sometimes,

a random action is selected with a small probability ε, independently of the action-value estimations. This strategy allows us to control exploration rate. The component Ψ_1 maps states to actions, is fully defined by the action-value function Q in addition with the exploration strategy σ.

The state exploration component requires the Q-factor for all possible actions. Therefore, we have the other component Q-factor table to store the Q values. This is one of the major different between the previously introduced Bellman inference network. In the inference network, there is no memory component, as inputs such as probabilities and rewards are assuming constantly supply. The Q-factor table is needed for updating the Q-factor by the Q-learning rule. A minimum operator is following the outputs of the Q-factor table. It summarizes the Q-factor values and output the minimum value, as the expected reward $V(s)$ from the Eqs.((6.2)-(6.3)).

The last component is the exploitation, which realizes the Q-learning rule. Given that the Q-factor table with the minimum operator, we have the expected reward of the state, $V(s)$. Then the Q-learning rule can be modified accordingly for the architecture as follow

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left[r + \gamma V(s)\right] \tag{6.5}$$

where α is the learning rate, s is the state, a is the action generated from Ψ_1, r is the reward received from the environment and $V(s)$ is the expected reward from the minimum operator.

Figure 6.2 shows the architecture of a Q-learning computational unit $L(s)$ with all components. The unit inputs a set of $V(s')$ from other m computational units, which are the adjacent states of state s. The output a, input s' and r are action, next state and reward respectively interacting with the environment. The $V(s)$ is the output of the unit to other units.

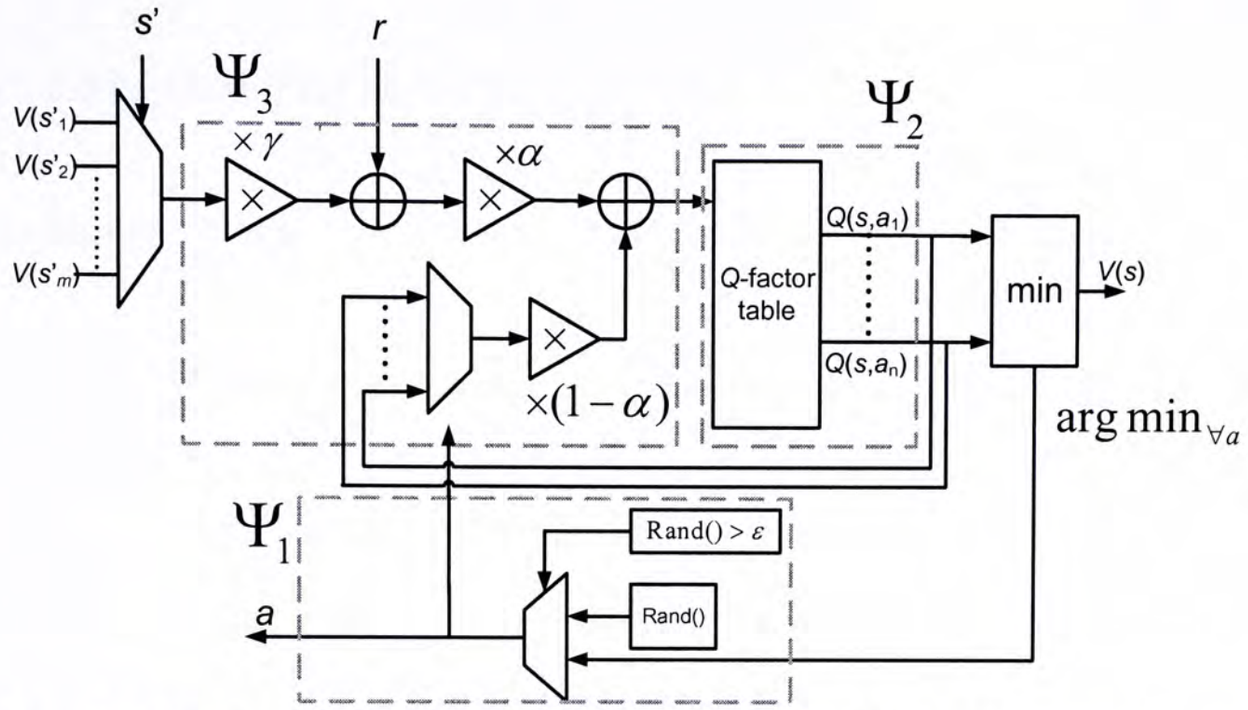


Figure 6.2 Schematic of a typical computational unit, in which consists of three major components, 1.) state exploration Ψ_1 ,for determining the action a to the environment; where rand() represents a random number between zero and one. 2.) Q-factor table Ψ_2 , for storing the Q-factor and 3.) exploitation logics Ψ_3 , for updating the Q-factor. There is also a minimum operator following the outputs of the Q-factor table. It is used for summarizing the state-action pairs and finding the best expected reward.

A distributed learning network approach for solving the reinforcement learning problem can be formulated as follows:

1. Construct a *m*-unit *Q*-learning network, where the architecture of each computational unit is defined by a set of equations Eqs.((2.4)-(2.5)).
2. The network has a complexity but regular interconnection structure following the topological structure of the given problem or graph. Each unit $L(i)$ sends its output $V(i)$ to a set of neighborhood units, which is the decedent state in the original problem graph. Each unit received at the same time the outputs from the set of neighborhood units, which can be reached from the state *i* taking proper action.
3. Each unit $L(i)$, interacts with the environment with regarding the variables, *a*, *r* and *s'*, where *a* is the action, *r* is the reward received and *s'* is the next state returned from the environment.

6.3 Experimental Results

6.3.1 Random Walk

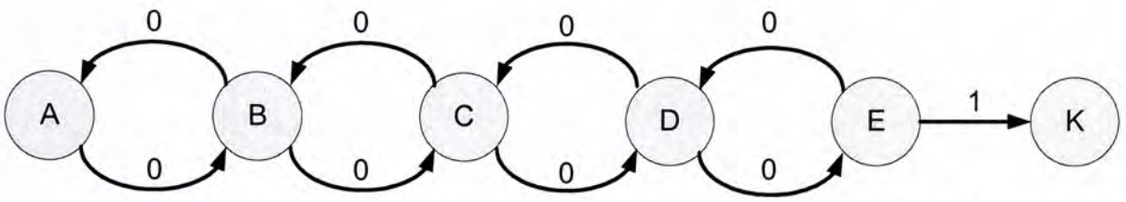


Figure 6.3 A small Markov process for generating random walks. The chain has 5 states in total, while the state 'K' is the terminal state. Transitions would result zero reward except the "right" action at state E would result reward of one, as shown in the arcs in the figure. However, unlike in the Bellman dynamic programming, the probabilities and the rewards are unknown to the learner or agent. Distributed learning network takes samples, such as rewards and state samples, from the environment by issuing actions to the environment.

Random walk problem can be solved efficiently using distributed *Q*-learning network approach. Consider a 6-state random walk problem, that the states are labeled by "{A, B, C, D, E, K}, where state "*K*" is the terminal state. Since the Markov decision process environment is unknown to the learner at the beginning. The underlying transition probability distribution of the environment is defined as follows. For state I, the available actions is denoted by $A_i = \{a_{ik}\}$, where the second subscript k in action a_{ik} taken by the learning system indicates the availability of more than one possible action when the environment is in state i and state k is the desirable next state. The transition of the environment from the state i to the new state j, for example, due to action a_{ik} is probabilistic in nature. The transition probability of the environment from state i to state j by taking action a_{ik} be defined as $T(i, a_{ik}, j)$. In our experiment, probability $T(i, a_{ik}, i') = 0.9$ if $k = i'$ and $\sum_{\forall k} T(i, a_{ik}, i') = 0.1$ if $k \neq i'$, where i is the current state, i' is the next state and a_{ik} is the action with k is the desirable next state.

One computational unit is used to represent one state whereas the terminal state "*K*" can be omitted. Each computational unit would interact with the environment independently by issuing an action and receiving the reward and state transition information. There is intensive communication between the units. Interconnection between units follows the original graph from

the random walk problem to form a learning network. The detailed of the network architecture is presented at Figure 6.4.

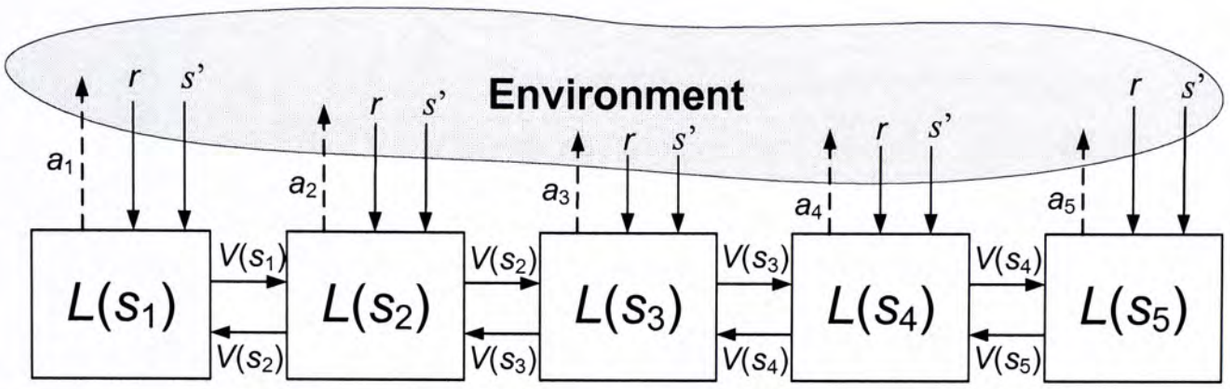


Figure 6.4 A distributed Q-learning framework to solve the random walk problem. The computational units, $L(s_1)$, $L(s_2)$, $L(s_3)$, $L(s_4)$ and $L(s_5)$ are corresponding to state A, B, C, D and E, forms a learning network. Each computational unit issue an action to the environment, and reward and next state are received. There is intensive communication between computation units for updating the expected reward of the adjacent states.

Figure 6.5 shows the results generated from the *Q*-learning network with learning rate α equals to 0.2 (upper) and 0.5 (lower). The network converges to the optimal expected reward, which is the Bellman optimality criterion, for both cases. Also, it can be found that perturbation around the optimal solution appeared after the network had converged. It is because the stochastic exploration component in the computational unit trying to search better optimal solution. For the case of larger learning rate, more vigorous perturbation is found. Higher convergence rate is found for the larger learning rate, though a vivid perturbation at the optimal solution.

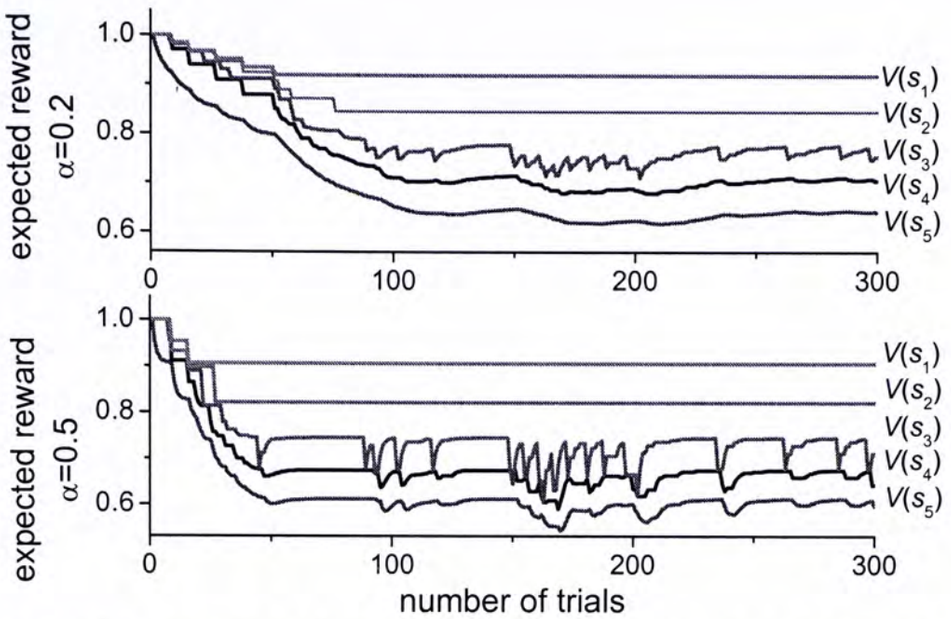


Figure 6.5 Learning curve of the distributed Q-learning network, the (upper) graph is with learning rate α equals to 0.2 and the (lower) graph is with learning rate α equals to 0.5. In both cases, the learning network converges to the optimal solution, while small perturbations around the optimal solution are found after converged. It is because the introduction stochastic exploration at each computational unit, the network is still looking for possible better solution. The one with larger learning rate has larger perturbation. The learning rate serves as step size, as the larger learning rate will result larger perturbation.

6.3.2 The Shortest Path Problem

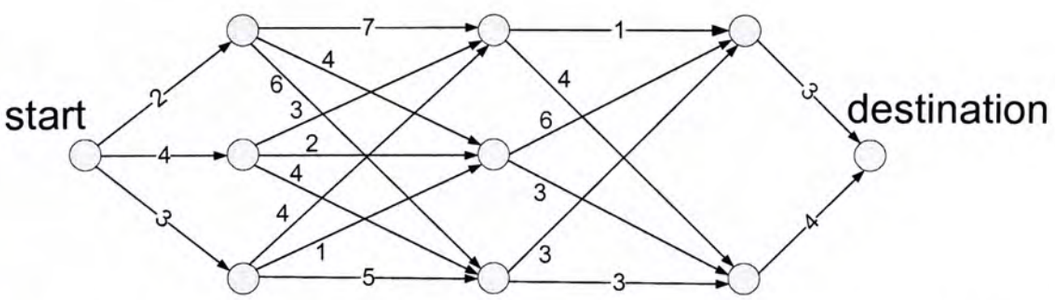


Figure 6.6 The flow-graph for stagecoach problem(Haykin 1999). In this case, the transition probability distribution and the rewards are unknown to the traveler or learning agent. The information can be obtained by experiencing, which is taking actions and receiving rewards and observing state transitions.

We consider a stagecoach problem. A fortune seeker in Missouri decided to go west to join the gold rush in California in the mid-nineteenth century. The journey required traveling by

stagecoach through unsettled country, which posed a serious danger of attack by marauders along the way. The starting point of the journey (Missouri) and the destination (California) were fixed, but there was considerable choice as to which other eight states to travel through the route, as shown in Figure 6.6.

There is also cost of life insurance policy for taking any stagecoach run based on a careful evaluation of the safety of that run. The problem is to find the route from the starting point to the destination with the cheapest insurance policy. Also, the transition probabilities and the reward of the environment are unknown. To find the optimal route, we consider a Q-learning network with 10 units. The units connected following the topology given in the figure. Also the transition probabilities of the problem is given as $T(i, a_{ik}, i') = 0.8$ if $k = i'$ and $\sum_{\forall k} T(i, a_{ik}, i') = 0.2$ if $k \neq i'$, where i is the current state, i' is the next state and a_{ik} is the action with k is the desirable next state.

Forming a network of nine computational units, and each of the units communicates with the environment for taking reward samples. It is assumed that the environment is able to respond to the learning network with different independent actions. The unit for the destination can be omitted, as the expected reward at the destination state is simply assumed to be zero.

Comparing to the typical single agent Q-learning algorithm, the Q-learning network converged at around the 30-th epoch, while the typical Q-learning algorithm converged at around 600-th epoch (See Figure 6.7). The learning network approach is at around 20 times faster than the conventional approach. This result demonstrates the high performance of collective network approach for learning and optimization. Further, vivid perturbations around the optimal solution are found after the learning network converged. This can be attributed to the large learning rate, $\alpha = 0.5$, in this case. Considered a smaller learning rate, $\alpha = 0.1$, a much smoother learning curve was found with less perturbation (See Figure 6.8).

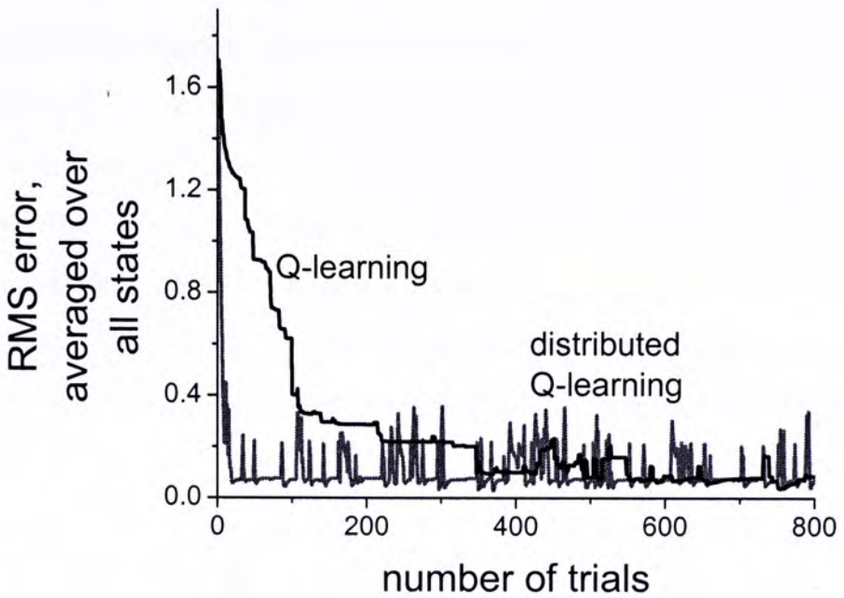


Figure 6.7 Comparison between typical Q-learning and the distributed Q-learning network approach. The graph shows the Root-mean-square (RMS) error with averaged over all states against the number of trials. For one trial, in the Q-learning case, the agent makes one decision whereas in the distributed Q-learning case, all computational units issue one action. It can be observed that the distributed Q-learning approach converges much faster (at around 30 trials) while Q-learning approach converges at around 600 trials. The sharp perturbative peaks from the distributed Q-learning curve can be explained by the large learning rate (i.e. α =0.5).

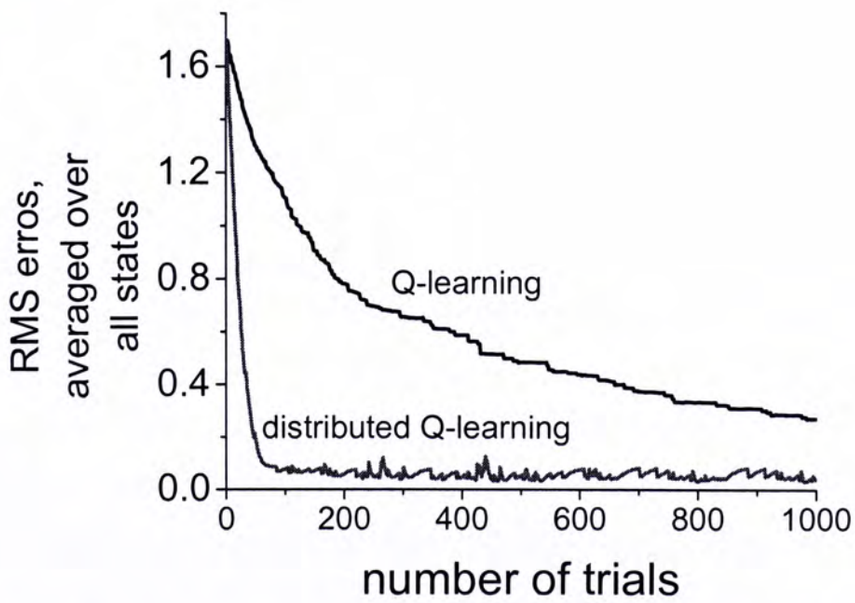


Figure 6.8 This is the experiment for comparison of convergence between Q-learning and distributed Q-learning the with smaller learning rate (i.e. α =0.1). It shows that the distributed Q-learning network converges much faster than typical Q-learning. Also the perturbation because of the learning rate becomes smaller.

Learning rate is critical in affecting the convergence rate of the *Q*-learning network. We examined network for convergence at different learning rate defined. Figure 6.9 shows the results of the experiment. It shows that the network converges to the optimal solution for different learning rate. Also, for larger learning rate, the network converges faster. But, perturbation at the optimal solution is expected for network with larger learning rate. This can be explained by the stochastic exploration energy of the network is larger with larger learning rate. One of the modification can be made is that to assign a decreasing function for the learning rate parameter, such that the network will stop perturbation after it has converged to the optimal result.

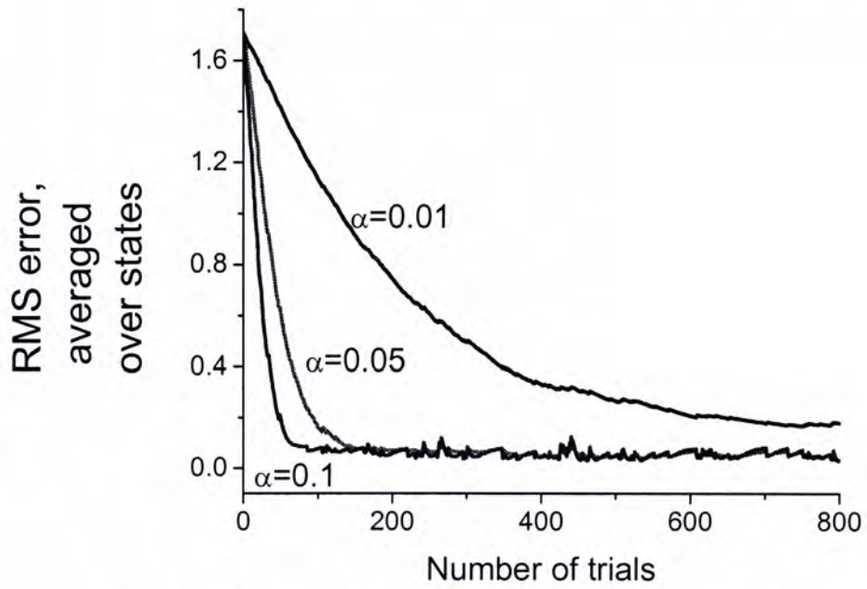


Figure 6.9 the convergence of the distributed Q-learning network can be varied with different learning rate. The larger learning rate would result a faster convergence and a vigorous perturbation around the optimal solution. The one with smaller learning rate would result a slower but smooth convergence.

One of the interesting observations for the *Q*-learning network is about the discount factor. Since, in the continuous-time Bellman inference network, discount factor γ, which defines the importance of the longer term reward, would affect the convergence time of the network. It was found that the larger discount factor, the longer convergence time. However, in the case of *Q*-learning network, it was found that the discount factor γ plays no significant in the convergence time of the network. Figure 6.10 shows the learning curve of the network with difference discount factor. There is no conclusive observation can be found in the figure that there is no

correlation between the convergence time and the discount factor. One of the possible explanation is that the Q-learning is a sampling approach, which the stochastic exploration of the network overwhelms the effect of discount factor in the optimization.

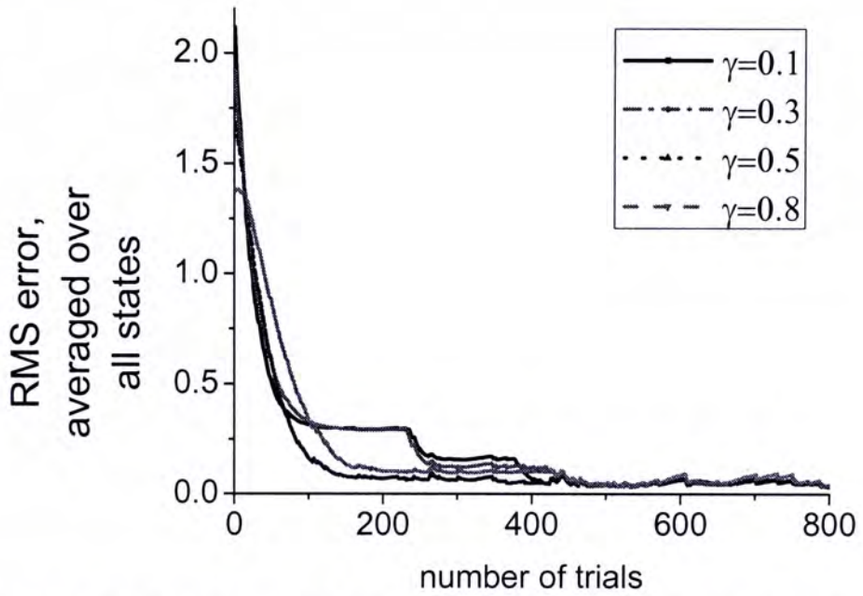


Figure 6.10 For the Bellman inference network, the network needs longer convergence time for a larger discount factor γ. However, for the distributed Q-learning network, there are no big differences in the convergence time for different discount factors.

6.4 Discussion

Distributed Q-learning network demonstrates the high performance of learning a Markov decision process via a collective mechanism with a network of interconnected computational units. Results show that the Q-learning network algorithm outperforms the conventional Q-learning implementation in term of learning speed. The design takes the advantage of given the distributive characteristic of environment. The Q-learning network can also applied for sequential decision making problems. However, to gain the full capability and the highest performance of the network convergence rate, parallel accessing to the environment would definite boost the throughput of the learning system based on the Q-learning network architecture.

6.4.1 Related work

There are close resembles of distributed reinforcement learning designs available. The architectural designs found are typically dedicating for specific needs and can be categorized into three application domains, which are 1.) multi-agents reinforcement learning, 2.) computer network and communication application and 3.) autonomous robot path planning.

The area of multi-agent reinforcement learning is growing rapidly in recent years and appearing as an interdisciplinary subject attracts attention of researchers from the domains of computer science, economic (Kealbling, Littman et al. 1996; Shoham, Powers et al. 2004). Agents in the multi-agent system know little about other agents, and the environment changes during learning. The framework adopted is stochastic games (also called Markov games) (Filar and Vrieze 1997), which are the generalization of the Markov decision process to the case of two or more controllers. The stochastic game is thought to be a close resemble of Nash-type games, that a Nash equilibrium, each agent's choice is the best response to the other agents' choices. Thus, no agent can gain by unilateral deviation. The principle of the multi-agents reinforcement learning is interestingly contributes to the design of artificial intelligence system with applying basic principle from Economic. However, it is still controversial the clarity on the formulation of the basic multi-agents systems and the potential of the design on real world applications (Shoham, Powers et al. 2004).

The application of distributed reinforcement learning in network adaptive routing problem abound. The approaches and the degree of modifications from the original Q-learning scheme are varied. Peshkin and Savova (Peshkin and Savova 2002) proposed a multiple agents reinforcement learning architecture for telecommunication systems, where an individual router is an agent which makes its routing decisions according to an individual policy. Their approach allows update the local policies while avoiding the necessity for centralized control or global knowledge of network structure. A rather sophisticated gradient decent learning algorithm is adopted for the agents to find the optimal routing policy. Similarly, Druschel and Chen (Subramanian, Druschel et al. 1997) adopt an approach from ant colonies, that is very similar in spirit of Peshkin's work, for network routing problems. The routing information is periodically updated by "ants", as agents, for minimizing the network loading. A more general and simple distributed value functions approach demonstrate the collective power of the reinforcement learning in a power grids design problem (Schneider, Wong et al. 1999).

The distributed value function approach has some degree of similarity with the distributed *Q*-learning network approach, in term of the formulation of *Q*-learning in distributed context. Yet, the distributed value function approach does not incorporate the component of state exploration, that it is aimed for the purpose of optimization with given transition probabilities and rewards. Further, the *Q*-learning network is realized in VLSI circuit, where high bandwidth processor communication is provided while the distributed value function approach is targeting to the network of general purpose computers with lower communication bandwidth.

Lastly, parallel *Q*-learning architecture is proposed for autonomous robotic application. In (Laurent and Piat 2001), in order to increase the speed of learning regarding a large state space, the robotic controller was designed to parallelize the maximum/minimum operator using hardware. Unlike the *Q*-learning network approach, the single decision and reward is received by the robot in the parallel *Q*-learning approach. The parallel paradigm proposed in (Laurent and Piat 2001) would introduce speed-up and fit for the application of single learning agent or environment for sequential decision making as the typical *Q*-learning algorithm.

6.5 FPGA Implementation

In recent years, Field Programmable Gate Array (FPGA) technology has emerged as a high-speed digital computation platform. The flexibility of the FPGA's programmable logic combined with its high-speed operation potentially allows it to making decision and performing optimization in real time. Additionally, FPGAs can be used to accelerate prototyping of analog hardware models of brain processes by quickly building a simulation platform to study the functional behavior of the proposed model in a much shorter design cycle.

Our logic-level design makes extensive use of Xilinx's System Generator (SG) that works under MATLAB's Simulink environment (Hwang, shirazi et al. 2001). Simulink provides a schematic design environment of logic gate blocks with which to implement the FPGA model. SG then automatically converts the simulink code from the schematic design to a bit stream file to configure the FPGA hardware.

In Recent advance of Field Programmable Gate Arrays technology, Vertex-class FPGA introduced block Read-only-memory (BRAM) and multipliers as embedded elements in the FPGA. The embedded dedicated multipliers and BRAM offer a wide alternative of resources

management from mapping the arithmetic unit into the dedication circuit instead of distributed logics. The dedicated circuits, such as 18-bit by 18-bit multipliers provide much higher performance than multipliers constructed using FPGA logics. Besides, since the *Q*-learning network architecture requires memory storage for the *Q*-factor table, embedded BRAM in FPGA provides excellent design alternative. Instead of having the *Q*-factor table distributing among the parallel registers, the *Q*-factor can be stored in the BRAM.

Adapting to the FPGA architecture, there are two difference approaches that can map the distributed *Q*-learning network to FPGA hardware. The two approaches are 1.) distributed registering approach and 2.) BRAM-based approach.

6.5.1 Distributed registering approach

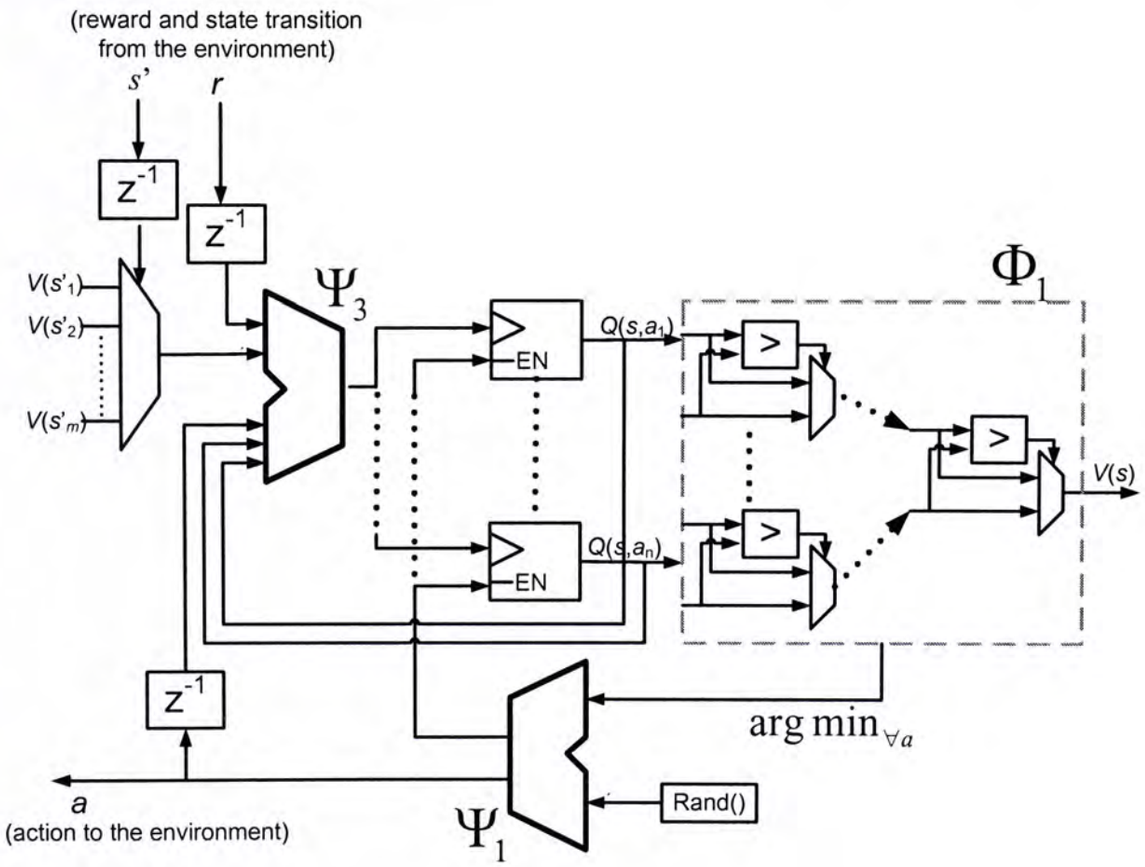


Figure 6.11 Schematic design for the computational unit which represents state s and n actions. For the sake of clarity, the state exploration and exploitation logics are simply denoted by Ψ_1 and Ψ_3. The Q-factor table are mapping to an array of registers in parallel. Each register has an enable input, which is controlled by the exploration unit. It is to control only one Q-factor would be updated with corresponding action. The unit Φ_1 is the minimum operator, which is implemented in parallel by a set of comparator and multiplexer. Besides, there are registers, "z^{-1}",

to keep the input information from the environment, such as state s' and reward r. The action a is also registered, as it is used to select the corresponding "old" Q-factor from the array of factors for the updating.

Registers and logics are abundance in the FPGA chip that distributed design can be benefited from the distributed resources and the high performance of the parallel computation. The *Q*-factor table component Ψ_2 can be parallelized in such way that the Q factors are storing using distributed registers and the minimum operator is realized using parallel design. Figure 6.11 depicts the schematic design of the computational unit based on the distributed registering approach.

The major characteristic of the design is to map the *Q*-factor to an array of distributed registers and using a parallel minimum operator. There is less design effort on the control logics while more logics are expected for the parallelism. Besides, suppose that the computational time for state exploration Ψ_1 and exploitation Ψ_3 is constant with regarding to the number of action, the computational complexity for the A actions state would be reduced to $O(1)$ from $O(A)$. It is because the parallelism applied on the minimum operator, which would take A steps if computes in serial. On the other hand, the resources consumption would grow linearly with the number of actions. Further empirical analysis would be done in the following section.

6.5.2 Serial BRAM storing approach

In contrast to the parallel registering approach, design can take advantage of the availability of the embedded memory in the FPGA. Based on the Xilinx Virtex-II XC2V-2000 FPGA, there are 56 embedded Block RAM (BRAM) available, each with 18kbits storage. As an alternative to the distributed registering design, the *Q*-factor table can be mapped to be stored in the RAM. The schematic design is depicted in Figure 6.12.

The BRAM-based design consumes less logic resources in the minimum operator and the resources consumption would be fixed with regarding to the increasing number of actions. For one BRAM, it would be able to store 1125 *Q*-factor for 16-bit precision and 562 *Q*-factor for 32 bit precision based on the XC2V2000 FPGA. On the other hand, longer computational delay would be expected for the sequential BRAM-based design.

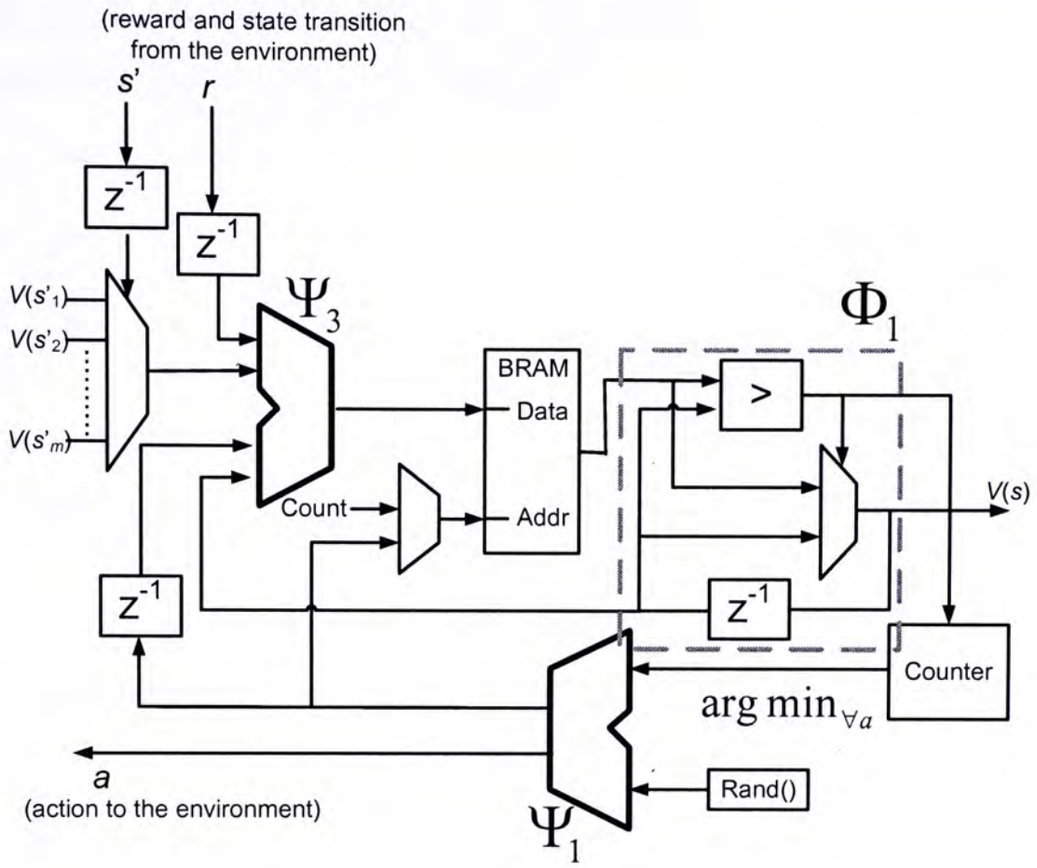


Figure 6.12 In this design, the Q-factor table are mapping to the internal RAM and addressed by the action index. For computing the minimum value, all values from the RAM are needed to be evaluated. The minimum operator Φ_1, which is implemented in serial by evaluating all Q-factor from the RAM. The minimum operator consists of a comparator, multiplexer and a register. The register stores the most updated minimum value and being used to compare with the values from the RAM. If there is another value smaller found, it would be registered. Further, there are two inputs for the address of the RAM. One is used for the minimum operation, as which is the counter output to enumerate all addresses of the RAM. The other input is the action index which is outputs from the exploration component.

6.5.3 Comparison

Current design have that each computational unit represents a binary relation between a state and the destination, whereas the number of state increases linearly with the number of computational units in trivial. In contrast, the possible actions can be taken at a state can be varied according to the problem specification. The distributed register approach and the BRAM approach might dedicate for alternative applications with different needs. Resources utilization

includes BRAM consumption and slice (i.e. logic resources), updating rate, which is the maximum speed the system can perform, would directly correlates to the architectural design.

Table 6.1 Comparison between the two approaches on the consumption of logic slice with respect to the number of actions and bit length

		Number of actions			
	Bit length	**2**	**4**	**8**	**16**
Distributed Registering Approach	**16**	143	169	371	731
	24	164	276	531	1059
	32	200	356	771	1387
	64	381	676	1331	2699
BRAM-based Approach	**16**	146	146	168	208
	24	182	186	216	276
	32	218	226	288	344
	64	386	386	456	616

Both distributed and BRAM-based design increases linearly in resources consumption with increased number of actions. Though the BRAM-based approach consumes more resources in the case of small number of actions (i.e. binary actions), BRAM-based approach increases much slower in the logic usage than the distributed approach with increased number of action. It is simply because the parallel distributed approach requires more logics for the parallelism design. The exceptional case for the binary actions in BRAM-based approach is because that there are extra logics for controlling the read/write process of RAM. Since these logics would not increase with increased number of action, the BRAM-based approach shows more efficient in logic utilization by storing the Q-factor in the memory, provided that sufficient embedded RAMs are available.

The pay-off for the efficient logic utilization for the RAM-based approach is the computational speed. We compared the updating rate, which is the reciprocal of the time from receiving rewards and new transition states to the Q-factor is updated and stored, between two approaches. The higher updating rate shows a design with higher capability of reacting to the changes of environment in real time. The distributed approach has higher updating rate than the BRAM-based approach. For the larger number of actions, the parallel approach has over 12 times faster updating rate than the BRAM-based approach. For larger number of actions, the BRAM-based approach requires more iteration loops to enumerate the values from the BRAM, which is computational expensive.

Table 6.2 Maximum updating rate[1]

	Number of actions			
	2	**4**	**8**	**16**
Distributed Registers Approach	14.02 MHz	13.98 MHz	13.87 MHz	12.287MHz
BRAM-based Approach	7.75 MHz	4.31 MHz	1.91 MHz	1.1 MHz

6.5.4 Discussion

Mapping the distributed Q-learning network to FPGA can have two difference approaches with benefited from the characteristic of FPGA. The distributed approaches benefited from the distributed logics and registers in the FPGA that massive parallelization would provide significant acceleration to the Q-learning computation. However, the resources consumption would be a threat for application with a large number of possible actions, as the logic consumption increases linearly with the increases number of actions. Alternatively, a BRAM-based approach can be more effective in term of logics utilization. The Q-factor table can be

[1] The maximum updating rate is referring to the Xilinx Virtex-II XC2V-2000 FPGA. The rate is based on the maximum frequency reported by the Xilinx-ISE software. Usually, this is maximum frequency is underestimating, as it is computed based on the worst condition of the FPGA board. In practical uses, 20 to 50 percentage improvement can always be found.

storing at the internal BRAM of the FPGA with a sequential procedure for the arithmetic computation. Though a slower updating rate are recorded for the BRAM-based approach, it would be suitable for applications with large number of actions.

6.6 Conclusion

To improve the speed of learning under an unknown environment, a distributed Q-learning network approach is proposed. As a substantial extension of the Bellman inference network, the Q-learning network realizes the architecture of parallel learning and optimization simultaneously. It has been shown a highly efficient approach in solving on-line reinforcement learning problem. We found that the Q-learning network has been greatly improved and outperformed the conventional Q-learning in term of learning speed and adaptation to distributed environment. In addition, we studied the Field Programmable Gate Array (FPGA) implementation of the Q-learning network. Two approaches, distributed registering and BRAM-based, are found for dedicating the FPGAs resources for different applications, such as high speed and problems of high dimension.

Chapter 7

Summary

We proposed a hybrid GA-DP approach to solve the computational intensive equivalence set search problem. The idea is to convert a searching problem to an optimization problem by considering the equivalence set criterion as the heuristic fitness measure in genetic algorithm. We found that the hybrid approach is a more efficient method to locate the equivalence set of genes from a large genetic network, especially when searching small equivalence set from large genetic network. We also offer an efficient FPGAs implementation, based on the Xilinx FPGA prototyping board. The FPGA system can speed-up the computation. Over thousand times acceleration could be achieved over software approach (Lam and Mak 2002). The hybrid GA-DP paradigm would benefit and greatly enhance the computational efficiency of the equivalence sets genes searching and help the development of large-scale genetic network and genetic network dynamics.

Evaluation of a given phylogenetic tree based on the maximum-likelihood criterion is computational intensive. It is also challenging for FPGAs implementation. It is because, 1.) fix-point architecture in FPGAs fails to support the precision demanding probabilistic computation; 2.) the recursive routine based on the tree data structure of the "pruning" algorithm is difficult to realize in FPGAs, as there is no such a FPGA equivalence data structure; 3.) basic computation such as logarithm and exponentiation, which is part of the maximum-likelihood computation, is a challenging task in FPGA implementation.

To circumvent the limitations mentioned above, we, firstly, proposed a simplified floating point architecture based on fixed-point arithmetic. The new architecture could dedicatedly support the probabilistic computation up to 16 significant figures while maintained economic hardware area. Secondly, we developed a recursive architecture based on the "stack" scheme. The recursive architecture supported the realization of the parallel maximum-likelihood

evaluation algorithm. Lastly, we studied the implementation of logarithm and exponentiation using the factoring approach; this turned out to be a hardware area and computational speed efficient approach. The FPGA-based maximum-likelihood hardware showed significant speed-up, from 30x to 100x, comparing to software implementation (Mak and Lam 2004; Mak and Lam 2004a.).

The FPGA-based system provides significant speed-up in the maximum-likelihood phylogeny evaluation. It can be regarded as a coprocessor in accelerating the phylogenetic tree evaluation and the microprocessor is dedicated for tree topology searching. However, there is a communication cost between the software and hardware. This might affect the speed-up ratio. The latency in communication is technology dependent. In (Mak and Lam 2003), we have shown that using HW/SW (Hardware/Software) codesign for GAML implementation can provide speedup over software implementation. The HW/SW approach is benefited from high performance hardware and the flexibility of software. In our preliminary design, the GAML is divided into two parts: the genetic algorithm (excluding the fitness evaluation) and the tree topology evaluation. Since the tree evaluation process is repeated extensively, the overall GAML runtime is reduced tremendously. It is also shown that our HW/SW system can scale up for real applications.

In (Mak and Lam 2004), the HW/SW co-design system has been extended to a more powerful embedded computing platform. In this platform, a microprocessor is immersed into FPGA fabric for realizing an effective environment for HW/SW co-design implementation. Significant improvements in data transmission between hardware and software and higher clock frequency of FPGA have been realized when compared to the parallel port interface in (Mak and Lam 2003).

An FPGA-based architecture for the numerical computation of NMDA and non-NMDA receptors activities and the resultant synaptic plasticity has been presented. The accuracy of the FPGA realization is comparable to software implementations and yet it operates at a much higher speeds. Additionally, the programmability of the FPGA system allows it to prototype analog circuit designs with a much shorter design cycle. Therefore, FPGA technology can be used to fill the gaps between software and hardware simulations. This technology can potentially be used as a tool suitable for dynamic clamp experiments, or to control neuro-prosthetic devices for

chronic replacement of damaged neurons in central regions of the brain (Mak, Rachmouth et al. 2005).

We describes a novel connectionist approach for mapping the value iteration algorithm to a continuous-time inference network solving the dynamic programming in the continuous-time domain. Some of the limitations for the conventional heuristic search techniques, such as increased inefficiency for problem of larger scale, and the discrete time asynchronous distributed approach with requirement for the synchronization of control flags between processors, exponentially increasing of computational loads when discount factor approaches one, have been effectively overcome. Numerical simulations have been used to demonstrate the salient features of such type of continuous-time inference network, for which fully-analog parallel implementation can be made arbitrarily fast and its required computational time is practically independent of problem size. This opens up the possibility for future realization of this class of optimization and learning circuits for real-time decision making applications, such as rover path planning and trajectory design for UAVs for surveillance mission.

Previous work on FPGAs implementation for shortest path problem using the Inference network ravels the high performance parallel architecture (Ng, Mak et al. 2003). We also derived a discrete-time version of the inference network that the network can be well mapped to FPGAs for solving dynamic programming.

To improve the speed of learning under an unknown environment, a distributed Q-learning network approach is proposed. As a substantial extension of the Bellman inference network, the Q-learning network realizes the architecture of parallel learning and optimization simultaneously. It has been shown a highly efficient approach in solving on-line reinforcement learning problem. We found that the Q-learning network has been greatly improved and outperformed the conventional Q-learning in term of learning speed and adaptation to distributed environment. In addition, we studied the FPGAs implementation of the Q-learning network. Two approaches, distributed registering and BRAM-based, are found for dedicating the FPGAs resources for different applications, such as high speed and problems of high dimension.

Bibliography

Adachi, J. and M. Hasegawa (1996). MOLPHY version 2.3, program for molecular phylogenetics based on Maximum Likelihood. Tokyo, Japan, The Institute of Statistical Mathematics.

Akutsu, T., S. Kuhara, et al. (1998). Identification of gene regularity networks by strategic gene disruptions and gene over expressions. Proc. 9th ACM-SIAM Symp. Discrete Algorithm.

Alter, O., P. O. Brown, et al. (2000). "Singular value decomposition for genome-wide expression data processing and modeling." proc. Natl. Acad. Sci. USA **97**(10): 101-106.

Andraka, R. (1998). A survey of CORDIC algorithms for FPGAs. 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays, Monterey, CA.

Bader, D. A., B. M. E. Moret, et al. (2001). Industrial Applications of High-Performance Computing for Phylogeny Reconstruction. SPIE ITCombv: Commercial Applications for High-Performance Computing.

Ballard, D. H., G. E. Hinton, et al. (1983). "Parallel visual computation." Nature **306**: 21-26.

Baltimore, D. (2001). "Our genome unveiled." Nature **409**("Feb. 15").

Barto, A., S. J. Bradtke, et al. (1995). "Learning to Act Using Real-Time Dynamic Programming." Artificial Intelligence **72**(1): 81-138.

Barto, A., R. S. Sutton, et al. (1983). "Neuronlike adaptive elements that can solve difficult learning control problems." IEEE Transactions on Systems, Man, and Cybernetics **13**(5): 834-846.

Baturone, I., J. L. Huertas, et al. (1994). "Current-mode Multiple-input Max circuit." Electronic Letters **30**: 678-680.

Bellman, R. (1957). Dynamic Programming. Princeton, NJ, Princeton University Press.

Berger, T. W., Baudry M., Dias Brinton, R., Liaw, J-S, Marmarelis, V. Z., PARK, A.Y., B J. Sheu, B.J., and TANGUAY, A.R., JR. (2001). "Brain-Implantable Biomimetic Electronics as the Next Era in Neural Prosthetics." Proceedings of the IEEE **89**(7): 993-1013.

Bertin, P. and H. Touati (1994). PAM Programming Environments: Practice and Experience.

IEEE Workshop FPGAs for Custom Computing Machines, Los Alamitos, Calif., CS Press.

Bertsekas, D. (1982). "Distributed Dynamic Programming." IEEE Transactions on Automatic Control **27**(3).

Bertsekas, D. (1987). Dynamic Programming: Deterministic and Stochastic Models. Englewood Cliffs, NJ, Prentice-Hall.

Bi, G. Q. and M. M. Poo (2001). "Synaptic modification by correlated activity: Hebb's postulate revisited." Annual Review of Neuroscience **24**: 139-166.

Bienenstock, E. L., L. N. Cooper, et al. (1982). "Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex." Journal of Neuroscience **2**(1): 32-48.

Bliss, T. V. and T. Lomo (1973). "Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path." Journal of Physiology **232**: 331-356.

Burke, G., S. Cozy, et al. (2004). Operation of FPGAs at Extremely Low Temperatures. 2004 MAPLD International Conference.

Butera, R. J., and M. L. McCarthy, (2004). "Analysis of real-time numerical integration methods applied to dynamic clamp experiments." IEEE Journal of Neural Engineering **1**: 187-194.

C. Chen, R. c. a. C. Y. (2000). "Pipelined Computation of Very Large Word-Length LS Addition/Subtraction with Polynomial Hardware Cost." IEEE Trans. Compu. **49**(7).

Canham, R. O. and A. M. Tyrrell (2003). "A hardware artificial immune system and embryonic array for fault tolerant systems." Genetic Programming and Evolvable Machine **4**: 359-382.

Caro, D. and M. Dorigo (1998). "AntNet: Distrbuted Stigmergetic Control for Communications Networks." Journal of Artificial Intelligence Research **9**: 317-365.

Casper, J. and R. R. Murphy (2003). "Human-Robot Interactions During the Robot-Assisted Urban Search and Rescue Response at the World Trade Center." IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics **33**(3): 367-385.

Cauwenberghs, G. (1997). "Analog VLSI Stochastic Perturbative Learning Architecture." Analog Integrated Circuits and Signal Processing **13**: 195-209.

Cauwenberghs, G. and M. Bayoumi (1999). Learning on Silicon-- Adaptive VLSI Neural

Systems. Norwell MA, Kluwer Academic.

Chan, C. F., H. S. Ling, et al. (1995). "A one volt four-quadrant analog current mode multiplier." IEEE Journal of Solid-state circuits **30**(9): 1018-1019.

Chen, T. C. (1972). "Automatic computation of logarithms, exponentials, ratios and square roots." IBM j. Res. and Dev **16**: 380-388.

Cheng, S.-T. (1998). "Topological optimization of a reliable communication network." IEEE Transactions on Reliability **47**(3): 225-233.

Condon, A. (1992). "The Complexity of Stochastic Games." Information and Computation **96**(2): 203-224.

International Human Genome Project Consortium. (2001). "Initial sequencing and analysis of the human genome." Nature **409**(Feb. 15): 860.

Cormen, T., C. Leiserson, et al. (1990). Introduction to Algorithm, MIT Press.

Crair, M. C. and R. C. Malenka (1995). "A critical period for long-term potentiation at thalamocortical synapses." Nature **375**: 325-328.

D'Epenoux, F. (1963). "A Probabilistic Production and Inventory Problem." Management Science **10**: 98-108.

D'haeseleer, P., S. Liang, et al. (2000). "Genetic network inference: from co-expression clustering to reverse engineering." Bioinformatics **16**(8): 707-726.

D. DasSarna, D. W. m. (1994). "Measuring the accuracy of ROM reciprocal tables." IEEE Trans. Comput. **43**: 932-940.

Darwin, C. (1929). the Origin of Species. London, Watts and Co.

Davis, L. (1991). Handbook of Genetic Algorithm. New York, van Nostrand Reinhold.

Derman, C. (1970). Finite State Markovian Decision Process. New York, Academic Press.

Drosten C., e. a. (2003). "Identification of a Novel Coronavirus in Patients with Severe Acute Respiratory Syndrome." New England Journal of Medicine **348**(20): 1967-1976.

Drummond, A. and K. Strimmer (2001). "PAL: An object-oriented programming library for molecular evolution and phylogenetics." Bioinformatics **17**: 662-663.

Dudek, S. M. and M. F. Bear (1992). "Homosynaptic long-term depression in area CA1 of hippocampus and effects of N-methyl-D-aspartate receptor blockade." Proceedings of the National Academy of Sciences USA **89**: 4363-4367.

Dydek, S. and P. Bala (2004). Large Scale Protein Sequence Alignment. Field Programmable

Logic and Applications.

Eisen, J. A. and C. M. Fraser (2003). "Phylogenomics: Intersection of evolution and genomics." Science **300**(300): 1706-1707.

Eisen, M. B., B. T. Spellman, et al. (1998). "Cluster analysis and display of genome-wide expression patterns." Proc. Natl. Acad. Sci. USA **95**(25): 12863-14868.

Ercegovac, M. and T. Lang (2004.). Digital Arithmetic. San Francisco, Morgan Kaufmann.

Ewe, C. T., P. Y. K. Cheung, et al. (2004). Dual Fixed-point: An Efficient Alternative to Floating-Point Computation. Field Programmable Logic and Applications.

Felsenstein, J. (1981). "Evolutionary trees from DNA sequences: a maximum likelihood approach." J. Mol. Evol. **17**: 368-376.

Felsenstein, J. (1989). PHYLIP - Phylogeny Inference Package (Version 3.2).

Filar, J. and K. Vrieze (1997). Competitive Markov Decision Process, Springer-Verlag.

Frazier, M., D. Thomassen, et al. (2003). Stepping up the pace of discovery: the Genomes to life program. Computational Systems Bioinformatics, Stanford University.

George A. Constantinides, Peter. Y. K. Cheung, and Wayne Luk (2004). Synthesis and Optimization of DSP Algorithms. Dordrecht, The Netherlands, Kluwer Academic Publishers.

Gilbert, B. (1968). "A Precise Four-Quadrant Multiplier with Subnanosecond Response." IEEE Journal of solid-state circuits **3**(4): 365-373.

Graham, P. and B. Nelson (1996). Genetic algorithms in software and in hardware - a performance analysis of workstation and custom computing machine implementations. Fourth Annual IEEE Symp. on FPGAs for Custom Computing Machines.

Guccione, S. A. and E. Keller (2002). Gene matching using JBits. Field-Programmable Logic and Applications, Montpellier, France.

Han, G. and S. Edgar (1998). "CMOS Transconductance Multipliers: A Tutorial." IEEE Trans. on circuits and systems - II: Analog and digital signal processing **45**(12): 1550-1563.

Hasegawa, M., H. Kishino, et al. (1985). "Dating of the human-ape splitting by a molecular clock of mitochondrial DNA." Journal of Molecular Evolution **21**: 160-174.

Haykin, S. (1999). Neural Network: A Comprehensive Foundation, Prentice Hall.

Hebb, D. (1949). The Organization of Behavior. New York, Wiley.

Hoffman, A. and R. Karp (1966). "On Nondeterminating Stochastic Games." Management

Science **12**(359-370).

Hopfield, J. J. (1982). "Neural networks and physical systems with emergent collective computational abilities." Proc. Natl. Acad.Sci. USA **79**: 2554-2558.

Hopfield, J. J. (1984). "Neurons with graded response have collective computational properties like those of two-state neurons." Proc. Natl. Acad.Sci. USA **81**: 3088-3092.

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

Horowitz, E., S. Sahni, et al. (1996). Fundamentals of Data Structure in C. New York, Computer Science Press.

Hwang, B. M. J., N. shirazi, et al. (2001). System Level Tools for DSP in FPGAs. Fied Programmable and Logic Applications 2001.

IBM (2002). "[IBM CoreConnect Bus Architecture, in web http://www-3.ibm.com/chips/techlib/."

Intel (2005). Intel® Pentium® 4 Processor 670, 660, 650, 640, and 630Δ and Intel® Pentium® 4 Processor Extreme Edition Datasheet.

Isaacs, R., Weber, D., and A. Schwartz (2000). "Work Toward Real-time Control of a Cortical Neural Prosthesis." IEEE Transactions on Rehabilitation Engineering **8**(2).

J. Hwang, B. M., N. shirazi and J. Stroomer (2001). System Level Tools for DSP in FPGAs. FPL'01.

J. Pineiro, M. D. E. J. B. (2004). "Algorithm and architecture for logarithm, exponential, and powering computaiton." IEEE Trans. Compu. **53**(9): 1085-1096.

J. Pineiro, S. F. O., J. Muller and J. Bruguera (2005). "High Speed Function Approximation Using a Minimax Quadratic Interpolator." IEE Proceedings Comput. Digit. Tech. **54**(3): 304-318.

Jalali, A. and M. J. Ferguson (1992). "On Distributed Dynamic Programming." IEEE Transactions on Automatic Control.

Jukes, T. H. and C. R. Cantor (1969). Evolution of protein molecules. Mammalian Protein Metabolism. H. N. Munro. New York, Academic Press**:** 21-123.

Jun, M. and R. D'Andrea (2003). Path Planning for Unmanned Aerial Vehicles in Uncertain and Adversarial Environments, Springer.

Kandel, E. R., J. H. Schwartz, et al. (2000). Principles of Neural Science, McGraw-Hill

Companies, Inc.

Karp, R. M. (2003). Keynote Address: The Role of Algorithmic Research in Computational Genomics. Computational Systems Bioinformatics, Stanford University.

Kauffman, S. A. (1969). "Metabolic stability and epigenesis in randomly connected netws." J. Theoret. Biol. **22**: 437-467.

Kealbling, P. L., M. Littman, et al. (1996). "Reinforcement Learning: A Survey." Journal of Artificial Intelligence Research **4**: 237-285.

Kimura, S., M. Hatakeyama, et al. (2003). Inference of S-system models of genetic networks using a genetic local search. The 2003 Congress on Evolutionary Computation.

Kishino, H., T. Miyata, et al. (1990). "Maximum Likelihood Inference of Protein Phylogeny and the Origin of Chloroplasts." J. Mol. Evol. **31**: 151-160.

Kitajima, T. and K. Hara (1990). "A model of the mechanisms of long-term potentiation in the hippocampus." Biological Cybernetics **64**(33-39).

Kostopoulos, D. K. (1991). "An algorithm for the computation of binary logarithms." IEEE Trans. Compu. **40**(11): 1267-1270.

Lam, K. P. (1991). A Continuous-time inference network for minimum cost path problems. IEEE/INNS International Joint Conference on Neural Network, Seattle.

Lam, K. P. (1996). "A Binary Relation Inference Network (Part 2)." Int. J. Systems Science **27**(4): 399-404.

Lam, K. P. (1996). "A Continuous-time Inferece Network and its Hybrid Implementation." Int. J. Systems Science **27**: 1425-1433.

Lam, K. P. and S. T. Mak (2002). On computing transitive-closure equivalence sets using a hybrid GA-DP approach. Field Programmable Logic and Applications, Montpellier, France.

Lam, K. P. and C. J. Su (1996). "A Binary Relation Inference Network (Part 1)." Int. J. Systems Science **27**(4): 387-398.

Lam, K. P. and C. W. Tong (1996). "Closed semiring connectionist network for the Bellman-Ford computation." IEE Proc. Comput. Dig. Tech. **143** (3).

Lam, K. P. and C. W. Tong (1997). "Connectionist network for dynamic programming." IEE Proceedings, Computer and Digital Techniques(144): 163-168.

Langdon, W. B. (1998). Genetic Programming and Data Structures: Genetic Prgramming + Data

Structures = Automatic Programming. Boston, Kluwer.

Laurent, G. and E. Piat (2001). Parallel Q-learning for a block-pushing problem. International conference on intelligent robots and systems, Havaii, USA.

Lemmon, A. and M. Milinkovitch (2002). "The metapopulation genetic algorithm: An efficient solution for the problem of large phylogeny estimation." Proceedings of the National Academy of Sciences **99**(16).

Leong, N. M. P. (2001). FPGA Design Methologies for High Performance Applications. department of Computer Science and Engineering. Hong Kong, The Chinese University of Hong Kong.

Leung, Y., G. Li, et al. (1998). "A genetic algorithm for the multiple destination routing problems." IEEE Transactions on Evolutionary Computation **2**(4): 150–161.

Lewis, P. (1998). "A Genetic Algorithm for Maximum Likelihood Phylogeny Inference Using Nucleotide Sequence Data." Mol. Biol. Evol. **15**(3): 277-283.

Littman, M. and J. Boyan A distributed reinforcement learning scheme for network routing.

Liu, S.-C., J. Kramer, et al. (2002). Analog VLSI: Circuits and Principles. Cambridge, Massachusetts, The MIT Press.

Loeb, G. E. (1990). "Cochlear prosthetics." Ann. Rev. Neurosci. **13**: 357-371.

Lugish, B. D. (1970). A class of algorithms for automatic evaluation of functions and computation in a digital computer. Dept. of Comput. Sci. Univ. of Illinois. Urbana.

Mak, T. and K. P. Lam (2004). Embedded Computation of Maximum-Likelihood Phylogeny Inference Using Platform FPGA. IEEE Computer Society Bioinformatics Conference.

Mak, T. S. and K. P. Lam (2004). FPGA-based Computation for Maximum-Likelihood Phylogenetic Tree Evaluation. Field Programmable Logic and Applications, Antwerp, Belgium.

Mak, T. S. and K. P. Lam (2004.). On Maximum-Likelihood Phylogeny Using FPGA. PhD Forum of Field Programmable Logic and Applications, Antwerp, Belgium.

Mak, T. S., G. Rachmouth, et al. (2005). Field Programmable Gate Array Implementation of neuronal Ion Channel Dynamics. The 2-nd International IEEE EMBS Conference on Neural Engineering, Arlinton.

Mak, T. S. T. and K. P. Lam (2003). High Speed GAML-based Phylogenetic Tree Reconstruction Using HW/SW Codesign. IEEE Computer Society Bioinformatics

Conference.

Maki, Y., D. Tominaga, et al. (2001). Development of a system for the inference of large scale genetic network. Proc. Pacific Symposium on Biocomputing.

Martin, P. (2001). "A hardware implementation of a genetic programming system using FPGAs and Handle-C." Genetic Programming and Evovable Machines **2**: 317-343.

Mauritz, K. H. and H. P. Peckham (1987). "Restoration of grasping functions in quadriplegic patients by functional electrical stimulation (FES)." Int. J. Rehab. Res. **10**: 57-61.

Mead, C. A. (1989). Analog VLSI and Neural System.

Mead, C. A. (1990). "Neuromorphic electronic systems." Proceedings of the IEEE **78**: 1629-1636.

Mencer, O., M. Morf, et al. (1998). PAM-Blox: High Performance FPGA Design for Adaptive Computing. IEEE Symposium on FPGAs for Custom Computing Machines.

Milos D. Ercegovac, T. L. (2003). Digital Arithmetic, Morgan Kaufmann.

Mjolsness, E., T. Mann, et al. (1999). From coexpression to co-regulation: an approach to inferring transcriptional regulation among gene classes from large-sclae expression data, Jet Propelsion Laboratory.

Mulkey, R. M. and R. C. Malenka (1992). "Mechanisms underlying induction of homosynaptic long-term depression in area CA1 of the hippocampu." Neuron **9**: 967-975.

Muller, J. M. (1985). "Discrete basis and computation of elementary functions." IEEE Trans. Compu. **C34**(9): 857-862.

Ng, H. S. (1996). Applications and Implementation of Neuro-Connectionist Architectures. Systems Engineering. Hong Kong, The Chinese University of Hong Kong.

Ng, H. S. and K. P. Lam (1996). Current-mode optimiaztion circuits for minimax path problems. IEEE International Symposium in Circuits and Systems, Atlanta.

Ng, H. S. and K. P. Lam (2003). "Analog and digital FPGA implementation of BRIN for optimization problems." IEEE Transactions on Neural Networks **14** (5): 1413–1425.

Ng, H. S., S. T. Mak, et al. (2003). Field Programmable Gate Array and Analog Implementation of BRIN for Optimization Problems. International Symposium on Circuit and Systems, Bangkok, Thailand.

Olsen, G. J., H. Matsuda, et al. (1994). "fastDNAml: A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood." Comput. Appl. Biosci. **10**: 41-48.

Peshkin, L. and V. Savova (2002). Reinforcement learning for adaptive routing. Joint conference on Neural Network.

Peterson, W. W. and E. J. W. Jr. (1972). Error-Correction Codes. Boston, MA, MIT Press.

Rachmuth, G., and Poon, C-S., (2004). In-Silico model of NMDA and non-NMDA receptor activities using analog VLSI circuis. Post Genomic Perspectives in Modeling and Control of Breathing. J. Champagnat, Kluwer Academic/Plenum Publishers. **551:** 171-175.

Rachmuth, G. and C.-S. Poon (2003). Design of a neuromorphic Hebbian synapse using analog VLSI. Neural Engineering, 2003. Conference Proceedings. First International IEEE EMBS Conference on, Capri, Italy, IEEE.

Rambaut, A., D. Posasa, et al. (2004). "The causes and consequences of HIV evolution." Nature Reviews Genetics **5**: 52-61.

Renger, J. J., C. Egles, et al. (2001). "A Developmental Switch in Neurotransmitter Flux Enhances Synaptic Efficacy by Affecting AMPA Receptor Activation." Neuron **29**: 469-484.

Sasaki, M., J. Inouk, et al. (1990). "Fuzzy multiple-input maaximum and minimum circuits in Current mode and their analyses using bounded-different equations." IEEE Trans. on Computers **39**: 768-774.

Schneider, J., W. Wong, et al. (1999). Distributed Value Functions. 16th International Conference on Machine Learning.

Shackleford, B., G. Snider, et al. (2001). "A high-performance, pipelined, FPGA-based genetic algorithm machine." Genetic Programming and Evovable Machines **2**(33-60).

Sharp A., O. N. M., Abbott LF, Marder E. (1993). "Dynamic clamp: computer-generated conductances in real neurons." Journal of Neurophysiology **69**(3): 992-995.

Shoham, Y., R. Powers, et al. (2004). Multi-agent reinforcement learning: a critical survey. AAAI Fall Symposium on Artificial Multi-Agent Learning.

Shouval, H. Z., M. F. Bear, et al. (2002). "A unified model of NMDA receptor-dependent bidirectional synaptic plasticity." Proceedings of the National Academy of Sciences USA **99**(16): 10831-10836.

Smyth, G. K., Y. H. Yang, et al. (2002). Statistical Issues in cDNA microarray Data Analysis, Functional Genomics: Methods and Protocols. Totowa, NJ, Humana Press.

Stamatakis, A., T. Ludwig, et al. (2002). AxML: A Fast Program for Sequential and Parallel

Phylogenetic Tree Calculations Based on the Maximum Likelihood Method. Proceedings of 1st IEEE Computer Society Bioinformatics Conference, Palo Alto, California.

Strimmer, K. and A. Haeseler (1996). "Quartet Puzzling A Quartet Maximum-Likelihood Method for Reconstructing Tree Topologies." Mol. Biol. Evol. **13**(7): 964-969.

Strimmer, K. and A. V. Haeseler (2003). Nucleotide Substitution Models. The Phylogenetic Handbook. A. M. V. M. Salemi. Cambridge, UK, Cambridge University Press.

Subramanian, D., P. Druschel, et al. (1997). Ants and reinforcement learning: A case study in routing in dynamic networks. Fifteenth International Joint Conference on Artificial Intelligence.

Sutton, R. and A. Barto (1998). Reinforcement Learning: An Introduction, MIT Press.

Swofford, D. L. (2003). PAUP*. Phylogenetic Analysis Using Parsimony (*and Other Methods). Version 4. Sunderland, Massachusetts, Sinauer Associates.

Swofford, D. L., G. J. Olsen, et al. (1996). Phylogenetic Inference. Molecular Systematics. D. M. Hillis, C. Moritz and B. K. Mable. Sunderland, Mass., Sinauer.

Takagi, H. H. a. N. (1995). Function Evaluation by Table Look-up and Addition. 12th Symposium on Computer Arithmetic.

Taylor DM, H. T. S., and Schwartz AB (2002). "Direct Cortical Control of 3D Neuroprosthetic Devices." Science **296**: 1829-1832.

Team, R. (1997). "Characterization of the Martian Surface Deposits by the Mars Pathfinder Rover, Sojourner." Science **278**: 1765-1767.

TimeLogic (2002). World Wide Web site http://www.timelogic.com.

Tin, C. (2004). Robust Multi-UAV Planning in Dynamic and Uncertain Environments. Department of Aeronautics and Astronautics. Massachusetts, Massachusetts Institute of Technology.

Toumazou, C., F. J. Lidgey, et al. (1990). Analog IC design: the current-mode approach. London, Peter Peregrinus Ltd.

Trimberger, S., R. Pang, et al. (2003). A 12 Gbps DES Encryptor/Decryptor Core in an FPGA. Cryptographic Hardware and Embedded Systems - CHES 2000: Second International Workshop.

Tsitsiklis, J. (1994). "Asynchronous Stochastic Aproximation and Q-learning." Machine Learning **16**: 185-202.

Venter, J. C. and e. al. (2001). "The sequence of the human genome." Science **291**(Feb. 16): 1304.

Villalba, J. H. a. J. (2000). A hardware algorithm for variable-precision logarithm. IEEE International Conference on Application-Specific Systems, Architectures, and Processors.

Vittoz, E. A. (1994). "Analog VLSI signal processing: Why, Where, and How?" Journal of VLSI signal processing **8**: 27-44.

Volpe, R., T. Estlin, et al. (2000). Enhanced Mars Rover Navigation Techniques. Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco CA.

Wall, M. E., P. A. Dyck, et al. (2001). "SVDMAN-Singular value decomposition analysis of microarray data." Bioinformatics **17**(6): 566-568.

Watkins, C. (1989). Learning from Delayed Rewards. Cambridge, University of Cambridge.

Watkins, C. and P. Dayan (1992). "Q-learning." Machine Learning **8**(3): 279-292.

Weib, G. (1995). "Distributed reinforcement learning." Robotics and Autonomous Systems **15**: 135-142.

Whitman, W. B., D. C. Coleman, et al. (1998). "Prokaryotes: The unseen majority." Proc. Natl. Acad. Sci. USA **95**: 6578.

Williams, B. C., P. Kim, et al. (2001). Model-based Reactive Programming of Cooperative Vehicles for Mars Exploration. Int. Symp. on Artificial Intelligence, Robotics and Automation in Space, St-Hubert, Canada.

Worobey, M., M. L. Santiago, et al. (2004). "Origin of AIDS: Contaminated polio vaccine theory refuted." Nature **428**: 820.

Xilinx (2002). "Virtex-II Pro(TM) Platform FPGA Handbook."

Xilinx (2003). Xilinx FPGAs Aboard Mars 2003 Exploration Mission.

Yang, Z. H. (2001). Maximum Likelihood Analysis of Adaptive Evolution in HIV-1 GP120 ENV Gene. Pacific Symposium on Biocomputing, Hawaii.

Yu, C. W., K. H. Kwong, et al. (2003). A Smith-Waterman Systolic Cell. Field Programmable Logic and Applications, Lisbon.

Zador, A., C. Koch, et al. (1990). "Biophysical model of a Hebbian synapse." Proceedings of the National Academy of Sciences USA **87**: 6718-6722.

Appendix A

Simplified Floating-Point Arithmetic

For any real positive number A, which can be represented as the simplified floating-point based on the fixed-point architecture, that $A = A' \times 2^{-R}$ where A' and R' are represented in fixed-point. Thus $R' = \sum_{j=1}^{p} r_j \cdot 2^{-j}$ (exponent) and $A' = \sum_{i=1}^{m} a_i \cdot 2^{-i}$ (mantissa), where a_i and r_i is either zero or one and m is the word length. In addition, the number A is *normalized* that A' is always a number between 0.5 and 1. The implementation of multiplying two numbers is as follow:

$$A \times B = A' \cdot 2^{-s} \times B' \cdot 2^{-t} = \begin{cases} \left(A' \times B'\right) \cdot 2^{-s-t}, & \text{if } A' \times B' \geq 0.5 \\ \left(A' \times B' \times 2\right) \cdot 2^{-s-t-1}, & \text{otherwise} \end{cases} \quad \text{(A.1)}$$

where the number in the blanket is the mantissa and the superscripts are the exponent. The mantissa of the product $A' \times B'$ will be within the range of 0.25 and 1, as A and B were normalized. Simple logic to detect whether the product is larger than 0.5 is required.

In addition operation, assumed that the input number is normalized:

$$A + B = A' \cdot 2^{-s} + B' \cdot 2^{-t} = \begin{cases} \left(A' \cdot 2^{-1} + B' \cdot 2^{-t+s-1}\right) \cdot 2^{-s+1}, & \text{if overflow occurs} \\ \left(A' + B' \cdot 2^{-t+s}\right) \cdot 2^{-s}, & \text{otherwise} \end{cases} \quad \text{(A.2)}$$

where $|s| < |t|$ and overflow occurs if $A' + B' \cdot 2^{-t+s} > 1$. The mantissa B' is shifted in order to have the same exponent as A'.

Appendix B

FPGA Implementation for Logarithm, Exponentiation, and Division

B.1 Introduction

Evaluation of basic functions, such as logarithm $\log X$, exponentiation e^X and division Y/X are in great demand in many application of the Field Programmable Gate Array (FPGA) system design, such as computer 3D graphics, digital signal processing (DSP), scientific computation and biomedical signal processing (Constantinides et al. 2004). Architecture for basic function evaluation becomes a critical issue, as design trade-off landscape in digital hardware is generally more complicated than traditional software approach. Because of the limited hardware resources, hardware system design usually comprises multiple design objectives, such as hardware area, memory and power consumption minimization. In contrast, optimization of the computational speed and accuracy appears to be critical in software approach, of which memory and power is not a critical concern.

Since FPGA allows flexible allocation of on-chip hardware resources, such as Block-Random-access-memory (BRAM) and logic gates, approach or architecture for basic function evaluation permits flexible trade-off on hardware area, computational time and accuracy are highly preferred and in great demand for the optimization of the multiple objectives in FPGA system design.

Traditional software routines provide accurate results in elementary function computations given enough time for iterations. For example, the shift-and-add method employs only the addition and shift operations (i.e. multiplications by a power of the radix of the number system used) that slowly converges to the desired approximated solution (Lugish 1970; Chen 1972;

Muller 1985; C. Chen 2000). For example, COordinate Rotation DIgital Computer (CORDIC) iterative solutions for trigonometric and other transcendental functions that use only shifts and adds to perform. The trigonometric functions are based on vector rotations, while other function such as square root are implemented using an incremental expression of the desired function (Andraka 1998). CORDIC generally produce one additional bit of accuracy for each iteration.

The other example is the multiplicative/additive normalization, which has been proposed to compute natural exponential function (Chen 1972). In additive normalization for evaluating the e^X, input operand $X_0=X$ is normalized to zero by successively subtracting the jth normalization term $\log(1+s_j r^{-j})$ from the jth remaining term X_j. In each step, the methods also evaluates the jth partial result term $e_j=e_j(1+s_j r^{-j})$ that is the partial result of the exponential function with accuracy up to r^{-j}. Suppose the base r equals to two, the evaluation for the jth partial result would be simply addition and shifting. Similar idea is used for the multiplicative normalization to compute the logarithm. But the input operant would be normalized to one with multiplying the normalization tem $(1+s_j r^{-j})$ while the partial result term is evaluated by subtracting the term $\log(1+s_j r^{-j})$ is the jth operation.

Alternatively, table based approach provides faster computation with the approximate or exact solution directly looked up from the pre-computed table stored in Read-only-memory (ROM) or Random-access-memory (RAM) (D. DasSarma 1994; Takagi 1995; Villalba 2000). However, the ROM/RAM space increases exponentially with the word length or number of accurate bits. While interpolation techniques with table-driven algorithm based on an enhanced minimax quadratic computation reduces the table size in single-precision floating-point format (J. Pineiro 2004; J. Pineiro 2005). However these routines generally numerical intensive requires high precision multiplier, which consumes a lot of hardware resources in FPGA. Though a very high precision can be obtained, the hardware resources hungry algorithm make the implementation not suitable (Kostopoulos 1991).

B.2 Approximation Scheme

B.2.1 Logarithm

Multiplicative normalization has been used for reciprocal, for division and for logarithm. In all cases, there is a sequence that converges towards one, and this controls the convergence of

another sequence towards the result linearly (Milos D. Ercegovac 2003).

The iterative algorithm consists of determining a sequence f_n such that the sequence y_n converges to one, where

$$y_{j+1} = y_j \cdot f_j \tag{B.1}$$

where we set $y_0=X$ and

$$f_j = 1 + s_j \cdot r^{-j} \tag{B.2}$$

where r is the radix of the algorithm and $s_n = s_0, s_1, \ldots, s_{m-1}$ is a sequence that controls the multiplication of factor f_j. If we define $s_i \in \{0,1\}, i = 1,2,\ldots,n$, then s_i is regarded as a factor selection variable controlling the multiplication of factor f_j. Note that with careful selection of s_i, the multiplicative normalization produces a continued product representation of the reciprocal of x, that is

$$\frac{1}{X} \approx \prod_{j=1}^{n} f_j = \prod_{j=1}^{n} (1 + s_j 2^{-j}) \tag{B.3}$$

where this normalization can be used to produce an approximation of the reciprocal function. With little modification on Eq.(B.3), we can approximate the logarithm values. Suppose we take logarithm on Eq(B.3), such that it becomes

$$\log X \approx -\sum_{j=1}^{n} \log f_j = -\sum_{j=1}^{n} \log(1 + s_j 2^{-j}) \tag{B.4}$$

However, n iterations are required for the multiplicative normalization converging to a solution with error less than $\log(1+2^{n})$. For given available of larger memory storage, that tradeoff between computational time and memory consumption is feasible.

B.2.1.1 Integrate Direct Table Look-up and Computation

Suppose for taking the logarithm of a number X, which can be written as $X=1+b+c$, where $b>c$. The term b and c can be expressed as a fixed point representation with binary number x_i equals to either zero or one. Thus we have $b= 0.x_0x_1x_2\ldots x_{k-1}$ *and* $c= 0.00\ldots0x_k\ldots x_n$. In other words, b

can be represented by k-bit with all bits are fractional bits. Then the logarithm of X can be expressed as the sum of two terms,

$$\log(1+b+c) = \log(1+b) + \log(1+\frac{c}{1+b}) \tag{B.5}$$

The first term $\log(1+b)$ can be exactly evaluated by Look-Up-Table of k-bit address and m-bit word length. We denote the LUT by $F(w_k) = \log(w_k)$, w_k is any input number with k-bit representation. The results from the table look-up can be directly fed into the multiplicative normalization to approximate the function for the remaining term.

Since the logarithm of y has been changed to $\log(1+c/(1+b))$, the original formulation of normalization is not applicable. It is difficult to evaluate the input value, there is a division operation involved in the expression $\log(1+c/(1+b))$. But the function approximation can be modified, such that $1+c/(1+b)$ is multiplied by a sequence of factors with converging to one as follow

$$\frac{1+b+c}{1+b}\cdot\prod_{j=k}^{m}(1+s_j 2^{-j}) \to 1, \quad \text{if } (1+b+c)\cdot\prod_{j=k}^{m}(1+s_j 2^{-j}) \to 1+b \tag{B.6}$$

such that $(1+b+c)\prod_{j=k}^{m}(1+s_j 2^{-j}) \to (1+b)$ and $\log(1+b)+\sum_{j=k}^{m}\log(1+2_j s^{-j}) \to \log(X)$. The algorithm is as follows. We set the initial input $y_0 = 1+b+c$ and $x_0 = \log(1+b) = F(1+b)$.

for j from k to m

$$s_j = \begin{cases} 0, & \text{if } y_j\cdot(1+2^{-j}) - (1+b) \geq 0 \\ 1, & \text{otherwise} \end{cases}$$

$$y_{j+1} = y_j \cdot f_j, \quad \text{where } f_j = 1+s_j 2^{-j}$$

$$x_{j+1} = x_j - \log(1+s_j 2^{-j})$$

end

B.2.2 Exponentiation

Additive normalization is used for exponentiation approximation. Similar to the method to compute logarithm, exponential is approximated by the product of a sequences. Note that the

exponential is with base 2 in our discussion. The method is applicable for other base[1]. In this case, the input value X is converging to zero by adding a sequence of $-f_i$, as follows

$$X - \sum_{j=1}^{n} f_j \to 0, \quad \prod_{j=1}^{n} f_j = \prod_{j=1}^{n} (1 + s_j 2^{-j}) \to 2^X \tag{B.7}$$

The exponential value is a product of f_i.

B.2.2.1 Integrate Direct Table Look-up and Computation

Unlike the method on computing the logarithm, the hybrid approach is to partition the exponential function into two product terms. The first term consists of the first k bit of the exponential X, and the second term consists of the remaining bits, such that $X = b + c$, where $b=0.x_0x_1x_2,\ldots,x_{k-1}$ and $c = 0.00,\ldots,0x_kx_{k+1}x_{k+2},\ldots,x_n$. Therefore we can look up the exponential value of the first term with k bit from a table. A table of 2^k depth and m-bit word length can be designed for the function $G(X) = 2^X-1$ that the value will be all less than one. Since evaluation of exponential of X has been changed to the evaluation of the $2^{0.00\ldots0x_kx_{k+1}x_{k+2},\ldots,x_n}$, the original formulation of normalization is not applicable. Thus we can set the initial input $y_0 = b$ and $x_0 = G(b)+1$.

for j from k to m

$$s_j = \begin{cases} 0, & \text{if } y_j - \log(1 + 2^{-j}) < 0 \\ 1, & \text{otherwise} \end{cases}$$

$$y_{j+1} = y_j - f_j, \quad \text{where } f_j = 1 + s_j 2^{-j}$$

$$x_{j+1} = x_j \cdot (1 + s_j 2^{-j})$$

end

B.2.3 Division

Multiplicative normalization is used for approximating the division Y/X. Reciprocal is a special case of division that Y equals to one. The approximation formulation has been shown in Eq.(B.3). We now consider the hybrid solution with considering the first k-bit of X is a table

[1]. The other common bases are e, 10, 8, etc

look-up instead of computation.

Suppose for taking the division Y/X, that X can be written as X=1+b+c, where b>c. The term b and c can be expressed as a fixed point representation with binary number x_i equals to either zero or one. Thus we have b= $0.x_0x_1x_2...x_{k-1}$ and c= $0.00...0x_k...x_n$. In other words, b can be represented by k-bit with all bits are fractional bits. Then the division of X can be expressed as the product of two terms,

$$\frac{Y}{X}=\frac{Y}{1+b+c}=\frac{1}{1+b}\cdot\frac{Y\cdot(1+b)}{1+b+c} \tag{B.8}$$

The first term $1/(1+b)$ can be directly by Look-Up-Table of k-bit address and m-bit word length. We denote the LUT by $H(w_k)$ = $1/(1+w_k)$, w_k is any input number with k-bit representation. The results from the table look-up can be directly fed into the multiplicative normalization to approximate the function for the remaining term.

Since the division been changed to $\frac{Y\cdot(1+b)}{1+b+c}$, the original formulation of normalization is not applicable. Similar to the method used in the logarithm, the function approximation can be modified, such that

$$\frac{1+b}{1+b+c}\cdot\prod_{j=k}^{m}\frac{1}{(1+s_j2^{-j})}\rightarrow 1,\quad \text{if } (1+b+c)\cdot\prod_{j=k}^{m}(1+s_j2^{-j})\rightarrow 1+b \tag{B.9}$$

Thus we set the initial input y_0 = 1+b+c and $x_0 = Y\times H(1+b)$.

for j from k to m

$$s_j=\begin{cases}0, & \text{if } y_j\cdot(1+2^{-j})-(1+b)\geq 0\\ 1, & \text{otherwise}\end{cases}$$

$$y_{j+1}=y_j\cdot f_j,\quad \text{where } f_j=1+s_j2^{-j}$$

$$x_{j+1}=x_j\cdot(1+s_j2^{-j})$$

end

Appendix C
Analog VLSI Implementation

In a standard-IC implementation of the binary relation inference network (Lam and Tong 1997), it was shown that it was possible to implement the inference network with analog processing units. However, with practical problems that involve large number of nodes, this implementation would have problems in building a network with reasonable size. The simple computations involved in the units and the regular interconnection in the network are very suitable for a VLSI implementation. The following describe a VLSI design in solving MDPs Bellman optimization problems with binary relation inference network.

C.1 Site Function

In the last few decades, the vast majority of analog circuits have used voltages to represent and process relevant signals. However, recently, current-mode signal processing circuit, in which signals and state variables are represented by currents rather than voltages (Toumazou, Lidgey et al. 1990), have shown advantages over their voltage-mode counterparts. Their advantages include higher bandwidth, higher dynamic range, and they are more amenable to lower power suppliers (Liu, Kramer et al. 2002). A number of possible current-mode configurations are feasible to define the arithmetic operations in the site function describe in Eq. 1.8. As signal is presented in the form of current, they can be added by simply connecting them together. However, analog multiplication is still a challenging subject (Han and Edgar 1998), as noise and low bandwidth often diminish the performance of the multiplier.

C.1.1 A multiplication cell

The Gilbert multiplier cell (Gilbert 1968) has been the most popular multiplier circuit for the past three decades. Taking advantages of the current-mode approach, the original voltage-mode design has been modified and several current-mode forms multiplier cell (or Gilbert cell) emerge (Toumazou, Lidgey et al. 1990). In line with Gilbert's work, Chan et. al, proposed a simple layout circuit with high dynamic range to achieve the multiplication function.

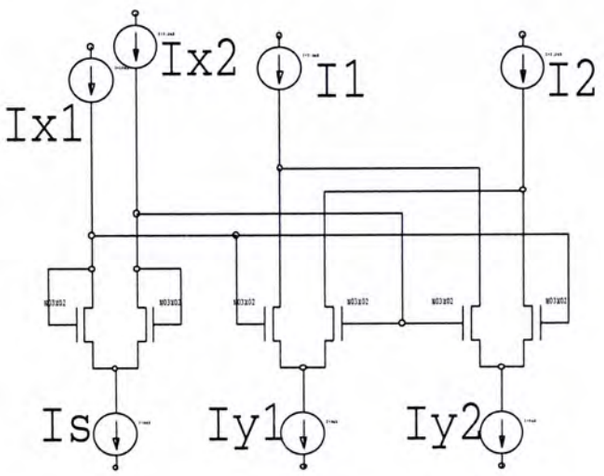


Figure C.1. Four-quadrant current-mode multiplier cell (Chan, Ling et al. 1995)

Fig. C.1 illustrate the architecture of the multiplier cell. The relationships for the important signals $Ix1$, $Ix2$, Is, $Iy1$, $Iy2$, $I1$ and $I2$ are described as

$$I_1 - I_2 = \frac{(Iy1 - Iy2)(Ix1 - Ix2)}{Is} \tag{C.1}$$

$$Is = (Ix1 + Ix2) \tag{C.2}$$

The multiplication performs by ($Ix1$-$Ix2$) and ($Iy1$-$Iy2$), where Ix and Iy are current inputs. Is is another current-input which equals to the current $Ix1$+$Ix2$. To fit our inference network application, modification of the circuit is required that the circuit is expected able to perform simple two quadrant multiplication where $Z = K \cdot XY$ with X and Y are inputs, K is constant. The modifier multiplier cell is shown in Fig. C.2. We introduce a pair of current mirror Q1, which are mirroring currents Ix1 and Ix2. The sum of the two currents from Q1 is connected to Is, which is another current source equaling to the sum of $Ix1$ and $Ix2$ as in Eq. C.2. The

other pair of current mirrors Q2 are used for mirroring the output currents I1 and I2. Also current mirror Q3 is used to invert the current of I2 following Eq 1.1.

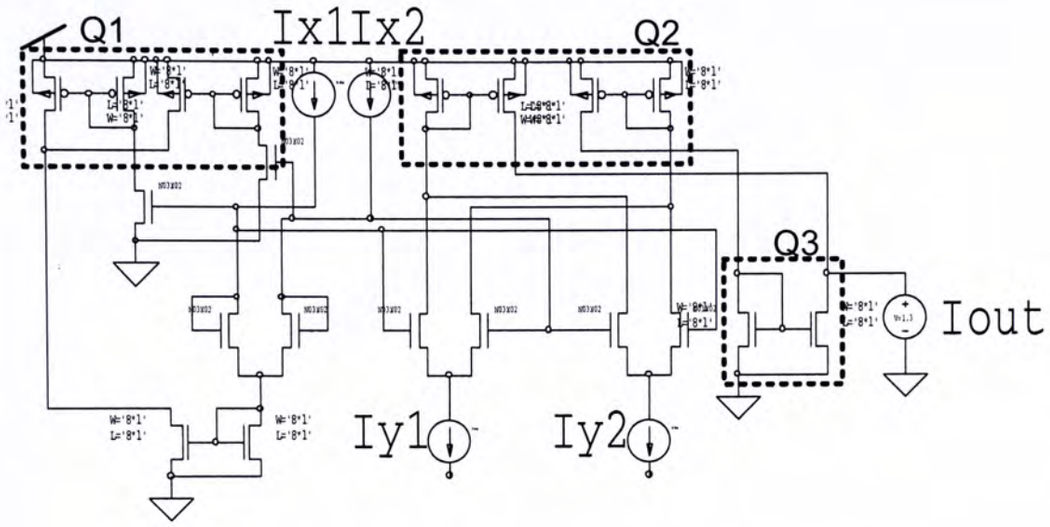


Figure C.2. Current-mode multiplier cell

Following the circuit in Fig. C.2, the multiplication can be easily realized with letting Iy2 equals to zero. For simplicity, we let Ix1 equals to 1. Consider $z = x' \cdot y$ that equation C.1 can be modified as

$$Iout = Iy2 \cdot x' \tag{C.3}$$

where

$$Iy1 = 0, \quad Ix1 = 1 \tag{C.4}$$

$$x' = \frac{1 - Ix2}{1 + Ix2} \tag{C.5}$$

Following Eq. C.5, the required input for Ix2 can be easily computed. For example, x'=0.5, Ix2 = 0.5/1.5 = 0.3333.

Because of the current-mode design, a wider operation range can be found when comparing to the voltage-mode design (Chan, Ling et al. 1995). Response of the multiplier was shown in Figure 3 for x' between zero and one, which is range of probability values. A fairly good performance of the multiplication results is shown in the circuit response figure. It can be observed that the multiplication of two larger numbers would give a better accuracy.

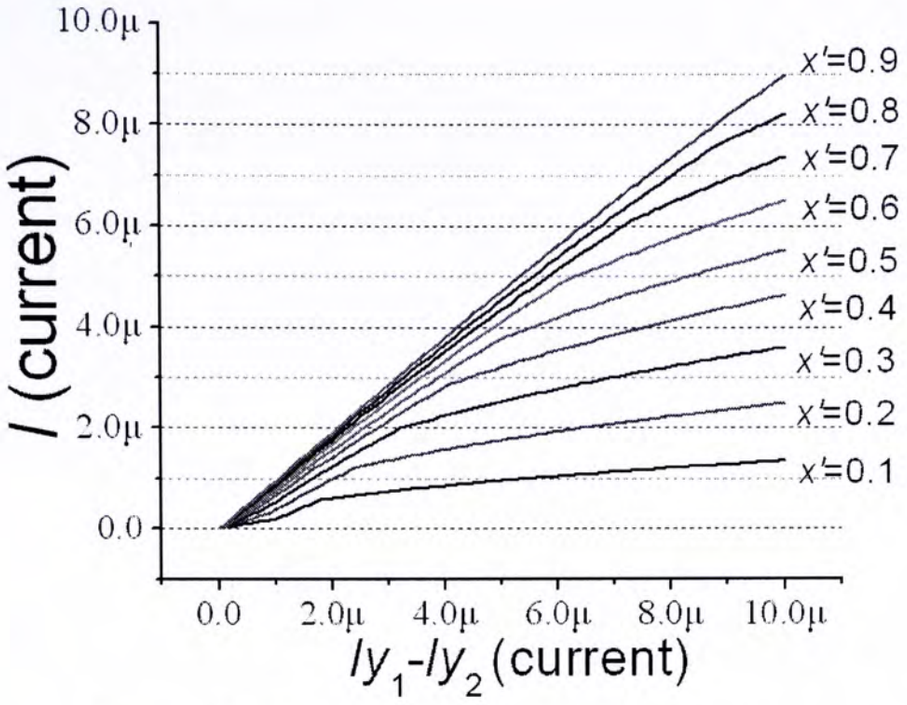


Figure C.3 Current-mode multiplier cell response

C.2 The Unit Function

A minimum selecting function block is required in the unit output function. Several circuits are reported in finding the maximum or the minimum from a group of signals. In (Baturone, Huertas et al. 1994), a current-mode minimum circuit was reported. The circuit can be constructed by the addition and subtraction of replicas of two currents. Also they can be expressed as bounded-difference and algebraic sum (Sasaki, Inouk et al. 1990).

Following the proposed minimum circuit in (Vittoz 1994), we construct the unit function, which is used to resolve the outputs from the site functions. The schematic is shown in Fig. 4. The circuit was shown with a large dynamic range and with best performance in the 0 - 17 μA (Ng and Lam 1996). The minimum circuit response was shown in Fig. 5.

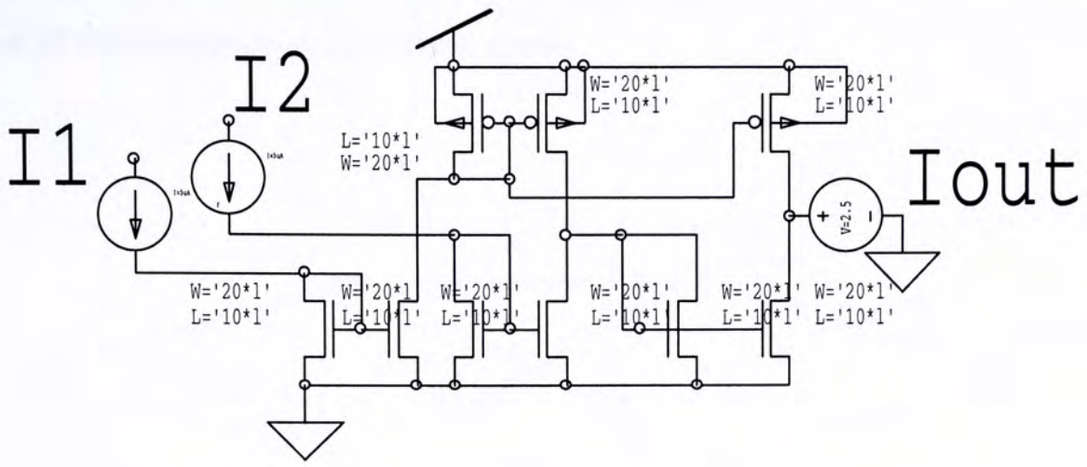


Figure C.4 Current-mode minimum circuit

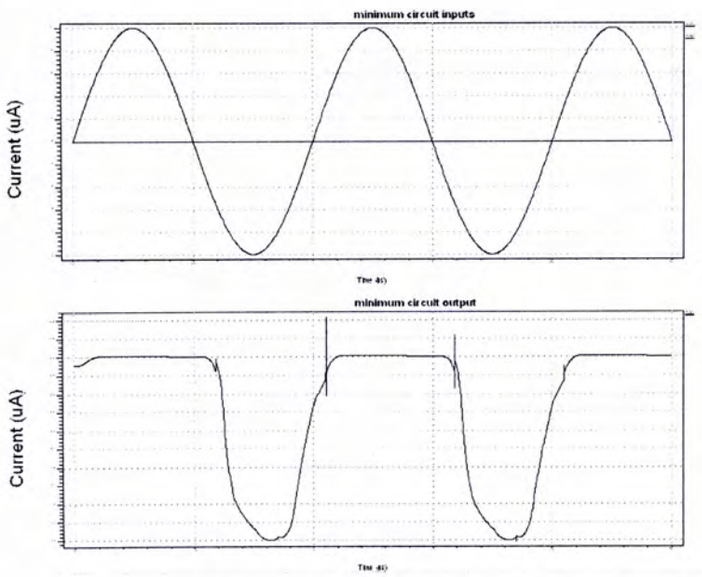


Figure C.5 Minimum circuit response to a sinusoid and constant inputs. The circuit delay can be found at around 80ns

C.3 The Inference Network Computation

Considered a 6-node shortest path problem in Fig. 6, in which node A represents the starting city and node F represents the destination. There are arc costs associating with each pair of states. Further, probability $T(i, a_{ik}, i')=0.8$ if $k = i'$ and $T(i, a_{ik}, i')=0.2$ if $k \neq i'$. The semantic meaning is that one can reach the next city, which would be the desire city with probability 0.8, or one will arrive at some other random states with probability 0.2. The

problem is mapping to the BRIN computational network using CMOS transistor based on the description of unit construction discussion above.

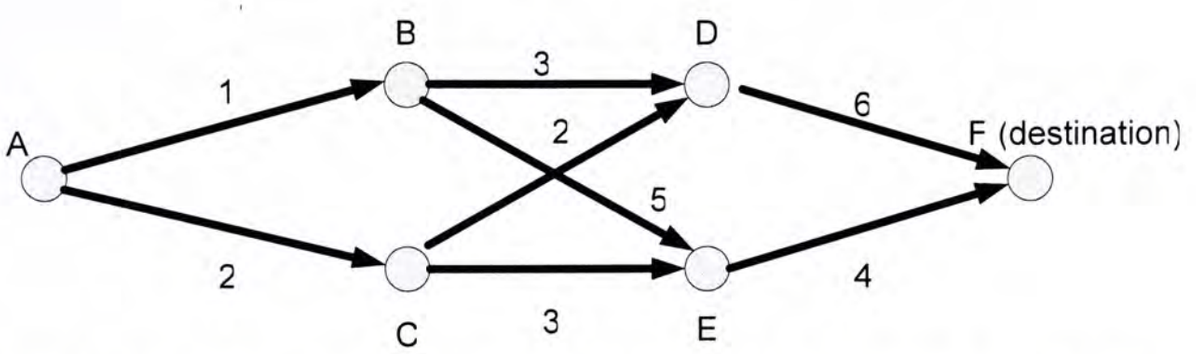


Figure C.6 A typical 6-state MDPs problem, where state F is the regarded as destination and state A is regarded as start. There are arc costs associating with each pair of states. Further, probability $T(i, a_{ik}, i')=0.8$ if $k = i'$ and $T(i, a_{ik}, i')=0.2$ if $k \neq i'$. The semantic meaning is that one can reach the next state, which would be the desire state with probability 0.8, or one will arrive at some other random states with probability 0.2

The worst case propagation delay of a unit, τ_{unit} is given by the sum of delays in the multiplication circuit and the minimum circuit. For a 6-node network, assuming that the discount factor λ equals to 0.5, τ_{unit} is 300ns. The results are shown in Fig. 7. The figure on left is the numerical simulation of the 6-node single-destination inference network based on solving first-order ordinal differential equation using Matlab. The figure on the right is the inference network circuit simulation for the same problem. The expected value for state A, B and C (referring to Fig. C.6) is [4.2, 4.8, 6.4], as shown in the left figure. In the circuit simulation, expected values are presented by current in the μA operational range. The results obtained is [3.95 μA, 4.83 μA, 6.74 μA]. The results from the circuit simulation can accurate compute the optimal value based on the Bellman formulation, as it realized the Bellman value iteration, but using a continuous-time inference network approach. Both numerical simulation and the circuit simulation show a similar behavior of the typical network convergence to the desired optimal solution based on the Bellman criterion.

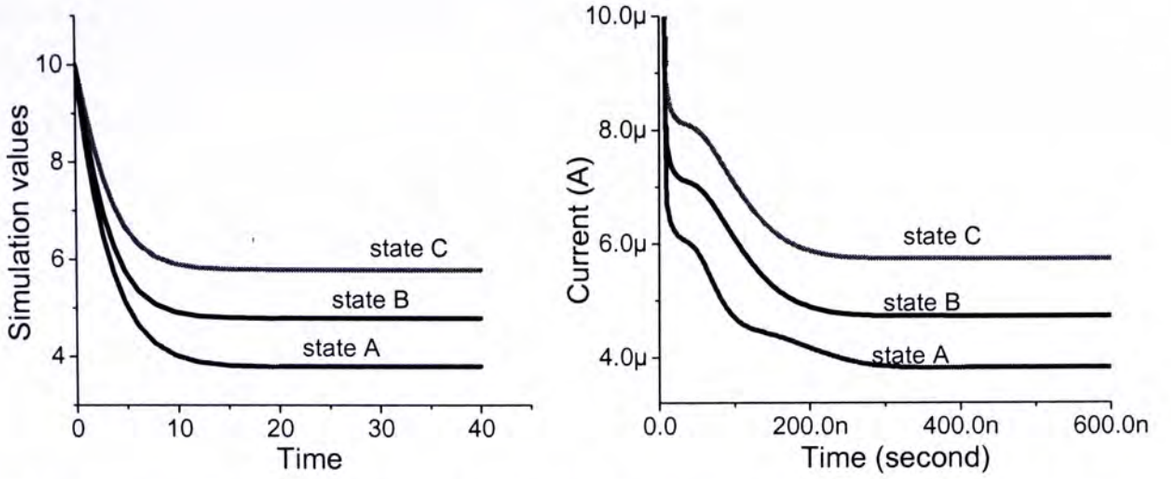


Figure C.7 (left) Numerical simulation of the 6-node single-destination inference network based on solving first-order ordinal differential equation (right) Inference network circuit simulation for the same problem. The expected value for state A, B and C (referring to Fig. 6) is [4.2, 4.8, 6.4], as shown in the left figure. In the circuit simulation, expected values are presented by current in the μA operational range. The results obtained is [3.95 μA , 4.83 μA , 6.74 μA]. Both numerical simulation and the circuit simulation show a similar behavior of the typical network convergence to the desired optimal solution based on the Bellman criterion.

From the numerical simulation in previous chapter, we know that the discount factor γ affects the convergence rate of the network. From the inference network circuit, we confirm that the discount factor would affect the conference of the inference network as shown in the numerical simulation. Fig. C.8. A similar variations are shown when comparing to the numerical simulation in previous chapter. The network with larger discount factor γ shown slower convergence rate (See γ =0.9) while a smaller discount factor shows faster convergence rate (See γ =0.1).

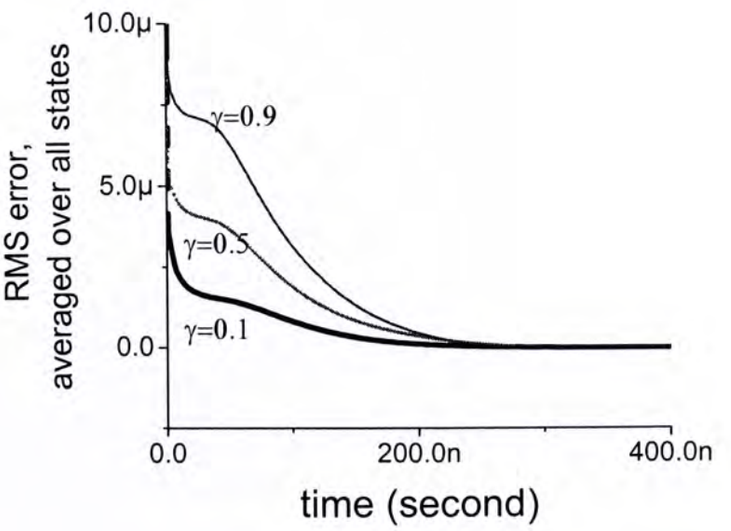


Figure C.8 Root-mean-square (RMS) errors with averaged over all states are computed for the inference network with different discount factor γ. Similar variations are shown when comparing to the numerical simulation in previous chapter. The network with larger discount factor γ shown slower convergence rate (See γ=0.9) while a smaller discount factor shows faster convergence rate (See γ=0.1).

C.4 Layout

The analog VLSI design layout for a current mirror is shown figure C.9. In order to gain high symmetric for against the defect in the fabrication process, the current mirror consists of four transistors with organized in a square shape.

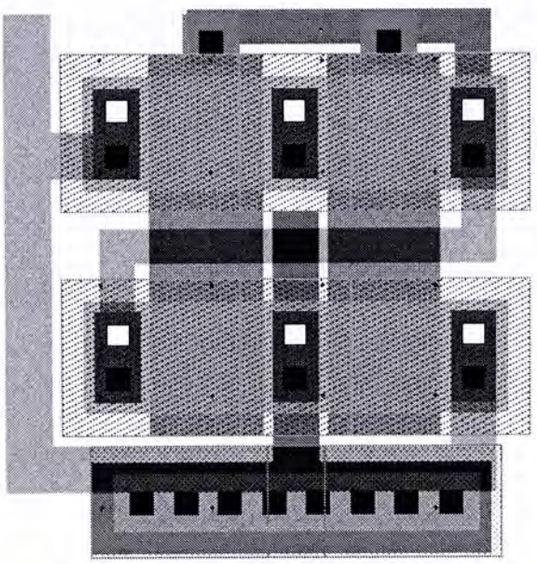


Figure C.9 Layout of a n-type current mirror

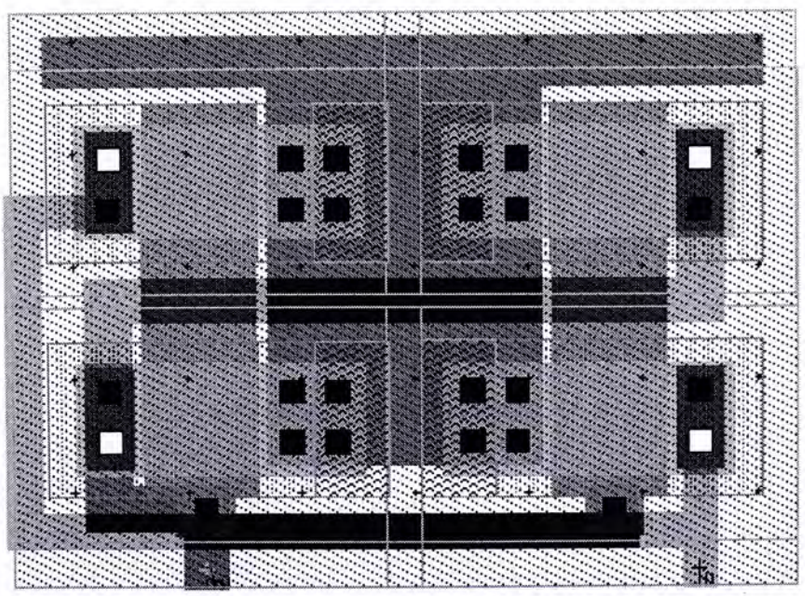


Figure C.10 Layout of a p-type current mirror

The two-input minimum operator consists of three n-type, p-type current mirror, and one p-type transistor.

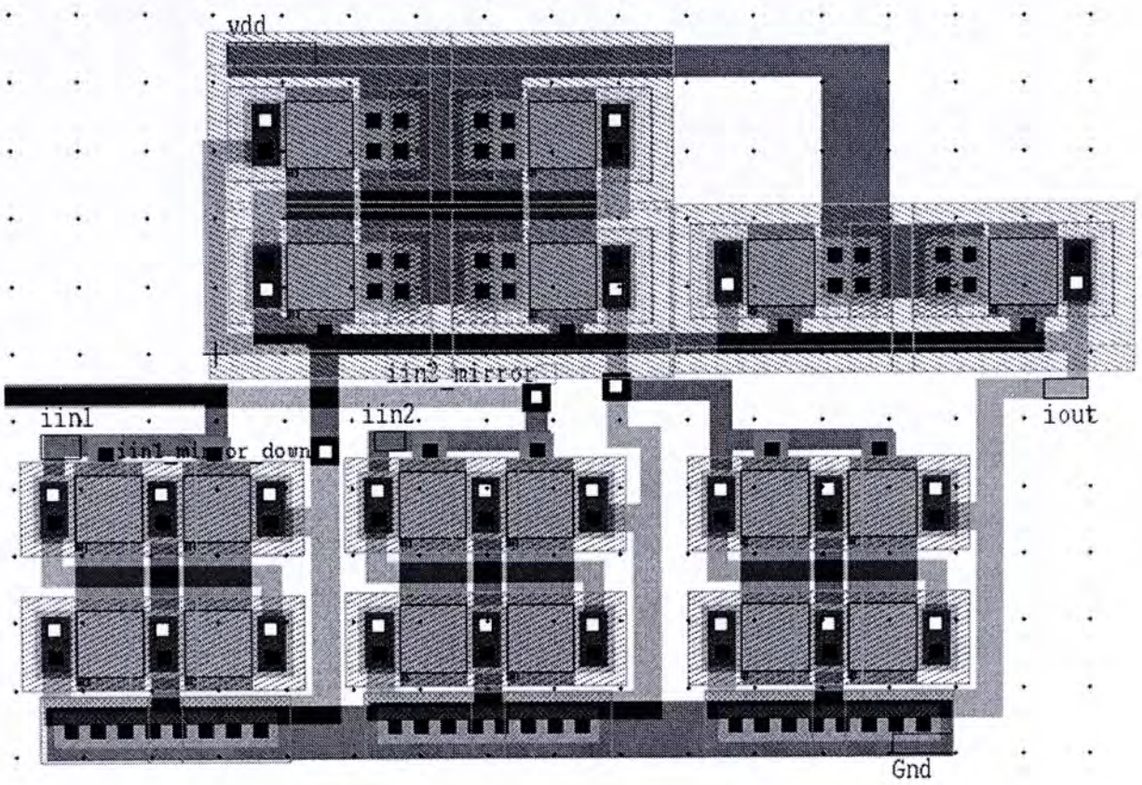


Figure C.11 The two-input minimum operator

For realizing the Binary Inference Network for shortest path problem, we have finish the layout of the circuit. For each computational unit, there are 34 transistors including both p-type and n-type. The layout is shown in C.12. The inference network is arranged vertically with the metal on the right-hand side is ground. Each computational unit is corresponding to a node from the graph in Fig. 5.12.

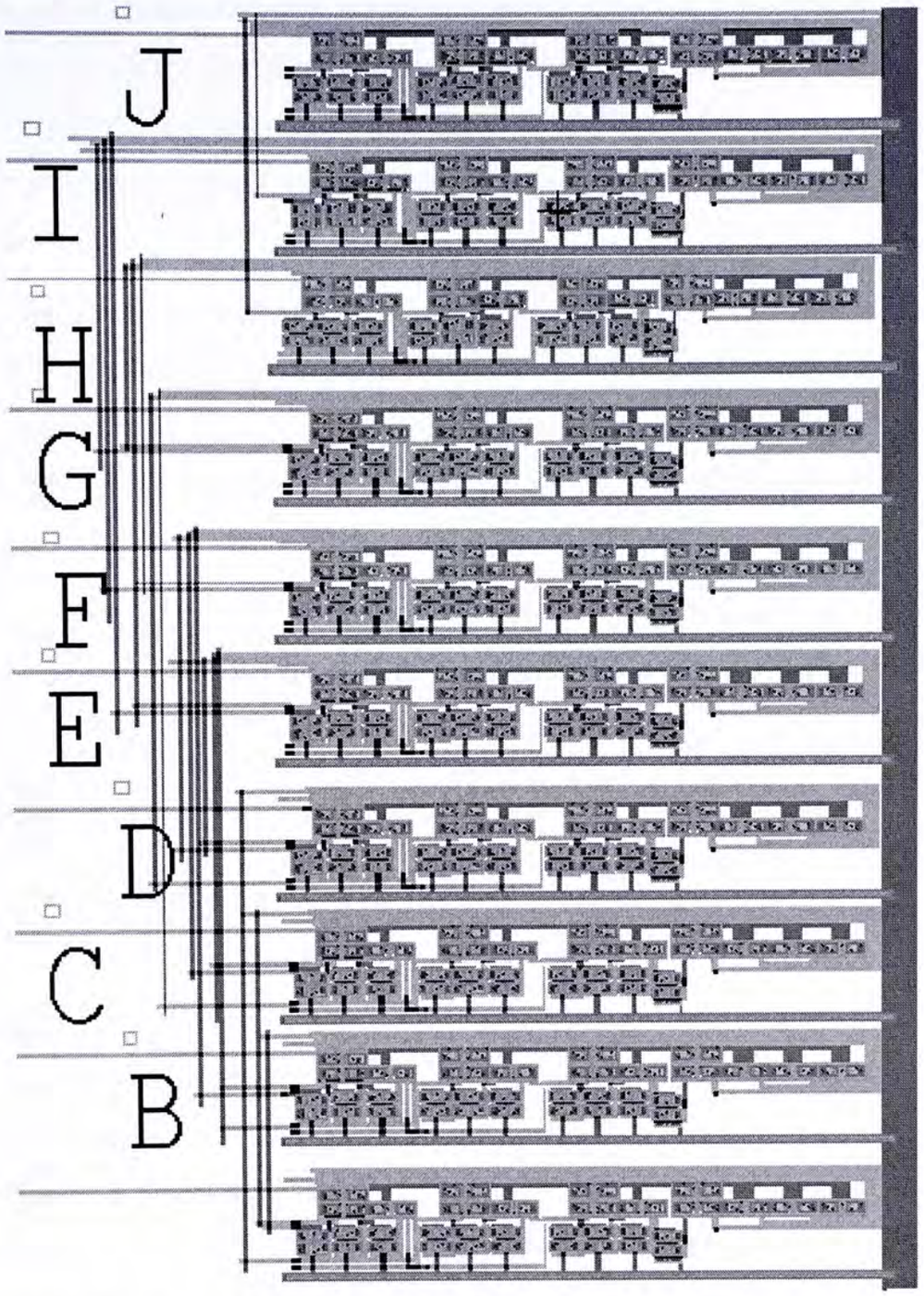


Figure C.12 Binary Relation Inference Network for 10-node Shortest Path Problem

C.5 Fabrication

The goal of this fabrication of the first stage of the design which will consist of a network of computational units that are connected in a network on a large die that has a larger number of pins, such as the 4.6mm × 4.6mm chip that contains 116 pins. This design will allow the user to build high performance computational systems for reinforcement learning and able to run large scale simulations of neural learning model in real time.

The overall design consists of about 500 MOS transistors, the vast majority measuring $8\mu m$

× 8μm each. The design includes larger transistors when used in critical nodes, such as current mirrors, to approximately 2800 μm^2 in order to achieve better matching. Since the goal of this iteration is to show proof of principle, no attempt was made to optimize transistor sizes. There are also six capacitors, each of 1pF. The main system components and their size is shown in table C.1

The BRIN for reinforcement learning is composed two separate subsystems: a shortest path circuit, and a drug addiction modeling circuits. The shortest path circuit measures 500μm × 900μm, or an area of 0.45mm^2. At this size, a conservative estimate of 20 BRIN circuits can be fit on half of a 4.6mm×4.6mm die. This is already a very large network in computational terms, and can be useful in computing shortest path for 80 cities.

Table C.1 Main system component in the circuit fabrication

	Shortest path circuit	Q-learning circuit	Whole system
Number of transistors	50 MOS	45 MOS 1pF×6 cap	~500 MOS 6 Cap
Size	240,000 μm^2	260,000 μm^2	1.5mm^2

The sum of the layout area of the learning subcircuits' is 1 mm^2. This conservative estimate takes into account wiring to the pads. The entire reinforcement learning system is approximately 1.5mm^2.

The circuits are laid out using Tanner L-Edit software in a hierarchal fashion. DRC checks are performed on each of the smaller subcircuits to make sure that no design rules are violated. The design is laid out inside an Pad frame generated by Tanner. The layout for the pad frame is shown in Figure C.13. There are five sub-circuits, current-mirrors, minimum circuit, Binary Inference Network for shortest path problem, Q-learning network circuit and one of the units from the Q-learning network, in the overall padframe layout sending for fabrication. In total,

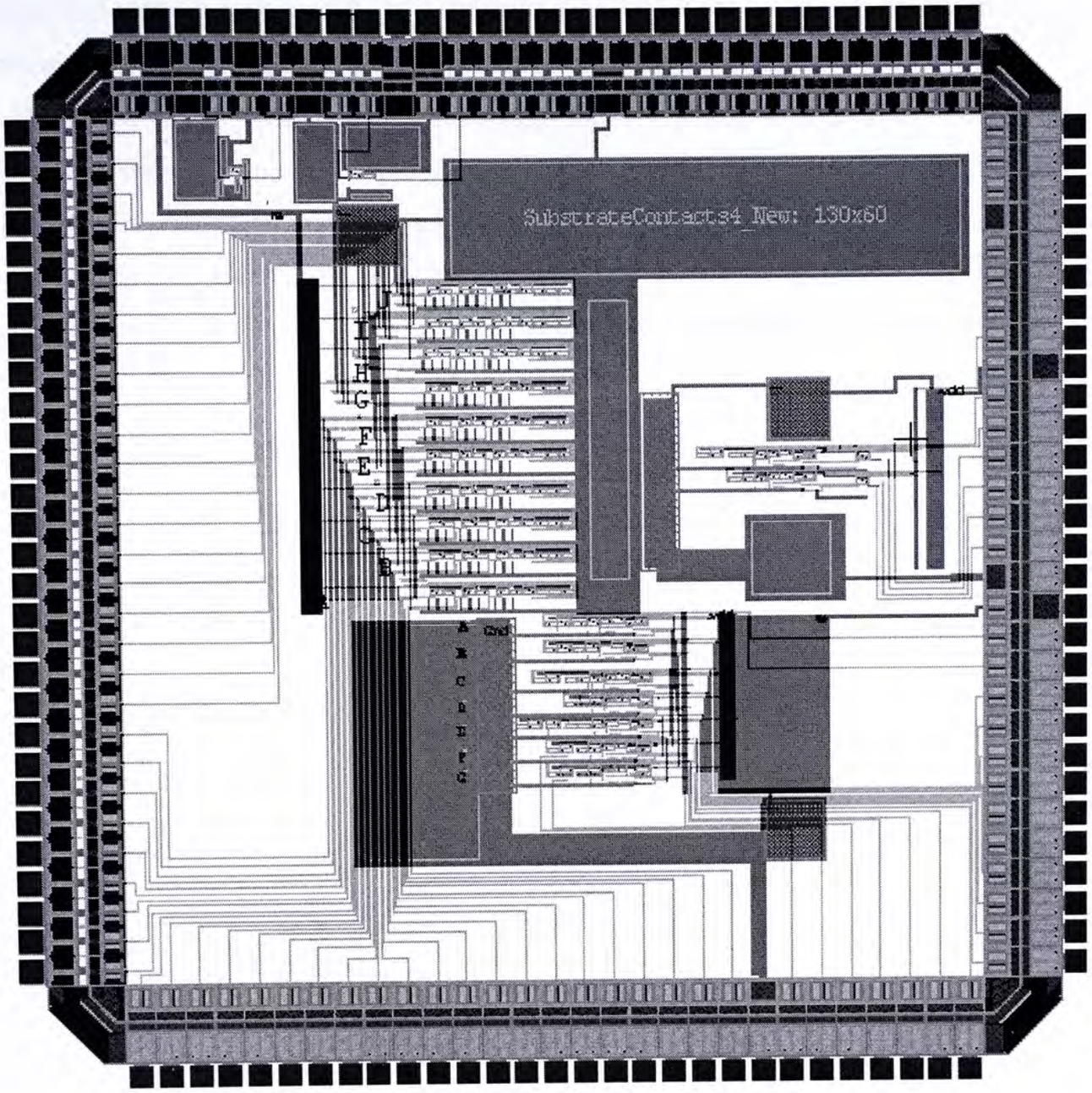


Figure C.13 Overall design with the padframe

C.5.1 Testing and Characterization

The process chosen for fabrication is the AMI 1.5 μm process. The choice of this conservative process is to address the issue of circuit performance, which will invariably differ from simulation results. Analog designs are dependent on good matching for optimal performance. In this iteration, the most critical transistors of the design are made especially big so that the ΔL and ΔW are small, which will help the circuits operate near the ideal case of perfect matching.

Simulations of the chip were performed on Tanner's T-spice and included the pad frame and other off chip circuits. All circuits have been optimized and seem to work fine. Important

layout techniques, such as the use of guard rings to protect against latchup, and optimal placement of critical nodes on the die, is implemented in this design as well.

The layout plan for this chip requires a large number of pins in order to be able to build a functional network. Special care will be taken to protect critical nodes with guard rings, although it is recognized that the 1.5μ process is not optimized for guard ring design. Careful layout techniques will be used to separate digital and analog components, as well as design larger dimension mirrors and other circuit elements to achieve good matching.