(Revised 08/14)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Keyu Ruan

Entitled
Identification of Unknown Petri net Structures from Growing Observation Sequences

For the degree of Master of Science in Electrical and Computer Engineering

Is approved by the final examining committee:

Lingxi Li

Brian King

Stanley Yung-Ping Chien

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification/Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Lingxi Li

Approved by Major Professor(s): _____

_____

Approved by: Brian King 06/08/2015

Head of the Department Graduate Program Date

IDENTIFICATION OF UNKNOWN PETRI NET STRUCTURES FROM GROWING OBSERVATION SEQUENCES

A Thesis

Submitted to the Faculty

of

Purdue University

by

Keyu Ruan

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

August 2015

Purdue University

Indianapolis, Indiana

ACKNOWLEDGMENTS

I wound thank Dr. Lingxi Li for the help in this thesis and also all the things in the past two years at IUPUI he has done for me. I also would like to thank Dr. Brian King and Dr. Stanley Chien who are my committee members in my final defense as well as great instructors in my study. I also want to thank my parents Shengrong Ruan and Liumei Di for their support in my study and life in my whole lifetime.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Ruan, Keyu M.S.E.C.E, Purdue University, August 2015. Identification of Unknown Petri net Structures from Growing Observation Sequences. Major Professor: Lingxi Li.

This thesis proposed an algorithm that can find optimized Petri nets from given observation sequences according to some rules of optimization. The basic idea of this algorithm is that although the length of the observation sequences can keep growing, we can think of the growing as periodic and algorithm deals with fixed obsevations at different time. And the algorithm developed has polynomial complexity. An segment of example code programed according to this algorithm has also been shown. Furthermore, we modify this algorithm and it can check whether a Petri net could fit the observation sequences after several steps. The modified algorithm could work in constant time. These algorithms could be used in optimization of the control systems and communication networks to simplify their structures.

1. INTRODUCTION

1.1 Background and Motivation

Discrete event dynamic systems(DEDSs) are a class of dynamic systems which are asynchronous, with event-driven state evolution. DEDSs has been developing rapidly since 1980s. A variety of theoretical models and analysis techniques from different angles have been put forward. Usually only a finite number of discrete values are taken as the state of these systems which are corresponding to possible physical conditions such as the quality of the system components and the number of other parts waiting to be delt with, or the status of other macro-management like plan making and job scheduling. The changes of these states are due to the occurrence of various events, e.g., the appearance and disappearance of certain environmental conditions and the initialization or complete of the system operations. DEDSs have been widely applied in modern world and have been increasingly important in technology areas like automatic control systems, communication networks, and transportation systems, which makes it considered as an important theoretical basis of analysis and design for large and complex information processing and control systems. How to integrate the various models and theoretical methods and form multi-level, multi-model theory system in order to fully reflect the complexity of the DEDSs and give the effective solutions to the practical problems have become the target of current study.

In this thesis, we will focus on one of the DEDSs model, Petri nets. A Petri net is one of several mathematical modeling languages for the description of distributed systems. It is a powerful method for describing dynamic systems and is especially suitable for simulating the dynamic feature of the asynchronous concurrent systems. Petri net has also been successfully used to analyze the performance of the fault tolerance of operation systems and computer system architecture.

In this thesis, we will study how to find the optimized Petri nets from given observed conditions. After introducing some notations and presenting the standard concept of Petri nets, in Chapter 3 we will first talk about the problem of finding possible markings from given observation sequences. Because it has already been shown that, if the length of the sequences is infinite, it is impossible to scan all marking in the reachability tree in polynomial time, so we need to add some limiting conditions to these sequences or the Petri net itself to make it possible to deal with this problem in polynomial time. In this thesis, we have added limits to the observation sequences. In the next part of Chapter 3 is to find V vectors which are keys to this problem. Before doing that, we need to figure out how many times the transitions in the desired Petri net have fired and the range of number of transitions to simplify this problem. The given observation sequences lead to a lot of information. Once we know the number of firing actions and transitions, we can use mathematical methods to calculate exactly how many V vectors we have. After that, we have all conditions for solving equations, and we could get the structures of Petri nets we need.

Chapter 4 will give the complete algorithm for solving this problem and practical Matlab code. We will also analyze the time complexity of this algorithm to see whether it could work as predicted in polynomial time.

In Chapter 5, we will talk about how to use the method in another way that aims at checking whether a Petri net could generate certain observation sequences. This is useful in optimizing the system describing by this Petri net in order to fit the requirement of the given conditions. A modified version of the algorithm to achieve this goal will also be proposed.

We will talk about the practical use of this algorithm in Chapter 6. As mentioned above, Petri nets have been widely used in automatic control systems and communication networks, we also could apply this algorithm in these fields. Optimized Petri nets will have the minimum number transitions and arcs, which can imply minimum number of devices and materials in practical use. This will have significant impact on time and money saving.

Before further study of optimization of structure of Petri nets, we will talk about some related studies with our subject and point out the similarity and differences between our work and theirs.

1.2 Related Work

1.2.1 Identification of unkown Petri net structures

The main purpose of this thesis is to find the unknown structure of Petri net from given observation sequences. This is actually the extension of the problem of identification of unknown Petri net structures. We could find many similar studies by other researchers. Authors in [1] have introduced a problem of identifying unobservable transitions from given system events. Besides observable system events, the given conditions also include the observable part of the Petri net. With these conditions, after running certain algorithm, the unobservable structure of the system could be generated in fast speed.

Authors in [2] have introduced a algorithm to determine whether to delete fault transitions from current existing Petri net or not. The method of doing this is to separate unobservable places and fault transitions from the regular ones. After a flow of operations, we can make conclusions from the result firing vector. Although the algorithm needs us to know the structure of the Petri net, it still could give us some ideas about checking whether a transition is valid or not to be added into the Petri net system.

Similar with the problem above, in this thesis, we also use given system events (observation sequences) to identify unknown Petri net structures. The difference is that, other than except the number of places, we don't know anything about the structure of Petri net. This is actually a challenging mission to solve this problem in polynomial time. Hense, we proposed some ideas to add limits to the given system events to simplify the problem. The advantage of identifying the whole structure is the scope of application of this method will be much larger. According to the given

conditions, what we could find is an estimation of the real Petri net. Although there could be errors between the result of our outcomes and the real structure, the method introduced in this thesis is a good try of expanding the ideas in [5]. As the method in this thesis becomes more mature, it will also be a quite useful tool in unknown system identification.

1.2.2 Marking scanning and firing sequences formation

Marking scanning looks like a simple problem and has been studied by quite a lot of researchers. Actually, it is challenging to be done in polynomial time. Authors in [3] have presented a method to scan possible markings that can fit both the given observation sequences and the structure of the given Petri net. In this paper, the length of the observation sequences is variable. The authors have shown in this case the time complexity of this algorithm will be exponential. This means if we want to lower its complexity, some limitation must be added. Authors in [4] have lowered the complexity for a little by using some observation places but the complexity is still exponential. Authors in [4] provide us a good thought of how to reduce complexity. In this thesis, we also need to do marking scanning. But here, we don't know the structure of the Petri net which will definitely make the number of markings to be scanned very large. However, this won't be a big problem the largest possible number of markings is proportional to the longest length of observation sequences-th power of the number of the transitions [3] . In order to reduce the complexity, we need to make the length of sequences be constant or using some methods to pick constant number of transitions that could be fired in every loop. The latter one needs us to come up with a way to make sure there are only a constant number of transitions are legally fired, which is quite tough work. In this thesis, we use the former method. After the length of sequences be limited, this problem becomes much simpler. The reason we can limit length of observation sequences is also reasonable, which will be introduced in later chapters. Authors in [5] proposed a problem of marking estimation with

unknown initial marking and introduced a useful tool (observer coverability graph) to help doing that. This tool could keep track of the estimation error on each place of the net. Problems that has been dealt with in this paper are different from ours. But we also deal with some structure estimation problems in later chapters and this observer coverability graph could be another solution for those problems.

The step after marking scanning will be firing sequences formation. This is also a problem has been studied by many researchers. Authors in [6] proposed an approach to estimate the firing sequences with the least cost, where each transition is associated with a nonnegative cost. In that case, finding a sequence with the least cost becomes really important and challenging. Authors in [7] have also mentioned similar problem. But here, the Petri net is associated with time delay. The author managed to come up with some methods to solve the sequence estimation problem. In this thesis, we also need firing sequences for the solving of state equations. However the formation of firing sequences won't be associated with any extra parameters. They will come from all the markings we scanned. The way to find certain ones from a whole set of markings to form a functional firing sequences is also challenging. All these will be introduced in later chapters.

1.2.3 Practical use

The problem of the practical use of Petri nets is also a topic that has been studied by many people for many years. Petri nets is a quite useful and convenient tool to simulate actual system from both structure and working state aspects. Authors in [8] have proposed the use of a device called middleware in sensor networks. Though the concept of Petri net has not been mentioned in this article, the structure of the system could be represented by Petri net. Different devices in this system could be represented by places and transitions in Petri net and the connections could be represented by arcs. Authors in [9] have talked about the practical use of Petri nets from another aspect, process flow. In this article, Petri net has been used to represent

the flow of the system. Places represent the states of the system, transitions represent the actions leading the system from state to state and arcs represent the connections between states and actions. The Petri will start at a place called "*Ready*" with one token in it. When the system works, the token will be moved from state to state and finally reach the final place called "*End session*" which marks the end of the process flow. Authors in [10] used Petri net to model the approach they proposed to detect the failure components in the system. They used places to represent failure modes and transitions to represent the conditions that lead the system to failure modes. Because the failure modes and conditions lead to failure modes are definitely unobservable, the authors also need to do structure identifications in their work that is also related to the problem in this thesis.

Those examples of practical use of Petri net has very high reference value. But we could also use our idea to help improve them. In this thesis, we will talk about the idea of using the algorithm which will be introduced in later chapters to optimize the structure of sensor networks proposed in [8] and the flow of system process mentioned in [9]. After that, we will also discuss the advantages of the optimization and the reason we need to do that.

1.3 Major contributions

The study of Petri net has significant effects on giving us deeper understanding of complex dynamical systems. Meanwhile learning its properties can give us more inspiration in solving practical problems. In this thesis, we mainly deal with two problems, that are scanning markings in polynomial time and finding optimized structure of Petri net. Our goal of is to find a method to generate a optimized Petri net in polynomial time with given observation sequences with variable length. We develop several methods to achieve that goal:(1) Adding limits to the observation sequences to make the process of marking scanning simpler. (2) Developing algorithms to scan all possible markings which could fit the given observation sequences. (3) Developing

algorithms to enumerate all V vectors that may appear for each possible number of firing actions and each number of transitions. Combining these conditions and finding the last result with minimum number of transitions and arcs. (4) Some practical use as well as the method to test whether the existed Petri net could fit the observation sequence have also been introduced to make the ideas in this thesis have more practical meanings.

1.3.1 Limits to the observation sequences

The idea of limiting the length aims at reducing the complexity of the process of the most important part, marking scanning. We assume the length of the sequences keeps growing. When we actually want to use it as input of an algorithm, we use just the existing observations when we load the sequences. At that time, the length actually is a constant no matter what will be generated later. This fact can reduce the complexity of the algorithm.

1.3.2 Marking scanning, V vectors and equation system solving

With the idea of constant length of observation sequences, the work in this part becomes simpler. The first problem has been solved is making sure the number of firing actions and the number of transitions are finite. It is impossible to try all number from one to infinite. But actually we could get finite possible markings from the given observation sequences. With the idea of fixed given sequences, we could get the range of the number of firing actions and the number of transitions with upper and lower bounds. The next problem is the maximum number of markings and V vectors. We need the upper bounds of them to stop the searching. Both of them could be found with some mathematical methods.

1.3.3 Existed Petri net Optimizing

As mentioned above, the limit of the observation sequences would cause errors in our estimation, which means the structure of Petri net would not be exactly the same with the real one. With the sequences grow, we may get more Petri nets with different structures but still maybe none of them would be the real one. In that case, instead of trying to find the exact real structure, we test the structure we have already found to check if it still could fit the current observation sequences and decide whether we need a new one that could make the Petri net work in good conditions as required. As the sequences grow, this action could be repeated as many times as we want to let the structure as accurate as it could be. This algorithm ensured the feasibility of using the idea in this thesis in practical problems.

1.4 Organization

This thesis is organized as follows. The next chapter, Chapter 2 will provide some basic concepts and mathematical background of Petri net models. Chapter 3 will give the method of scanning all possible markings which could be generated from the given observation sequences in polynomial time. The next part of Chapter 3 will show the work of finding all possible V vectors for equation system solving. Chapter 4 will provide the complete algorithm and practical code. Time complexity will also be analyzed. Chapter 5 will talk about some extention of this algorithm which could be used in optimizing existing Petri net from given observation conditions. Chapter 6 will study some practical uses of our algorithm. We conclude this thesis in Chapter 7 to make a summary of the idea and algorithm and discuss some future work directions that can be extended from this work.

2. BRIEF INTRODUCTION OF PETRI NET

2.1 Introduction

Petri net was first been proposed by Carl Adam Petri in 1960s. It is a mathematical modeling language for description of distributed systems. Petri net has both intuitive graphical expression and rigorous mathematical formulation. Both of the graphical expression and mathematical formulation of Petri net are excellent tools for practical problems solving. Graphical expression could us intuitive expressions to the target systems and mathematical formulation would make it more convenient to dealing with problems.

2.2 Basic concept of Petri net

2.2.1 Structure

Petri net N=(P,T,A,W) is a set consisted with places, tokens, transitions and weighted arcs. Here P=$\{p_1, p_2, ...p_n\}$ is a finite set of places. T=$\{t_1, t_2, ...t_n\}$ is a finite set of transitions. A $\subset$ (P$\times$T)$\cup$(T$\times$P) is a set of arcs including both from transitions to places and places to transitions. W:A $\rightarrow$ $\{1, 2, 3, ...\}$ is the weight function on the arcs. A simple example of Petri net is shown in Figure 2.1.

Places in Petri nets are marked as hollow circles. Places could not be connected with arcs to each other directly. There must be one or more transitions between every two places. The most important function of places are storing tokens which are those black dots. Those tokens would be transited from place to place through transitions.

Transitions are usually marked as short black bars. Similar with places, transitions could be connected with arcs with each other directly. There must be one or more

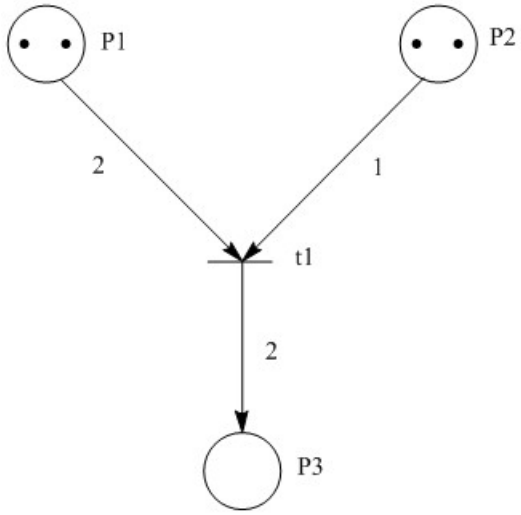

Fig. 2.1. Simple Petri net.

places between every two transitions. Different from places, tokens could not be stored in transitions. They are only tokens' hubs during transiting.

The relationship between places and transitions could also be represented by notations. We could let ${}^\bullet p_i$ (${}^\bullet t_i$) ($p_i \in$ P and $t_i \in$ T) represent the set of input transitions (places) for p_i (t_i) and let $p_i^\bullet$ ($t_i^\bullet$) represent the set of output transitions (places) for p_i (t_i). Also, ${}^\bullet p_i^\bullet = {}^\bullet p_i \cup p_i^\bullet$ (${}^\bullet t_i^\bullet = {}^\bullet t_i \cup t_i^\bullet$) could represent both input and output transitions (places) for p_i (t_i).[3]

Arcs are the directed arrows connecting places and transitions. The directions of the arcs will decide whether the arcs are input ones or output ones. Here, the concept of input and output are just for corresponding transitions. The important part of arcs is the numbers next to them which is called their weights. Those number would decide whether those tokens could successfully be transited.

Example 1 See the Petri net in Figure 1.[3]

In the Petri net in Figure 1, places are P=$\{p_1, p_2, p_3\}$, transitions T=$\{t_1\}$, arcs A=$\{(p_1 \times t_1),(p_2 \times t_1),(t_1 \times p_3)\}$ and weights W(p_1, t_1)=2, W(p_2, t_1)=1 and W(t_1, p_3)=1. And the relationship between places and transitions are represeted by $p_1^\bullet = \{t_1\}$, $p_2^\bullet = \{t_1\}$,${}^\bullet p_3 = \{t_1\}$, ${}^\bullet t_1 = \{p_1, p_2\}$, $t_1^\bullet = \{p_3\}$.

2.2.2 Marking

Marking is a column vector with elements number same as the number of places in the Petri net which is a important property of Petri net which could provide the state of Petri net at a time.. We use M_i=$[a_1, a_2, ...a_n]$ to represents markings. The integer elements in M_i corresponds to each places $\{p_1, p_2, ...p_n\}$ and represents the numbers of tokens in each places in a certain time. If there is any change happen in the net, we also could easily get to know through markings. We also use $M(_p)$ to represents marking of place p which is the number of tokens in place p. Petri net system could be denoted by $< N, M_0 >$. Here M_0 is the initial marking of the Petri net.

Example 2 In the Petri net in Figure 1 in current time, there are two tokens in p_1, two tokens in p_2 and no token in p_3. Then the marking for this Petri net now will be $M_0 = [2, 2, 0]^T$

2.2.3 Incident matrices

For the mathematical formulation of Petri net, several kinds of matrices need to be introduced: output incident matrix B^+, input incident matrix B^- and their difference, incident matrix $B = (B^+) - (B^-)$. The concepts of input and output are just for places. Incident matrix could give us a lot of information about the corresponding Petri net such as the number of places which is the number of rows, the number of transitions which the number of columns, the weight of every arc which is each element in the matrix (zero represents no connection, signs represent directions). We can see incident matrix could give us every condition we need to build a complete Petri net which means if we know the corresponding incident matrix, we could get one and only one solution of the structure of Petri net.

Example 3 The following matrices are corresponding with the Petri net in Figure 2.1.

$$\begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} 2 \\ 1 \\ -2 \end{bmatrix}$$

From up to bottom, the three matrices are input incident matrix B^-, output incident matrix B^+ and incident matrix B.

2.2.4 Enabling and firing transitions

We call the one transition t_j is enabled if the weight W of each input arc is smaller than the number of tokens in corresponding place $p_i \in P$, or in another way, $M(p_i)$ need to be no smaller than $B^-(p_i, t_j)$ for all $i \in \{1, 2, 3, ...n\}$. This could also been written as $M \geq B^-(:, t_j)$. Here $B^-(:, t_j)$ is the column vector in B^- corresponding to transition t_j. $M[t_j >$ is used to denote t_j is enable under marking M.

When a transition is enabled, it could be fired. After it is fired, same number with the weight of the corresponding arc of tokens will be removed from all input places connected to this transition. And same number with the weight of the corresponding arc of tokens will be added to all output places connected to this transition. Or in another way, $B^-(:, t_j)$ will be removed from previous marking M and then $B^+(:, t_j)$ will be added to it. So current marking will be: $M' = M - B^-(:, t_j) + B^+(:, t_j)$. We denote this firing action as M[$t_j >$M'.

Example 4 Consider the Petri net in Figure 1. The initial marking for that Petri net is $M_0 = [220]^T$. The input incident matrix $B^- = [2, 1, 0]^T$. So in this condition, $M_0 \geq B^-(:, t_1)$, transition t_1 could be enabled.

Example 5 In Petri net in Figure 1, t_1 is enabled under $[2, 2, 0]^T$ so we could fire it. During the firing, $B^-(:, t_j) = [2, 1, 0]^T$ is taken away from M_0, and $B^+(:, t_j) = [0, 0, 2]^T$ will be added to M_0. So after t_1 fired, $M_1 = [0, 1, 2]^T$ which is shown in Figure 2. This process is denoted by $M_0[t_1 > M_1$.

2.2.5 State equation

The state euqation of a Petri net is the equation shown below:

$$B * V = M_i - M_j$$

Here B represents the incident matrix of the Petri net, M_i is the state at time step i, M_j is the state at time step j and V represents the firing vectors of the Petri net. The firing vector V here is with dimension $m \times 1$ (m is the number of transitions) and with nonzero entries indicating the transition firings.

Example 6 Consider the Petri net in Figure 2.2.

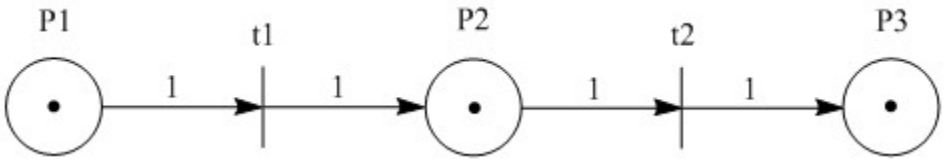

Fig. 2.2. Petri Net 2

The incident matrix is:

$$\begin{bmatrix} -1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix}$$

At the time t=0, $M_0 = [1\ 1\ 1]^T$. Transitions could be fired are t_1 and t_2. Let $M_1 = [0\ 0\ 3]^T$ at time t=1 which means t_1 has been fired once and t_2 has been fired twice. The corresponding firing vector $V = [1\ 2]^T$. Then the state equation for this condition will be:

$$\begin{bmatrix} -1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 3 \end{bmatrix}$$

Here we could see that the entries in firing vector V are actually the number of total times every transition been fired which means the V vector could also be written as the sum of all v_is (i=1,2,....) where v_i is firing vector for single firing action with only one nonzero entry. In example 6, $V = v_1 + v_2 + v_2 = [1\ 0]^T + [0\ 1]^T + [0\ 1]^T$ where v_1 represents t_1 be fired once and v_2 represents t_2 be fired once.

2.2.6 Reachability Graph

The structure of a Petri net could be more complex which would make several different transitions could be fired at the same time if we put enough tokens in places. We use a method called reachability graph to represent all the possible firing actions.

This reachability graph gives us all the possible markings we can get from a Petri net. In other word, if we want to study a finite Petri net, we don't need to know what exactly it looks like. We could just learn its reachability tree instead and then do the other work use mathematical method like state equations.

Example 7 Consider the Petri net in Figure 2.3. We could draw its corresponding reachability graph or reachability tree.

The graph in figure 3 is consisted with several markings and directed arcs weighted with the names of transitions. In this tree each arc shows a firing action and the weighted name is corresponded the fired transition. For instance, from initial marking $M_0 = [1\ 1\ 1]^T$ to marking $M_1 = [0\ 2\ 1]^T$, transition t_1 has been fired. And from initial

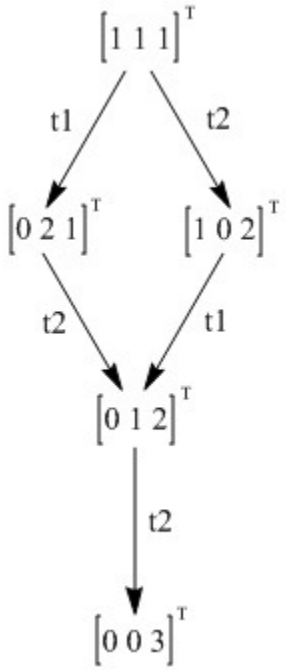

Fig. 2.3. Reachability Tree to PN2

marking $M_0 = [1\ 1\ 1]^T$ to final marking $M_K = [0\ 0\ 3]^T$, all fired transitions could form a sequence call firing sequence.

Definition 1 Let $se = t_{j1}t_{j2}....t_{jn}, n = 1, 2, ...$, this sequence of transitions is called a firing sequence between two markings M_{i1} and M_{i2} if we can start from M_{i1} and end at M_{i2} with all and only these transitions from t_{j1} to t_{jn} be fired in order.

In a reachability graph corresponding to a Petri net, sometimes we could get more than one firing sequence from initial marking M_0 to end marking M_k. In the reachability graph in Figure 3, we could get two firing sequences, $se_1 = t_1t_2t_2$ and $se_2 = t_2t_1t_2$. Though we have multiple firing sequences, the firing vector V is the same because incident matrix B and the difference of M_k and M_0 of the state equation keeps the same. This property is very important in solving problems in this thesis.

2.2.7 Sink and Source transitions

Two other concepts need to be mentioned are sink and source transitions.

Example 6 Consider the Petri net in Figure 2.4.

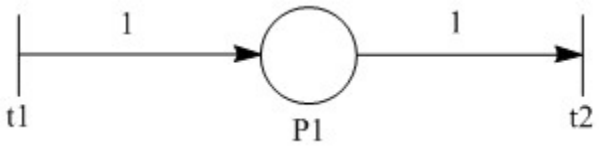

Fig. 2.4. Sink and Source transitions

In this Petri net, transition t_1 is called source transition which has no input place which means it could produce infinite tokens to its output places (here only P_1). Transition t_2 is called sink transition which has no output place which makes it could absorb infinite tokens from the input places and output nothing. Those transitions are both important to the Petri net study and its practical use. But in this paper, we need to get rid of them in order to simplify the problem.

2.3 Observation sequences

Observation sequences Os are special property used in this thesis which are important conditions for solving the problem. They could give the sequences of all token changing in all places in the Petri net.

Example 7 Consider the Petri net in figure 2 and its corresponding reachability graph. If we follow the firing sequences in the left side of the reachability graph which will be $[1\ 1\ 1]^T \rightarrow [0\ 2\ 1]^T \rightarrow [0\ 1\ 2]^T \rightarrow [0\ 0\ 3]^T$, the corresponding observation sequences would be like following:

$S_1 : 1 \rightarrow 0$

$S_2 : 1 \rightarrow 2 \rightarrow 1 \rightarrow 0$

$S_3 : 1 \rightarrow 2 \rightarrow 3$

Because the sequences are called observation sequences, they will grow only when there are token changing and if we don't knwo the exact number of places, we could only give the sequences corresponding to the places that are able be observed.

If there doesn't exist place we cannot observe, the observation sequences could actually provide more conditions besides changing of the token numbers. The most direct condition it could provide is the number of places. More details of this will be introduced in later chapters.

2.4 Summary

This chapter gives a brief introducing of the basic concepts of Petri net and Petri net language. We have talked about the structure of Petri net in graphical expression, markings, incident matrix, the concept of enabling and firing transitions, state equation, reachablity graph, sink transitions, source transitions and observation sequences. These concepts will all be used in the later chapters.

3. MARKING SCANNING AND FIRING VECTORS SEARCHING

3.1 Introduction

The problem we deal with in this paper is following. We are given a observation sequence Os of the number of tokens in each place which means we know all the changes of tokens in this Petri net. In this case, it is possible to find out what Petri net could generate this observation sequence. Our work is to creat a algorithm to achieve this goal.

To find the structure of a Petri net, we need to find the corresponding incident martix B from the state equation $B * V = M_i - M_j$. The first chanllenge is how to find the markings M_i and M_j. Doctor Lingxi Li from University of Illinois Urbana-Champaign proposed a way to find all possible firing sequences.[1] But He has also proved that it is impossible to find all sequences in polynomial time if we don't know the length of the observation sequences. So in this case, the first task of our work is to deal with the length of the observation sequences.

In order to create a algorithm for the target problem in polynomial time, our idea is to make the length of the observation sequences finite. The main idea of this is that though the length of the sequences keeps growing, it will always be a constant when we use it as input parameters in any problems. When we get to know that, we can easily solve this problem.

After finding all markings, we have half of the conditions to solve the state equation. The next step is to find all possible firing vectors V. We don't need observation sequences in this part which the method of V vector searching is a little different from marking scanning part.

Because we don't know the exact number of transitions, we also don't know the exact dimension of the V vectors. We could not create a algorithm working in polynomial time if we don't know how many targets we have. Similar with the part of marking scanning, we will introduce the method to make sure the range of number of firing vectors V. This is also the main problem we need to solve in this chapter.

Besides that, this chapter will give the method of finding firing vector V and also the algorithm of state equation solving. At last, we could get the optimized incident matrix B and the structure of Petri net we want with all the conditions we have.

3.2 Number of Markings

Because we want to find a Petri net with optimized structure from the given observation sequences, we need to find all possible markings. So we need to know how many markings we need to deal with in the worst case. Author in [11] gave a upper bound of number of consistant markings get from label observation sequences with unknown transitions. The upper bound given in that paper is polynomial in the length of the observation sequence. But different from that, our observation sequences is orgnized by number of tokens in different places. And in this thesis, the upper bound of the number of markings generated from the observation sequences is constant. The method we use to find the possible markings is similar with [12] with is proposed by the same author in [11] which is the method of enumerating, scanning all possible markings could generated from the given observation sequences.

The information we can get from the first glance of the observation sequences are the longest length of the sequences and the number of increasing arcs and decreasing arcs.

Example 8 Consider the observation sequences below.

$S_1 : 2 \to 1 \to 2 \to 3 \to 2 \to 3$

$S_2 : 1 \to 2 \to 1$

$S_3 : 1 \to 2 \to 0 \to 1$

$S_4 : 1 \rightarrow 2 \rightarrow 1 \rightarrow 0$

In this example, the longest length is the length of sequence S_1 which is five. There are seven increasing arcs: S_1: $1 \rightarrow 2$, $2 \rightarrow 3$, $2 \rightarrow 3$; S_2: $1 \rightarrow 2$; S_3: $1 \rightarrow 2$, $0 \rightarrow 1$; S_4: $1 \rightarrow 2$. And there are six decreasing arcs: S_1: $2 \rightarrow 1$, $3 \rightarrow 2$; S_2: $2 \rightarrow 1$; S_3: $2 \rightarrow 0$; S_4: $2 \rightarrow 1$, $1 \rightarrow 0$.

Proposition 1 The number of firing actions (name it Fa) should be a number between the longest length (name it L) of the observation sequences and the smaller one between the number of increasing arcs (name it L_i) and number of decreasing arcs (name it L_d).

Proof: The number of firing actions Fa should be at least equals the longest length of the observation sequences to make it possible for the observation sequences to make sense. So the lower bound of Fa should be L. Because the increasing arcs in the sequences represent tokens have been transmitted into corresponding places and decreasing arcs in the sequences represent tokens have been transmitted out of corresponding places, considering there could be multiple transitions connected to the same place, the smaller one of L_i and L_d would be more closer to the actual number of firing actions Fa. What has been proved after many practical tests is $min(L_i, L_d)$ will always be larger than L. So $min(L_i, L_d)$ should be the upper bound of Fa.

Remark 1 In this paper, we ignore the transitions have not been fired and the Petri net with self loop. In that case, the number of transitions (name it m) should be no larger than the number of firing actions Fa and should be larger than one.

The range of number of transitions m is ensured by Remark 1. The lower bound is one because we need at least at least one transition in Petri net. The upper bound is Fa because we need to make sure each transition could be fired at least once. In that case, if $m > Fa$, there will always be transition(s) would not be fired. Thus, $1 \leq m \leq Fa$.

With Proposition 1 and Remark 1, we could analyse the structure of corresponding Petri net in different conditions. For every possible $Fa - m$ pair, we could find all markings from the observation sequences. The total number of all possible markings for a fixed $Fa - m$ pair could be calculated.

Proposition 2 Mark the number of increasing arcs during one firing action as n_{in}. And also mark the number of decreasing arcs during one firing action as n_{de}. In the worst case, the number of markings during one firing action should equal to

$$(n_{in} + \frac{n_{in}(n_{in}-1)}{2!} + ... + \frac{n_{in}!}{n_{in}!}) * (n_{de} + \frac{n_{de}(n_{de}-1)}{2!} + ... + \frac{n_{de}!}{n_{de}!})$$

The process to get the result in Proposition 2 is actually a process of mathematical calculation. Now we try to prove it.

Proof: Assume we are given observation sequences with n places. The largest length of the sequences is L. Then we assume during every firing action there are n_{in} increasing arcs and n_{de} decreasing arcs. Note that $n_{in} + n_{de} = n$ because the sum of all increasings and decreasings should equal to the number of places. And because of this equation $n_{in} + n_{de} = n$, we can know the range of both n_{in} and n_{de} are $[1\ n-1]$ because we could not fire transitions if during one firing action we have only increasing or decreasing arcs.

Then, we can use the method of permutations and combinations to find all possible combinations of the increasing and decreasing arcs. In the worst case, any combinations of increasing and decreasing arcs could be possible. In that case, there are n_{in} kinds ways to pick one increasing arc from all possibilities, $\frac{n_{in}(n_{in}-1)}{2!}$ ways to pick two increasing arcs from all possibilities,....only one way to pick all increasing arcs. Sum up all the possible ways, for increasing arcs, there are $(n_{in} + \frac{n_{in}(n_{in}-1)}{2!} + ... + \frac{n_{in}!}{n_{in}!})$ ways for selection in total. For decreasing arcs, it's the same idea. So at last the number of possible combinations of increasing arc and decreasing arc will be the product, $(n_{in} + \frac{n_{in}(n_{in}-1)}{2!} + ... + \frac{n_{in}!}{n_{in}!}) * (n_{de} + \frac{n_{de}(n_{de}-1)}{2!} + ... + \frac{n_{de}!}{n_{de}!})$. Prove done.

Proposition 2 could give us the range of marking number during one firing action. And for the whole process, in the worst case, we could have the same number of possible markings. So we can get the total number of markings by multiply the largest length L with the the number of markings during one firing action. As long as we get the total number of markings, we know the range of how many loops we need to do for the traversing of all markings which make it possible to find all possible markings.

3.3 Markings and firing sequences

From the last section, we have already had enough information in finding all the possible markings. The part of finding is actually really simple which is just visit each marking once and store it. For some details, during every loop, we need to scan all the markings in the order of possible combinations of increasing and decreasing arcs from $1-1$ pair to $n_{in}-n_{de}$ pair which are $1-1,\ 1-2, \ldots .\ 1-n_{de},\ 2-1,\ 2-2, \ldots\ 2-n_{de}, \ldots\ n_{in}-1,\ n_{in}-2, \ldots\ n_{in}-n_{de}$. During this work, we would have duplication. Ignore those duplicated ones and start the next loop.

After we done this work for the total number of loops' times, stop and store the markings we get for later use. After that, we are ready to form firing sequences. We need to know a whole sequence to solve a state equation system. The method to get firing sequences are also simple. Find all the final markings in the markings we found, and then keep looking for the previous marking of the current one until we meet the initial markings. All the markings we have visited during that process consist one possible firing sequecne.

3.4 Range of number of firing actions and transitions

Similar with marking scanning, we also need to find the range of number of how many V vectors we have in order to make a algorithm working in polynomial time. To get to know that, we need to figure out two things, the dimension of firing vector

V which is also the number of transitions and the number of how many times the transitions have been fired which is also the number of firing actions. The number of firing actions Fa has already been given in last chapter of marking scanning which should be a number between the longest length L of the observation sequences and the smaller one between the number of increasing arcs L_i and number of decreasing arcs L_d. Let $Fa_{lower} = min(min(L_i, L_d), L)$ and $Fa_{upper} = max(min(L_i, L_d), L)$, then $Fa_{lower} \leq Fa \leq Fa_{upper}$.

And about the number of transitions, we could also know its range from the number of firing action Fa.

Proposition 3 The number of transitions m should be a number between one and the number of firing actions or,

$$1 \leq m \leq Fa$$

. This definition could only work in the conditions where all transitions have been fired at least once. In other word, we ignore all the unobservable transitions.

Proof: The lower bound equals one is obvious. We will always need at least one transition in a Petri net. About the upper bound, because in our hypothesis every transition in this Petri net have been fired at least once, So in order to achieve that, we could not get transitions more than the number of firing actions Fa. Else we would always have at least one transition has not been fired which is a contradiction to the hypothesis. So in that case, the definition of range of number of transitions m is correct. Prove done. After we know the range of those two parameters, we could know the number of firing vectors V.

3.5 Finding possible V vectors

We have already known the possible number of total firing actions and transitions in the last several parts. Because the elements in V vector represent how many times each corresponding transition be fired, the sum of all elements in V should equals to the number of total firing actions Fa. Known that, the problem of finding V vectors has been changed to figuring out all the possible ways to distribute firing actions Fa to transitions m. To get deeper understanding for this concept, we will use a simple example here to illustrate it.

Example 9 Consider the following situation.

Assume we have six firing actions which means $Fa = 6$. And the transition number equals five which means $m = 5$. We can treat $Fa = 1 + 1 + 1 + 1 + 1 + 1$, each one represents one firing action. And we can also treat $m = 1 + 1 + 1 + 1 + 1$, each one represents one transition assigning position. Separate the six ones in Fa to six parts and assign them into the five positions in m. We can use the method of permutations and combinations here. There are $5{*}\mathrm{A}_5^5$ ways for the assignment in total. In other word, there are $5{*}\mathrm{A}_5^5$ kinds of V vectors.

Because we have mentioned that we ignore all the transitions have not been fired to simplify our problem, all the elements in the V vector should be larger than zero. That means when we distribute the firing actions to transitions, we should not leave zero there and every entry in firing vector V should at least be equal to one. In that case, each transition assigning position should be assigned at least once. One necessary step in the process of finding all V vectors is duplication removing. This is because the ones to be assigned from Fa don't have any differences from each other. Assign them to m will definitely cause duplication. But it's easy to deal with by scanning all the existed V vectors and compare to the new generated one in algorithm. After finishing the step of generate firing vector V, we have all we need to solve state equation $B * V = M_1 - M - M2$.

3.6 Equation system solving

After we have all conditions we need, it's time to solve the equation system $B*V = M_1 - M_2$. The first thing we need to do before that is to pair V vector with the difference of markings $M_1 - M_2$ with the same dimension one by one. We don't know all the entries in incident matrix B which means we have $m * n$ variables. Each state equation $B * V = M_1 - M_2$ could provides us m equations. In that case, we need n state equations can we actually get the solution martix B.

What need to be noticed is because the paring is random, the system is not necessary to have a solution. What's more, the elements in B matrix represents the weight of each arcs in Perti net which should all be constant, so non-constant B would not be accepted. In that case, not every combination of vector-difference pair $M_1 - M_2$ and firing vector V could generate a useful B matrix. But that's also not a big problem in algorithm. Every time we find a rational incident matrix B, record it with corresponding transition number m.

Keep doing the work above for several loops. The number of total loops could be got from the maximum number of markings and firing vectors V calculated above in the worst case. After all rational incident matrices B found, we could start the optimization. Compare all matrices' corresponding transition number (m) and choose the smallest one. In all the B matrices with the minimum transitions, compare the zero numbers in the matrices and choose the maximum one. The non-zero number represents connection in Petri net. So most zeros in incident matrix B means least arcs in Petri net which could be a way to optimize the structure of the net. If we have multiply matrices with the same transition number m and zero number, they are all optimized.

Now we have the smallest m and incident matrix B with most zeros. That is the optimized solution for this problem in current condition which could be used in many practical situations.

3.7 Summary

In this chapter, we have finished the first and most important part of the problem, finding all markings. To deal with the markings, we learned the given observation sequences deeply and find a lot of useful information that could help solving the problem. What's more, we proposed an idea of giving limit to the length of observation sequence in order to solve the problem in polynomial time. This section could make good fundamental for solving the whole problem.

Then we discussed the method to find all possible firing vectors V. The key point for that is to make sure how many V vectors we have. The way to do that is use mathematical method to generate the number of V vectors from the number of transitions and the number of total firing actions. After the searching of V vectors, we also talked about the solving of state equation $B * V = M_1 - M_2$ and the optimization of the incident matrix B get from the result of the equations. During the process of equation solving, we have several problems need to notice. For the optimization after equation solving, we could achieve our target to get the optimized incident matrix B following some rules.

4. ALGORITHM AND COMPLEXITY ANALYSIS

4.1 Introduction

In this chapter, we will give the complete algorithm for the whole problem. After that we will analyze the complexity of this algorithm and figure out why it can work in polynomial time. At last, some parts of the Matlab code written with this algorithm will also be discussed.

4.2 Complete Algorithm

Algorithm 1

Input: Observation sequences with length L, fixed number places n and fixed initial marking M_0.

Output: Optimized B matrix and minimum transition number m.

1: Load the observation sequences and store it as a matrix M.
2: Assign -1 to those empty entries in M.
3: Calculate matrix T. Each entry in T is the number of one-step token changing from the observation sequences.
4: Find the upper range of Fa.
5: $min(min(L_i, L_d), L) \rightarrow Fa_l$
6: $max(min(L_i, L_d), L) \rightarrow Fa_h$
7: $Fa_l \leq Fa \leq Fa_h$
8: **for** Fa=1 to Fa_h **do**
9: Scan the number of markings on the last row of firing action.
10: **for** num=1 to the last of the markings on the last row **do**
11: Assign current markings to Mtemp, corresponding T to Ttemp.

```
12:     Separate the first column of Ttemp into positive part P and negative part N
13:     Do permutation to the indices of the vectors P and N. Name the new per-
        mutation matrices we get as PermP and PermN
14:     for i=1 to end of the columns of PermP do
15:       for j=1 to end of the columns of PermN do
16:         Select the first i columns of PermP and the first j columns of PermN.
17:         Do some operations to get rid of duplicated rows.
18:         for ii=1 to the last row of PermP do
19:           for jj=1 to the last row of PermN do
20:             Do shifting to all the entries in Mtemp and Ttemp corresponding
                to the indices in current row of PermP and PermN.
21:             Store the shifted Mtemp, Ttemp and the first column of Mtemp
                which is one of the possible markings in the first blank slot of next
                row with respect to unshifted Mtemp in the last row.
22:             Also store the location where we store the first column of corre-
                sponding Mtemp in the last row.
23:             if The first column of current Mtemp equals to the final markings
                and the entries of the second column of Ttemp is all -1 then
24:               Mark this column as final marking and store it.
25:             end if
26:           end for
27:         end for
28:       end for
29:     end for
30:   end for
31: end for
32: while There still has final markings exsiting in all the markings. do
33:   Find one final marking from all the markings we have.
```

Follow the previous location we stored to find all markings until we reached the initial marking.
Store all the markings during this search in a row and we have formed a marking sequence.
Mark the final marking we found this time as regular marking.
end while
for Fa=Fa_l to Fa_h **do**
  for m=1 to Fa **do**
    Prepare a zero vector V has length equals to m.
    Assign all entries in this vector V one and $Rest \leq Fa - m$
    if $Rest = 0$ **then**
      Store V as a sum firing vector.
    else
      Prepare a cell VP with each element in the first row be vector with length equals $Rest$ and all entries equal to one
      $Vrow \leftarrow 2$
      for Vlevel=1 to $Rest$ **do**
        $Start \leftarrow 1$
        for Vmulti=1 to m **do**
          At each value of Vmulti, loop for $m^{(Vlevel-1)}$ times
          if Vlevel=Rest **then**
            $Finish \leftarrow Start$
          else
            $Finish \leftarrow Start + m^{(Rest-Vlevel)} - 1$
          end if
          Find the vector elements in cell VP with index between $Start$ and $Finish$, multiply current $Vmulti$ to the $Vlevel$-th entry in those vectors and then store them in the $Vrow$-th row of VP
          $Start \leftarrow m^{(Rest-Vlevel)} + Start$

58: **if** Vlevel=Rest **then**
59: $Finish \leftarrow Start$
60: **else**
61: $Finish \leftarrow Start + m^{(Rest-Vlevel)} - 1$
62: **end if**
63: **end for**
64: $Vrow = Vrow + 1$
65: **end for**
66: $PermV \leftarrow thelastrowofVP$
67: **for** iii=1 to the last column of $PermV$ **do**
68: Add one to the entries in V corresponding to the value of elements in the vector from current slot of $PermV$.
69: Store V as a sum firing vector.
70: **end for**
71: **end if**
72: **end for**
73: **end for**
74: **for** num_v=1 to the last sum firing vector we have **do**
75: Prepare a empty vector VP_2.
76: Add x elements equals to the index of current sum firing vector if the corresponding entry has value x.
77: Do permutation to VP_2.
78: **for** iiii=1 to the last row of VP_2 **do**
79: Form one-step firing vectors with only one entry equals to one in the order according to the elements in current row of VP_2.
80: Store these one-step firing vectors as a firing sequences.
81: **end for**
82: **end for**
83: **for** k=1 to the last marking sequence **do**

84: **for** h=1 to the last firing sequence **do**
85: **if** They have the same firing action **then**
86: Pair these two sequences and use the first m elements to calculate incident matrix B.
87: **end if**
88: **if** This B has non integer entries **then**
89: BREAK
90: **else**
91: **if** This B could satisfy the state equations formed by the rest elements from these two sequences **then**
92: Store this B
93: **end if**
94: **end if**
95: **end for**
96: **end for**
97: Select the B_{least} with least column from all the Bs stored.
98: Select the B_{opt} with most zero entries from all the B_{least}s stored.
99: Output these B_{opt}s.

4.3 Complexity analysis

As mentioned, this algorithm works in polynomial time. In the part of marking scanning from line 8 to line 16, we will loop for enough times to ensure we could find all markings. We know in the worst case, there are $(n_{in} + \frac{n_{in}(n_{in}-1)}{2!} + ... + \frac{n_{in}!}{n_{in}!}) * (n_{de} + \frac{n_{de}(n_{de}-1)}{2!} + ... + \frac{n_{de}!}{n_{de}!})$ markings. So this part, the time complexity is constant because n_{in} and n_{de} are all constant if the input observation sequences are fixed. In the part of finding all possible firing vectors V from line 17 to line 35, we will also loop for enough times to ensure we could find all V vectors. We use a table with dimension $(Fa - m) \times m^{Fa-m}$. m has range $1 \leq m \leq Fa$ which means the worst case for the

dimension of this table is $(Fa-1) \times m^{Fa-1}$. Fa has constant range so visiting all elements in this table and do some simple calculation will take polynomial time. In the part of solving equation system, we have m equations and $m * n$ unknowns in the worst case for a certain $V - (M_k - M_l)$ pair. So we need n V vectors to get a possible B matrix. Solving a equation system with n unknowns has time complexity $O(n^3)$. So in this part, the time complexity is $O((m * n)^3)$. The last part of finding the optimized B matrix and transitions m, the worst case is every equation system in the previous part could generate a rational B matrix which still be a constant number because the number of B matrix equals one n-th of the number V vectors which is polynomial. The time complexity of comparing all the B matrices is also a constant. So in this case, the total time complexity will be polynomial. So this algorithm could work in polynomial time.

4.4 Matlab code analysis

According to the algorithm we had in the previous section, we could practically write some code using some program platform. In this thesis, we will use Matlab to do that. The actual code will be attached in appendix and we will only discuss the thought and process of programming. The same with the algorithm, the actual code also has been divided into several parts.

4.4.1 Preparation

The first thing we need to do is loading the observations to Matlab. To load the sequences, we need to first convert the sequences into matrix. Because the length of each sequence corresponding to each place are not even, here we use the method of adding negative one to the end of those sequences which don't have enough entries. The reason of using negative number is that in observations we won't have negative numbers which makes these negative ones could be used as flags to jump out of the loops. After loaded, name this matrix M. Figure 4.1 shows the flow in this part.

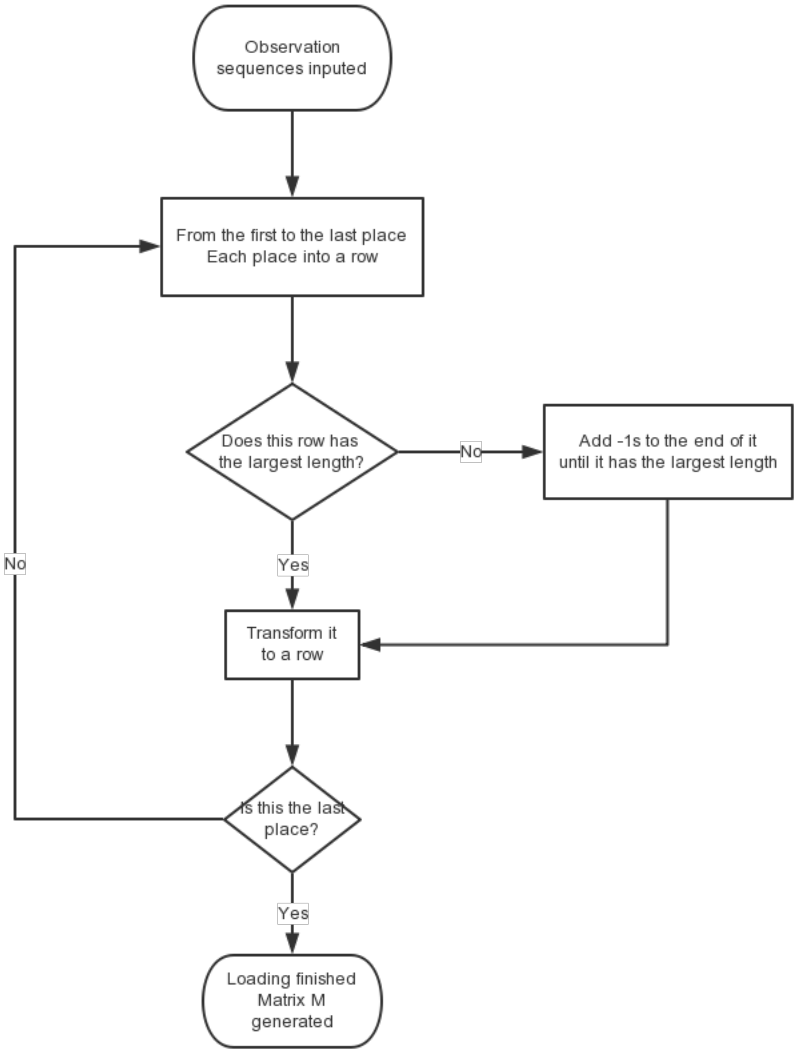

Fig. 4.1. Flow of observation sequcnes loading

Then we will deal with the increasing and decreasing arcs. As we have already loaded the observation sequences as matrix M, we could use that to represent the properties of increasing and decreasing of the arcs. Use those non-negative numbers in M to subtract the element at the left hand side of them in the same row. For those rows which don't have enough non-negative numbers, we add 0.1s to the end of the rows. Similar with the reason to use negative number for M, we use decimals

here because there will not exist decimal in weights of arc in Petri net. So we could use those decimals as flags too. After all the calculations done, we will have a new matrix T with all integer entries represent the weights of arcs. Then we can count the number of positive and negative entries which also means the number of increasing and decreasing arcs.

The next step to be done for preparation is to ensure initial and final markings. Initial marking is easy to find which is first column of loaded matrix. And to find final marking, we need to as negative ones introduced above as flag to find the actual last element of each row and combine them as a column with dimension number of places times one. This new combined column vector will be final marking. After doing this, we have done the preparation part.

4.4.2 Markings and firing sequences scanning

The part of marking and firing sequences scanning is the most important and most complex part in the whole code. We need to first find all possible markings. Here we find the markings by the step of each firing action which is to find all markings after one firing action. And all markings will be separated to several levels corresponding to firing actions. The total number of levels is the number of firing actions. To ensure we won't miss any marking, we use the largest possible value of firing actions which is the smaller one between number of increasing and decreasing arcs.

At each level, when pairing increasing and decreasing arcs, we need to consider the situation of multiple places connected to the same transition. So in order to find all possible markings, we need to try different combinations with different numbers of positive and negative entries from T matrix. Note that we will only need to care about the first column of T matrix when do firing action. Then count the total number of positive and negative entries and list all permutations of both of them. The listing action will form two new matrices P and N which contain all possible permutations of positive entries and negative entries from the first column of T matrix.

After the operation of duplication removing, we can pick the first x columns of P matrix to represent we connect x places as output to one transition and pick the first y columns of N matrix to represent we connect y places as input to the same transition. Each row of P and N has the information of which places will have tokens changing which means those places are selected to be connected. After one pairing operation done, store the position where we store current marking which is the first column of M in another cell. This operation will help forming firing sequences later. Then shift corresponding rows of M and T one column left and assign -1 to the end of M and 0,1 to the end of T. Store current M, T and the first column of M which is one of the markings in this level in cells starting from the first column. Keep doing these operations until we reach the last row in both P and N matrices. One level done. Notice that when we store M, T, the markings and the pre-position, the index where we store them should be the same to make it possible for us to find them together.

To do the same operations for the next level, we will pick the markings, Ms and Ts from the first column to the end. Now the markings we stored in the last level will be the previous markings for the markings we need to store in this level. Do the same operations of pairing, shifting and storing until we reach the bottom of P and N matrices. Notice for the new level, we need to store those matrices and markings in the next rows of corresponding cells. We assume the number of firing actions is maximum which is the smaller one between the number of increasing and decreasing arcs. But we usually we won't have that much firing actions for every firing sequences. In order to jump of the loop, we will use the -1s and 0.1s we added in M and T as flag. When we follow the process to shift the entries in M and T, if the next entry is -1 in M or is 0.1 in T, we know we reached the end of this sequences. Here we have two different situations. If the marking before shifting equals the final marking and the marking after shifting has entries all -1, we has a good sequence we need. Else, this sequence is a bad one and has to be abandoned. When we store final marking, we add one row with element 0 to the end of it to make sure we can recognize this is a final marking. Figure 4.2 below shows the flow of operations in this part.

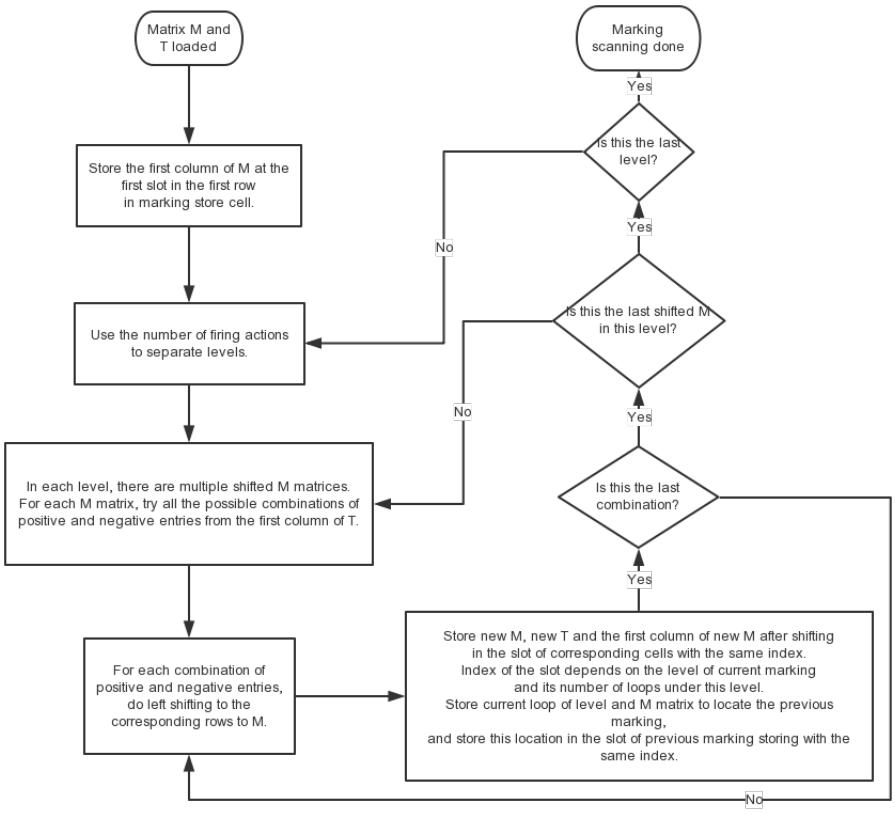

Fig. 4.2. Flow of marking scanning

After all the work above, we have completed the step of marking scanning. But that is not enough for the final equation system solving. We also need to know which markings are in the same sequence. Remember we has stored the previous marking positions of every marking in another cell with the same index with each marking. We can follow that to find previous markings for all markings. We should start with finding one final marking which is pretty simple because every final marking is one row longer than the others. After finding one final marking, follow the positions to find previous markings. When we find a previous marking is located at the first row and first column of the marking storing cell, we reached the initial marking which means we finished finding one firing sequence. Storing the markings in this sequence as a row in the sequence storing cell and cut the extra row of the checked final marking

off to make it won't be selected any more. Keep doing these operations until there is no final marking with one row longer which means we have all sequences we need. Figure 4.3 below shows the flow of operations in this part.

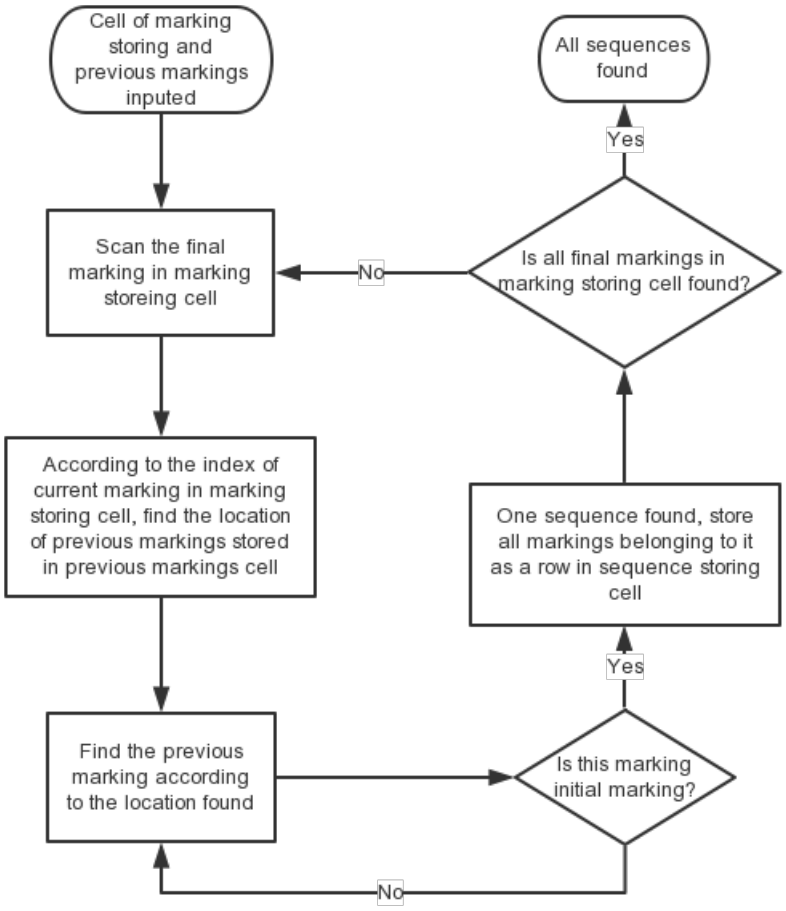

Fig. 4.3. Flow of sequences scanning

4.4.3 Firing vectors distribution

The method to find firing vectors used to be as following. Distribute the ones in firing action to slots of entries in firing vector. According to our algorithm, the number of firing action Fa will always be no smaller than the number of transitions

m which is also the number of entries in firing vector V. So we first get m ones out of Fa and assign one to each entry of V to satisfy the requirement that all transitions should be fired at least once. Note that if we have $Fa = m$, we have already get a possible distribution way. The step will be jump to the next loop. Only when $Fa > m$ can we continue the following part. When $Fa > m$, we need to distribute the rest $Fa - m$ ones and use the method of permutations again. One difference with similar work we did above is, in this part duplication of entries when distribute ones is allowed which means at the worst case, we could have all $Fa - m$ ones assigned to the same entry. In that case, we prepare a vector with entry from 1 to m all duplicated for $Fa - m$ times and then do permutation. We get a new matrix after that and name it V_{perm}. Select the first $Fa - m$ columns of V_{perm}. Those columns will form a matrix called V_{select} and each row of it corresponds to a way for the assignment. Select a row from V_{select} and assign a one to each entry in V corresponding to the number in this selected row. After that we will get a possible firing vector V. Store it in V_{store} cell, in the row with index same as the number of current number of transitions. Keep doing the selection from the first to the last row of V_{select} and store all the firing vectors we get in V_{store} cell in a row.

Though the method above could give us correct result, we could found that when the length of observation sequences become large enough, the number of different permutations will be quite huge. So we developed another method to solve this problem. First let prepare a cell VP with all elements in the first row be vectors with dimension $1 \times (Fa - m)$ and with all entries equal to one. Similar to the method above, Fa must be larger than m. Then $Fa - m \geq 1$. Do loop $Vlevel$ equals 1 to Fa-m. At each $Vlevel$ loop, do another loop $Vmulti$ equals 1 to m. Loop each loop of $Vmulti$ for another $m^{(Vlevel-1)}$ times which aims at scanning all elements in last row of VP. Then we need to initialize two parameters $Start$ and $Finish$. The initial value of $Start$ is simple with is one. The initial value of $Finish$ is a little more complex. Here we only give the result and will explain later. If current $Vlevel$ equals $Fa - m$, initial value of $Finish$ equals $Start$, else it equals $Start + m^{(Fa-m-Vlevel)-1}$.

Then multiply the value of current $Vmulti$ to the $Vlevel$-th entry of each element in current row of VP with index between $Start$ and $Finish$. After that re-assign value to $Start$ and $Finish$. Here new $Start$ equals $m^{(Fa-m-Vlevel)}$ plus current $Start$. The new value of $Finish$ also need to be separated into two different conditions but are the same with the initialization conditions and values. When we reach the last loop of $Vmulti$, before starting the next loop of $Vlevel$, move to the next row of VP and all the elements we get from the next loop of $Vlevel$ will be stored in the next row. Then we explain why the value of $Finish$ equals to $Start$ when $Vlevel$ equals $Fa-m$. The method we used here to find all possible ways to do distribution is jumping different columns at each level of $Vlevel$ to do multiplication with could ensure that we could actually list all possible distribution ways. The number of columns we need to jump at each level is $m^{(Fa-m-Vlevel)}$. When $Vlevel$ doesn't equal to $Fa-m$, there are more than one elements in this interval so that $Finish$ will be $m^{(Fa-m-Vlevel)}-1$ lager than $Start$. But when $Vlevel$ equals $Fa-m$, there is only one element in this interval, then $Finish$ has to be the same with $Start$. That is the reason why we select the value of $Finish$ like that in the previous content. After all these operations we also could get the same result with the method above. But we could save much more time and storage space than the previous one.

The operations above has only finished the work under a certain number of firing action and transition. What we need is all possible firing vectors under all possible number of firing actions and transitions. In that case, we use two levels of loops to realize that. The first level is the loop of firing actions from length of observation sequences to the smaller one between the number of increasing and decreasing arcs. The second level is the loop of number of transitions from one to current number of firing actions. In that case, in order to store all firing vectors we have and make us could easily find the right vector when we need, we expand the cell V_{store} where we store firing vectors to three dimensional. The first dimension corresponds to the rows which is also treated as the number of transitions to make it easier to find corresponding vectors, the second slot corresponds to the actual slots to store Vs

and the third dimension will correspond the number of firing actions. In this way of storing, we can find a vector with its corresponding number of transitions and firing actions easily and will save us much trouble in the part of equation solving. Figure 4.4 below shows the flow in this part.

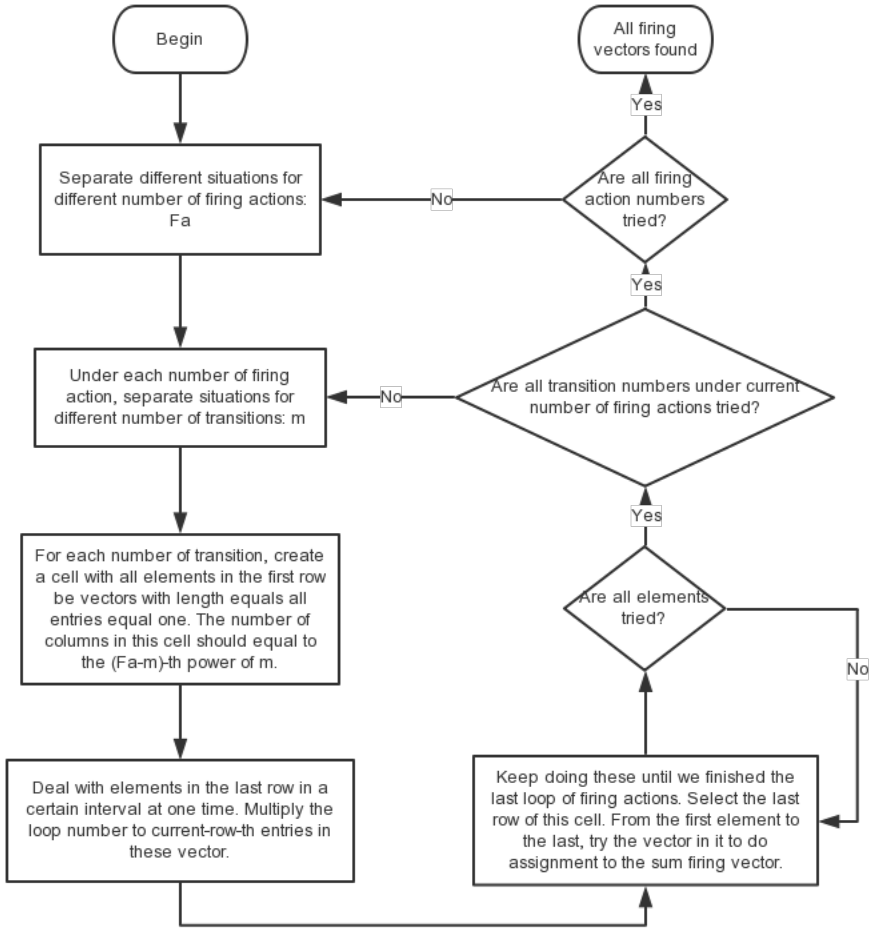

Fig. 4.4. Flow of vector distribution

4.4.4 Equation solving with result optimization

We have got a set of firing sequences and a set of firing vectors from previous subsections. In this part, we will do the equation solving. Note that in a equation system $B * V = M_i - M_j$, there are $m \times n$ variables where m is the number of transitions and n is the number of places. These variables are the entries in incident matrix B. But from one state equation system, we can only have m equations which is decided by firing vector V. Because of that we need n state equation systems to make it possible to solve all variables in B.

Fortunately, we can get many information from the set of firing sequences and the set of firing vectors. Select a row of firing vectors and a row of firing sequences with the same number of firing actions and transitions. Pick the first n firing vectors and $n + 1$ markings from the sequences. Calculate the differences between each two markings next to each other and get n difference vectors. These $2 \times n$ vectors are all column vectors. Combine all the n firing vectors into a $m \times n$ matrix V_{com}. Do the same thing to those difference vectors and get a $n \times n$ matrix Md_{com}. Then we can get a incident matrix B by use the command Md_{com}/V_{com}.

But the work is not finished after we get this incident matrix. Usually we will have extra firing vectors and markings in the same firing sequences. We need to test all of them using the B we just calculated to make sure this matrix could satisfy all the firing actions existed. If it could not satisfy any of these firing actions, this combination of set of firing vectors and set of firing sequences won't be able to generate a functional Petri net. Because according to the properties of Petri net, the entries in incident matrix could only be integer. In that case, if we get B matrices with even only one non-integer entry, we have to abandon it and start the next loop. Another possible situation is the number of firing actions is smaller than the number of places. Then we won't be able to have enough equations to solve the problem. We could just use number of firing actions that is larger than the number of places to solve this.

When we have had a incident matrix that could satisfy the whole firing sequences, store it into a slot of a three dimensional matrix B_{store}. Keep doing these operations until we reach the bottom of both firing vector set and firing sequences set. The flow of this part is shown in Figure 4.5.

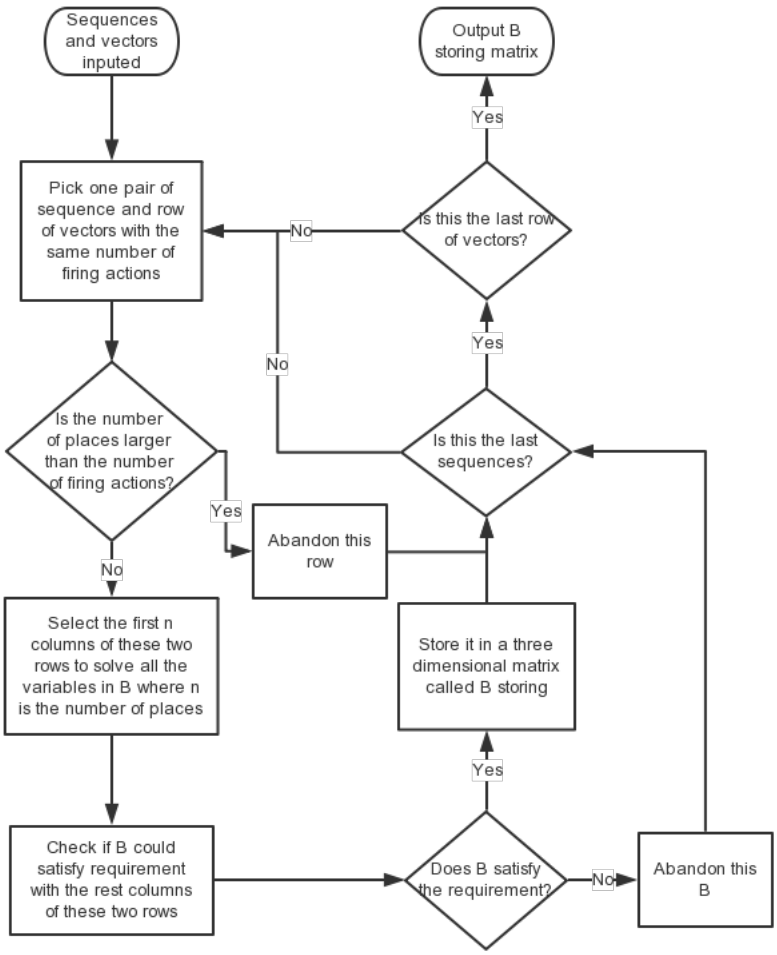

Fig. 4.5. Flow of equation solving

After we found all functional incident matrices, the next step is to compare them and find the optimized ones. The requirement of optimization is having the least number of transitions and for those that have the same number of transitions which equals to the least number, having the most number of zero entries. We could first

scan the dimension of all the B matrices stored in B_{store} to know their corresponding number of transitions and get the minimum number. Then pick out those Bs with the least transitions. After that, count the number of zeros in all the Bs and get the minimum number. Now we can know which matrices have the least zeros which means their structures is most simplified. We could have multiple matrices here that have the same least number of transitions and the same most number of zeros. Then, all of them could satisfy our requirement of optimization. Till here, the whole code has been completed. The flow of this part is shown in Figure 4.6.

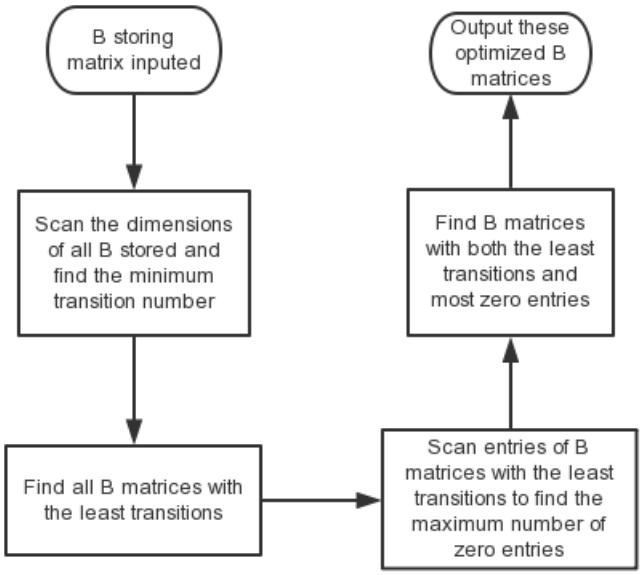

Fig. 4.6. Flow of result optimization

4.5 Simple example

4.5.1 Markings and sequences

Now we use a simple example to illustrate this algorithm in this section. We are given observation sequences which shown below.

$S_1 : 2 \rightarrow 1 \rightarrow 0$

$S_2 : 1 \rightarrow 0$

$S_3 : 0 \rightarrow 1 \rightarrow 2 \rightarrow 3$

First we count the largest length L. Here $L = 3$. Then find the number of increasing arcs L_i and decreasing arcs L_d where $L_i = 2$ and $L_d = 2$. So the number of firing actions which should satisfy $L \leq Fa \leq min(L_i, L_d)$ could only equal to 3. This means we have only one loop for Fa.

Then we do some preparations of marking scanning. Prepare five cells to store information: $MatrixM, MatrixT, Markings, PreMarkings$ and *incomp*. Store the initial marking $[2\ 1\ 0]^T$ to the first entry of $Markings$. Store "None" to the first entry of $PreMarking$. Store following matrix to the first entry of $MatrixM$.

$$MatrixM(1,1) = \begin{bmatrix} 2 & 1 & 0 & -1 \\ 1 & 0 & -1 & -1 \\ 0 & 1 & 2 & 3 \end{bmatrix}$$

Store following matrix to the first entry of $MatrixT$.

$$MatrixT(1,1) = \begin{bmatrix} -1 & -1 & 0.1 \\ -1 & 0.1 & 0.1 \\ 1 & 1 & 1 \end{bmatrix}$$

The upper level has only one marking which is the initial marking $[2\ 1\ 0]^T$. So we have only one loop of upper level markings for the beginning. Then we will do permutation for the indices of positive and negative entries in the first column of current corresponding matrix in cell Matrix T. For positive ones, we have two kinds of permutations, $[1\ 2]^T$ and $[2\ 1]^T$. We first deal with shifting one place at a time.

Do shifting for both of the two kinds of permutations and get two shifted incomplete matrices with only negative part done. These two matrices will be stored in the second row of cell *incomp* which is short for Incomplete.

$$incomp(2,1) = \begin{bmatrix} 1 & 0 & -1 & -1 \\ 1 & 0 & -1 & -1 \\ 0 & 1 & 2 & 3 \end{bmatrix} incomp(2,2) = \begin{bmatrix} 2 & 1 & 0 & -1 \\ 0 & -1 & -1 & -1 \\ 0 & 1 & 2 & 3 \end{bmatrix}$$

Then use these incomplete matrices combine with the positive part permutations to finish the shifting work. Here negative part has only one kind of permutation with is [1]. So there will two kinds of completed shifted matrices which will be stored in cells $MatrixM$.

$$MatrixM(2,1) = \begin{bmatrix} 1 & 0 & -1 & -1 \\ 1 & 0 & -1 & -1 \\ 1 & 2 & 3 & -1 \end{bmatrix} MatrixM(2,2) = \begin{bmatrix} 2 & 1 & 0 & -1 \\ 0 & -1 & -1 & -1 \\ 1 & 2 & 3 & -1 \end{bmatrix}$$

Corresponding T matrices will also be stored into cell $MatrixT$.

$$MatrixT(2,1) = \begin{bmatrix} -1 & 0.1 & 0.1 \\ -1 & 0.1 & 0.1 \\ 1 & 1 & 0.1 \end{bmatrix} MatrixT(2,2) = \begin{bmatrix} -1 & -1 & 0.1 \\ 0.1 & 0.1 & 0.1 \\ 1 & 1 & 0.1 \end{bmatrix}$$

And also the current markings which are the first columns of the elements in $MatrixM$.

$$Marking(2,1) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} Marking(2,2) = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}$$

At last, we need to record the locations of the previous markings to current markings. Because this is the beginning of all the operations, so the previous markings here are all initial markings and the location will be $[1\ 1]^T$ in cell $Markings$.

$$PreLocation(2,1) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad PreLocation(2,2) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

We can find that the corresponding elements in these cells have the same indices which could make us find a group of information easily. Keep doing this for all permutations, all number of transitions and all number of firing actions(here just one possibility) and we will have the final *Markings* cell with all markings we want. The completed *Markings* cell is shown in Table 4.1.

Table 4.1. Completed Cell *Markings*

	A	B	C	D	E
1	$[2\ 1\ 0]^T$				
2	$[1\ 1\ 1]^T$	$[2\ 0\ 1]^T$	$[1\ 0\ 1]^T$		
3	$[0\ 1\ 2]^T$	$[1\ 0\ 2]^T$	$[0\ 0\ 2]^T$	$[1\ 0\ 2]^T$	$[0\ 0\ 2]^T$
4	$[0\ 0\ 3\ 0]^T$	$[0\ 0\ 3\ 0]^T$	"*Invalid*"	$[0\ 0\ 3\ 0]^T$	"*Invalid*"

Note that the element below those two $[0\ 0\ 2]^T$s are marked as "Invilid" which should be abandoned and won't appear in real program results. We add it here to illustrate the work more clearly. The "Invilid" elements means at it's step, the marking before shifting was not the final marking, and after we do shifting, we will have -1 at the first column of corresponding M matrix. That means in this sequence, we won't reach the final marking $[0\ 0\ 3]^T$ with the given observation sequences. So this marking will be deleted from the *Markings* cell. Those $[0\ 0\ 3\ 0]^T$s are the final markings. They are one column longer than the regular markings which aims to recognize them more easily. From them, we follow the records in cell *PreMarkings* to find the completed sequences which are shown below.

$$Sequence\ 1: \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0 \\ 3 \end{bmatrix}$$

$$Sequence\ 2: \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0 \\ 3 \end{bmatrix}$$

$$Sequence\ 3: \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0 \\ 3 \end{bmatrix}$$

4.5.2 Firing vector distribution

We will only give the example of the second method of firing vector distribution which is functioned by jumping in elements and multiplication. First we need to create a three-dimensional cell to store firing vectors. We will talk about this cell later. Then we should make sure how many levels of loop we have. First level should be the loop of Fa, the number of firing actions. At each loop of Fa, the second level should be m, the number of transitions. Ensured that, the next step is to figure out how should we do assignment to V, the firing vectors. At each loop of m, the length of index vectors are all different.

We will continue the examples from the last part. Fa has only one value 3 so there is only one Fa loop. The number of transitions m could vary between 1 and $Fa = 3$ and we will start will $m = 1$. First we create a cell Vl_store with all elements in the first row be vectors with length equals $Fa - m = 2$ and have all one entries. The number of elements should be $m^{Fa-m} = 1$. So in this condition, there is only one element in the first row of Vl_store like the cell in Table 4.2.

Table 4.2. Initialization of Vl_store when Fa=3, m=1

	A
1	[1 1]

Then we start our operations from the second row of Vl_store. From here, we will deal with the elements in the last row with the column index in the interval with length $m^(Fa - m - Vlevel)$ where $Vlevel$ is the variable of the loop from 1 to $Fa - m = 2$ and num_m is the variable of the loop from 1 to $m = 1$. For example, in the second row, $num_m = 1$, $Vlevel = 1$ which makes the length of interval become 1. Then we only deal with the element in $Vl_store(1, 1)$. Multiply the first entry of the vector in $Vl_store(1, 1) = [1\ 1]$ by current $num_m = 1$ and we still get [1 1]. Store it in the location of $Vl_store(2, 1)$. Because $m = 1$, it seems we have finished the work for the loop of num_m But actually we need to keep doing the num_m loop for $m^(Vlevel-1)$ times at each value of num_m to make it possible to visit each element in the last row once when the number of columns growing larger. Though here, we only have one element in each row and $m^(Vlevel - 1) = 1$ which let this operation makes no differences. Then we have finished one loop of num_m and will move to the next row of Vl_store at the beginning of each loop of $Vlevel$. Now $Vlevel = 2$, $num_m = 1$ and $m^(Fa - m - Vlevel) = 1$. We will deal with the element in $Vl_store(2, 1)$ which is [1 1]. Multiply current $num_m = 1$ to the second entry and we will still get [1 1]. Store it in the location $Vl_store(3, 1)$. Then we get a Vl_store like the one in Table 4.3. This part has been done.

It is not obvious with all the work we need to do in this part with such small dimension of Vl_{store}. Then we assume we have $m = 3$ and $Fa = 5$ which makes $Fa - m = 2$. With these conditions, after we doing initialization to Vl_{store} we will have the cell in Table 4.4.

Table 4.3. Final version of Vl_{store}

	A
1	[1 1]
2	[1 1]
3	[1 1]

Table 4.4. Initialization of Vl_{store} when Fa=5, m=3

	1	2	3	4	5	6	7	8	9
1	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]

Then at row 2 and the loop of $Vlevel = 1, num_m = 1$, interval length equals $m^{(Fa-m-Vlevel)} = 3$ and we deal with the elements in row 3 in a group. Loop each value of num_m for $m^{(Vlevel-1)} = 1$ times. Multiply the first entries in the first three vectors by current $num_m = 1$ store them and get the cell in Table 4.5.

Table 4.5. Step 1 at row 2

	1	2	3	4	5	6	7	8	9
1	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]
2	[1 1]	[1 1]	[1 1]						

Then we come to the second loop of num_m when $num_m = 2$. Interval length still equals 3 but the we will start at the fourth element in row 1. Multiply the first entries in the fourth, fifth and sixth vectors by current $num_m = 2$, store them and get the cell in Table 4.6.

Table 4.6. Step 2 at row 2

	1	2	3	4	5	6	7	8	9
1	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]
2	[1 1]	[1 1]	[1 1]	[2 1]	[2 1]	[2 1]			

Do the same thing at the loop of $num_m = 3$, we will finish the loop of $Vlevel = 1$ and get the cell in Table 4.7. After moving to the row 3 at loop $Vlevel = 2$, interval length $m^(Fa - m - Vlevel) = 1$ and loop time $m^(Vlevel - 1) = 3$ which means we need to deal with elements in row 2 one by one and loop each value of num_m for three times. After all these operations, we will get our final version of Vl_store with $Fa = 5$ and $m = 3$ shown in Table 4.8.

Table 4.7. Step 3 at row 2

	1	2	3	4	5	6	7	8	9
1	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]
2	[1 1]	[1 1]	[1 1]	[2 1]	[2 1]	[2 1]	[3 1]	[3 1]	[3 1]

Table 4.8. Final result of Vl_{store} when Fa=5, m=3

	1	2	3	4	5	6	7	8	9
1	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]	[1 1]
2	[1 1]	[1 1]	[1 1]	[2 1]	[2 1]	[2 1]	[3 1]	[3 1]	[3 1]
3	[1 1]	[1 2]	[1 3]	[2 1]	[2 2]	[2 3]	[2 1]	[2 2]	[2 3]

After we get all the elements in cell Vl_store after last loop of $Vlevel$, the last row will be the vectors will want. We will provide the completed Vl_store cell and used for the later calculation which is shown in Table 4.9. Elements in the second row of this cell tell us which entry in V we should do assignment to. With the instruction, we could have two kinds of firing vectors.

Table 4.9. Final version of Vl_store when Fa=3, m=2

	1	2
1	1	1
2	1	2

$$V_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} V_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Do the same thing to the loop of $m = 3$, we will another possible firing vector $V_3 = [1\ 1\ 1]^T$. Store all the vectors we get till now in a matrix in rows corresponding to their transition numbers.

Those firing vectors are not enough for the state equation system solving because they are the sum of those one-step firing vectors and could only generate only one group of equations. So we need to do permutation again to generate all possible situations for one possible firing vector we get here. Same with the permutation work above, we will use index vectors to do that.

Now the length of index vector should equal to current Fa. Choose sum firing vector $V_2 = [2\ 1]^T$ as example, current $Fa = 3$, so the index vector should have dimension 1×3. Then, elements in the vector should be one of the indices exists in sum firing vector V_2 which are 1 and 2. Here because the first entry of V_2 is 2, the element 1 could appear twice in the index vector. Thus, corresponding index vector $VI = [1\ 1\ 2]^T$. Then do permutation to it and we get the permutation matrix $PermMat$

$$PermMat = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 1 \\ 2 & 1 & 1 \end{bmatrix}$$

$PermMat$ will provide us the way to separate the sum firing vectors into one-step firing vectors with could actually help solving the state equation system. And now we can explain why we need the cell $Vectors$ to be five dimensional. Because we when choose a pair of marking sequence and firing vector sequence, we need to make sure they are compatible with each other in Fa and m. At the same time, we also need to make sure we have go through all possibilities. To ensure that, this three-dimensional cell $Vectors$ has been introduced. The third dimension is corresponding to a certain sum firing vector, the other two dimensions will be used to store one-step firing vectors. Note that each row in a two-dimensional cell corresponding to a sum firing vector represents a actual firing sequences. Another important thing is the number of firing actions for every firing sequence. This could be known by scanning the length of each row which is easy to do. As we cleared all these above, we could continue with the following work. Table 4.10 shows the some part of cell $Vectors$.

Table 4.10. $V(:,:,2)$ of cell $Vectors$

	A	B	C
1	$[1\ 0]^T$	$[1\ 0]^T$	$[0\ 1]^T$
2	$[1\ 0]^T$	$[0\ 1]^T$	$[1\ 0]^T$
3	$[0\ 1]^T$	$[1\ 0]^T$	$[1\ 0]^T$

Keep doing these operations until all loops of Fa and m are done. Pair those marking sequences and firing sequences with the same Fa and use the first n marking differences and one-step firing vectors to get the solution of incident matrix B. If we pair $Sequence$ 1 with $V(1,:,2)$, we will get following matrix B_1. But if we pair $Sequence$ 1 with $V(2,:,2)$ or $V(2,:,3)$, there will be no solution for the state equation

system and we will abandon this pair and go on to the next one. If we pair *Sequence* 2 with $V(2,:,2)$, we will get matrix B_2. And if we pair *Sequence* 1 with $V(1,:,4) = \{[1\ 0\ 0]^T, [0\ 1\ 0]^T, [0\ 0\ 1]^T\}$, we will get following matrix B_3.

$$B_1 = \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 1 \end{bmatrix} B_2 = \begin{bmatrix} 0 & -1 \\ -1 & 0 \\ 1 & 1 \end{bmatrix} B_3 = \begin{bmatrix} -1 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 1 & 1 \end{bmatrix}$$

Keep doing the pairing, we could get all the integer incident matrices. The next step is to find the optimized ones. From above, we can see B_2 has one more transition than B_1 and B_2. So according to our rules of optimization, B_3 should be excluded. Then compare the number of zeros in B_1 and B_2. Both are two. So B_1 and B_2 are both optimized results.

4.6 Result

With all the flow analyzed above, we could have results with given observation sequences. We will use the sequences below as the input.

$S_1 : 2 \to 1 \to 0$

$S_2 : 1 \to 0$

$S_3 : 0 \to 1 \to 2 \to 3$

The sequences will be stored in text file as matrix in the same path with the Matlab code. And the result generated at each step has been listed below from Figure 4.7 to Figure 4.13.

If we use observation sequences with longer length as input, we could get more information in time costs. Some different samples with different length input has been shown in Table 4.11 below. With the data in Table 4.3, we could also draw a chart to show how the code perform in time complexity with different length of input sequences more intuitive. The result of that has been shown in Figure 4.14.

Marking ×

Marking <4x5 cell>

	1	2	3	4	5
1	[2;1;0]	[]	[]	[]	[]
2	[1;0;1]	[1;1;1]	[2;0;1]	[]	[]
3	[0;0;2]	[0;0;2]	[0;1;2]	[1;0;2]	[1;0;2]
4	[0;0;3]	[0;0;3]	[0;0;3]	[]	[]

Fig. 4.7. Result of marking scanning

Sequences ×

Sequences <3x4 cell>

	1	2	3	4
1	[0;0;3]	[0;1;2]	[1;1;1]	[2;1;0]
2	[0;0;3]	[1;0;2]	[1;1;1]	[2;1;0]
3	[0;0;3]	[1;0;2]	[2;0;1]	[2;1;0]

Fig. 4.8. Result of sequences formation

V_store ×

V_store <3x2 cell>

	1	2
1	3	[]
2	[2;1]	[1;2]
3	[1;1;1]	[]

Fig. 4.9. Result of sum vector searching

Table 4.11. Time cost in different length

Length	3	4	5	6	7	8
Time cost(s)	0.00877	0.04401	1.08690	30.47965	339.94708	14582.21290

V_step_store(:,:,1) =

[1]	[1]	[1]
[]	[]	[]
[]	[]	[]
[]	[]	[]
[]	[]	[]
[]	[]	[]

V_step_store(:,:,2) =

[2x1 double]	[2x1 double]	[2x1 double]
[2x1 double]	[2x1 double]	[2x1 double]
[2x1 double]	[2x1 double]	[2x1 double]
[]	[]	[]
[]	[]	[]
[]	[]	[]

V_step_store(:,:,3) =

[2x1 double]	[2x1 double]	[2x1 double]
[2x1 double]	[2x1 double]	[2x1 double]
[2x1 double]	[2x1 double]	[2x1 double]
[]	[]	[]
[]	[]	[]
[]	[]	[]

V_step_store(:,:,4) =

[3x1 double]	[3x1 double]	[3x1 double]
[3x1 double]	[3x1 double]	[3x1 double]
[3x1 double]	[3x1 double]	[3x1 double]
[3x1 double]	[3x1 double]	[3x1 double]
[3x1 double]	[3x1 double]	[3x1 double]
[3x1 double]	[3x1 double]	[3x1 double]

Fig. 4.10. Result of single step firing sequences searching

V_step_store_show

V_step_store_show <6x3 cell>

	1	2	3
1	[1;0;0]	[0;1;0]	[0;0;1]
2	[1;0;0]	[0;0;1]	[0;1;0]
3	[0;1;0]	[1;0;0]	[0;0;1]
4	[0;1;0]	[0;0;1]	[1;0;0]
5	[0;0;1]	[1;0;0]	[0;1;0]
6	[0;0;1]	[0;1;0]	[1;0;0]

Fig. 4.11. Part of the cell storing firing sequences

Bstore

Bstore <1x24 cell>

	1	2	3	4	5	6
1	[-1.0000,3....	[3.1402e-1...	[0,-1,-1;-1,...	[0,-1,-1;-1,...	[-1,0,-1;0,-...	[-1,0,-1;0,-...

Fig. 4.12. Part of the result of incident matrices before optimization searching

Bopt ×

Bopt <1x2 cell>

	1	2
1	[-1,0;0,-1;1,1]	[0,-1;-1,0;1,1]

Fig. 4.13. Final result: Optimized incident matrices

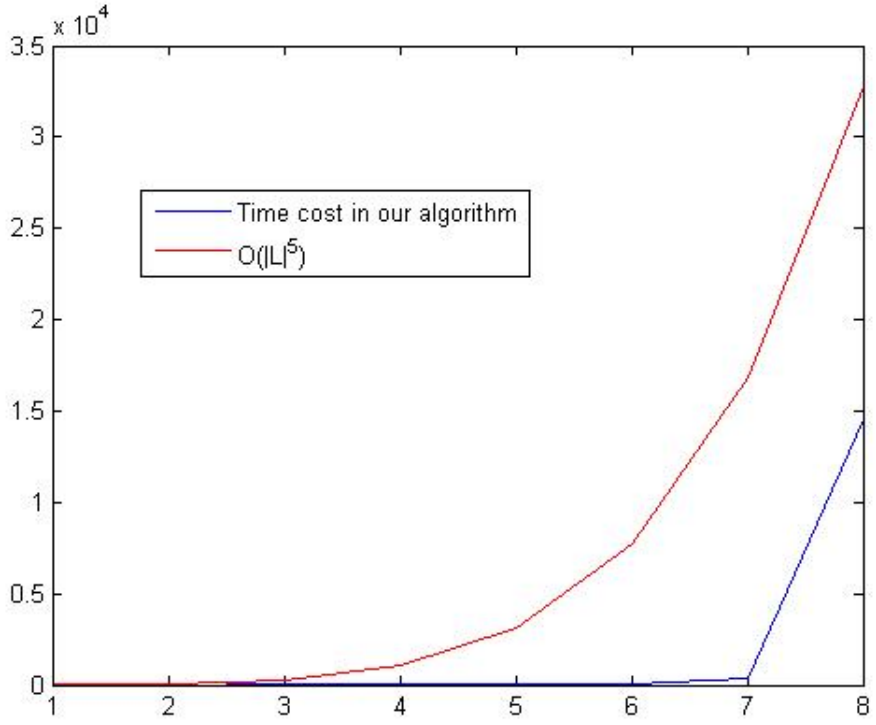

Fig. 4.14. Plot of time cost in different observation sequences length

It's easy to find that the code written under our algorithm has better performance in time cost which is another proof of the excellence of it. The tests of time costs are operated in a laptop equipped with a 2.50 GHz processor, 8GB RAM and Matlab R2013a.

4.7 Summary

This chapter gives the complete algorithm that solves the problem in this thesis. This algorithm use given observation sequences as input and could generate the optimized incident matrix of Petri net. After that we analyzed the complexity of it and proved it could actually work in polynomial time. Then, we showed the process to realize the algorithm with Matlab which has been divided into four parts: preparation, markings and firing sequences scanning, firing vectors distributing and equation solving with result optimization. In the part of preparation, we did the work of observation sequences loading, initial and final marking searching and number of increasing and decreasing arcs counting. In the part of markings and firing sequences scanning, we scanned all the markings that could be used to form firing sequences and formed a set of firing sequences using the method of storing previous markings. In the part of firing vectors distributing, we built all possible firing vectors using the relationship between number of firing actions Fa and number of transitions m. In the part of equation solving with result optimization, we solved the state equation systems using the information we get from previous part and have found the optimized ones from the results we got. We have already finished solving the main problem till this chapter.

5. RESULT UPDATE

5.1 Introduction

We have finished the optimization of the Petri net structure from given observation sequences in the last several chapters. But what need to be noticed is that we only get the approximation of the real Petri net that could generate the given observation sequences because we only use only part of the sequences as input of the algorithm. Though sometimes the result of the state equations could be the same the real Petri net if we are lucky, most of the time it is very hard for us to have exact the same structure. Then with the growing of the observation sequences, the optimized incident matrix B we have could be no longer meet the requirement. This problem will keep existing if we use observation sequences with fixed length. But we can develop another algorithm to help ease this problem.

In this chapter, we will mainly discuss the algorithm that can ease the problem that mentioned above.

5.2 Result incident matrix checking and updating

The method we use to ease the problem is that when the observation sequences growing, checking whether the exist Petri net could satisfy current observation sequences. If it could, then we can still use it. If it could not, we need to use the algorithm in chapter 5 to develop a new one that can fit current observation sequences.

The way that the algorithm works is similar to the one used for Petri net structure optimization that introduced in chapter 5. But here we have already known the incident matrix B. In order to know whether this B matrix can fit current observation

sequences, we use this matrix in state equation systems to calculate the marking difference $M_i - M_j$. Use known M_j, we can know the new M_i. Then we need to calculate the lower and upper bound of Fa' to make sure the range of firing action Fa' for the new generated observation sequences.Let L'_i equals the number of increasing arcs, L'_d equals the number of decreasing arcs in new generated sequences and assume the length of new generated sequences is L', according to the algorithm introduced in chapter 5, the range of number of firing actions should be,

$$Fa'_{lower} \leq Fa' \leq Fa'_{upper}$$

where,

$$Fa'_{lower} = min(min(L'_i, L'_d), L), Fa'_{upper} = max(min(L'_i, L'_d), L)$$

The next step is to check if it exist in new generated observation sequences. This work need to be done for every new generated firing action to make sure we could check every unit length of the observation sequences. If markings M_i exists in for each unit length of new generated observation sequences, that means the Petri net could still meet current requirement. Else we need to develop a new Petri net structure with the algorithm in chapter 5 using current observation sequences as input.

We have to scan every possible marking from the new generated observation sequences in order to find if there exists the same one with calculated M_i. Similar with the algorithm introduced in chapter 5, we still need to figure out the range of number of firing actions and the range of number of possible markings but the difference is we don't need to care about the range of transitions because incident matrix B is known.

5.3 Algorithm

Algorithm 2

Input: Origin observation sequences with length L_o and final marking M_0, new generated observation sequences with length L' and final marking M_k, incident matrix B calculated from origin observation sequences.

Output: Whether the incident matrix could satisfy the new generated observation sequences.

1: Build a set S to store possible marking sequences we found.
2: **for** $Fa' = Fa'_{lower}$ *to* Fa'_{upper} **do**
3: **for** $i = 1$ *to* Fa' **do**
4: **while** 1 **do**
5: Find a possible marking M_i after i times firing.
6: **if** $M_i = M_k$ **then**
7: **if** Marking sequence $M_0M_1...M_i \in S$ **then**
8: Goto line 4
9: **else**
10: Store it in S
11: **end if**
12: **end if**
13: **end while**
14: **end for**
15: **end for**
16: **while** $S \neq \emptyset$ **do**
17: Pick one marking sequence from S and delete it from S
18: **for** $j = 1$ *to* k **do**
19: Solve the state equation $B * V = M_j - M_{j-1}$ and get a V vector
20: **if** The sum of all entries in $V = j$ and all enrties are constant **then**
21: **if** $j \neq k$ **then**

22: Goto line 15
23: **else**
24: **Output** "Incident matrix B satisfies the new generated observation sequences"
25: **end if**
26: **else**
27: Goto line 14
28: **end if**
29: **if** $S = \emptyset$ and $j = k$ **then**
30: **Output** "Incident matrix B does not satisfy the new generated observation sequences"
31: **end if**
32: **end for**
33: **end while**

5.3.1 Complexity Analyze

Better than the optimization algorithm in chapter 5, this algorithm could also work in constant time if we use known incident matrix B as input. First, Marking sequences scanning part works in constant time because same with L, the length of new generated observation sequences L' is also a constant number. Then according to Doctor Li's study[1], it takes $O(m^{L'})$ time to find all possible markings which also means finding all possible marking sequences takes $O(m^{L'})$ time. Because both m and L' are constant, this part could works in constant time $O(1)$. Second, the equation solving part. This time the variable is firing vector V with constant dimension $m \times 1$. As mentioned above, the complexity of solving equation system with n variables is $O(n^3)$. So, the number of variables is constant, the complexity is $O(1)$. Then because the number of markings is constant, the total time complexity of finding all marking sequences is also constant which makes the total complexity constant.

5.3.2 Simple Example

Now we use a simple example to illustrate the algorithm 2. We have got a optimized incident matrix in the example of algorithm 1 which is,

$$B_1 = \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 1 \end{bmatrix}$$

If the observation sequences have grown into from the original ones into current ones shown below.

$S_1 : 2 \rightarrow 1 \rightarrow 0 \rightarrow 1$

$S_2 : 1 \rightarrow 0 \rightarrow 1 \rightarrow 2$

$S_3 : 0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 0$

To check whether matrix B_1 could satisfy this new observation sequences, we only need to use the new generated part as the input of algorithm 2. Original final marking $[0\ 0\ 3]^T$ becomes initial marking here. And the markings is $[1\ 2\ 0]^T$.

$S_1 : 0 \rightarrow 1$

$S_2 : 0 \rightarrow 1 \rightarrow 2$

$S_3 : 3 \rightarrow 2 \rightarrow 1 \rightarrow 0$

From the new generated observation sequences, we could know the range of number of firing actions Fa' where $3 \leq Fa' \leq 3$. So here $Fa' = 3$ which means there is only one loop for Fa'.

Then we do the same permutation operations as in algorithm 1 to find all possible markings from the new generated observation sequences. Permutation matrix $PermM'$ for index of positive part of the first column of T', initial element of $MatrixM'$ and initial element of $MatrixT'$ are shown as following.

$$PermM' = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

$$MatrixM'(1,1) = \begin{bmatrix} 0 & 1 & -1 & -1 \\ 0 & 1 & 2 & -1 \\ 3 & 2 & 1 & 0 \end{bmatrix} MatrixT'(1,1) = \begin{bmatrix} 1 & 0.1 & 0.1 \\ 1 & 1 & 0.1 \\ -1 & -1 & 0.1 \end{bmatrix}$$

First select the first column of $PermM'$, do shifting to the corresponding elements in $MatrixM'(1,1)$ and $MatrixT'(1,1)$. We can find markings from that.

$$Marking_1 = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} Marking_2 = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

Then select the first two columns of $PermM'$, do shifting to the corresponding elements in $MatrixM'(1,1)$ and $MatrixT'(1,1)$. We can find markings from that.

$$Marking_3 = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

Markings $Marking_1, Marking_2$ and $Marking_3$ are all possible markings we could find at the first firing action. Then, we calculate M_1 with state equation system and incident matrix B_1. There are two possible firing vectors $V_1 = [1\ 0]^T$ and $V_2 = [0\ 1]^T$ because B_1 has only two columns which means there are two transitions in this Petri net. We can get M_1 equals,

$$M_{11} = \begin{bmatrix} -1 \\ 0 \\ 4 \end{bmatrix} M_{12} = \begin{bmatrix} 0 \\ -1 \\ 4 \end{bmatrix}$$

We can see that there appears negative numbers in these calculated markings which makes it impossible that they could equal to any of the possible markings in set $\{Marking_1, Marking_2, Marking_3\}$. So we can make conclusion that this optimized incident matrix B_1 could not satisfy the observation sequences after growing which means we need to reconstruct a new Petri net could fit the new sequences with algorithm 1.

5.4 Summary

In this chapter, we talked about the problem that we will have in using the optimization algorithm in chapter 5 and also proposed a new algorithm that can work in constant time to ease this problem by checking if exist Petri net could fit current observation sequences. Then we used an example to illustrate our ideas to make it more clear how to use this algorithm. This algorithm aims at making up the problem that the optimization algorithm have which could not renew the result it generates. Combine these two algorithm together, we could have a complete method which could be used for practical need.

6. PRACTICAL USE

6.1 Introduction

Petri net is a useful tool in many information and technology fields like communication and automatic control system. The application of Petri net has been studied for many years. The ideas and algorithm studied in this thesis could also be useful in practical situations. Because of the unique structure Petri net has, it could represent actual networks and working flow well.

In this chapter, we will introduce some practical ways to use Petri net in IT fields from differnt aspects and give some thoughts about using our ideas and algorithms in these fields.

6.2 Optimization of Network Structure

Sensor network is an important method to transmit data in many fields besides control systems. Petri net could represent the structure of many network well. Wendi B. Heinzelman and the others [8] studied about the sensor networks with middleware. Middleware here has been used to bridging the gap between operation systems and easing the development of distributed applications. According to Wendi B. Heinzelman's idea, the middleware also has the function of managing the connection between client and server.

In this case, we can treat the whole sensor network with middleware as a Petri net. Operation systems and applications in the network will be treated as the places in Petri net, middlewares will be transitions, connections between operation systems and middlewares will be the arcs and the authority for the connections will be dicided by the relationship of arc weights and tokens in places. We can treat the data

transmitted in sensor network as data satisfied certain conditions and the middlewares give authority to corresponding operation systems and applications to make the transmission successful.

Then the problem is how to optimize this kind of networks to reduce the cost of hardware and still can make it functional as predicted. Usually We could not reduce the number of operation systems and applications because of load capacity. So we need to think about reducing the cost on middlewares and connections. We can try to connect more operation systems and applications to the same middlewares to reduce number of middlewares needed. And also, we could optimize the structure of sensor network to reduce the number of connections between Operation systems, applications and middlewares. Those ideas are actually what we talked about in this thesis which is to find the incident matrix B with least transition number m and most zero entries.

6.3 Optimization of Process Flow

Besides the use in optimizing the network structure, Petri net also could represent the process flow of data transmission or industry production. Hassan Jameel and the others [9] worked on a kind of data transmission system between mobile and grid. They developed a work flow of this system with Petri net themselves which has been showns in Figure 5.

In the Petri net represented process flow in Figure 5, place represents each state of the flow, transition represents certain action that can lead the flow go from one state to another and the arc here simply represents the connection between state and action. There will be only one token in this process flow which represents the process of the flow. The token will initially be in the first state marked "*Ready*" in the figure. And as the flow going, it will be transmitted through states and finally be in the state marked "*End session*".

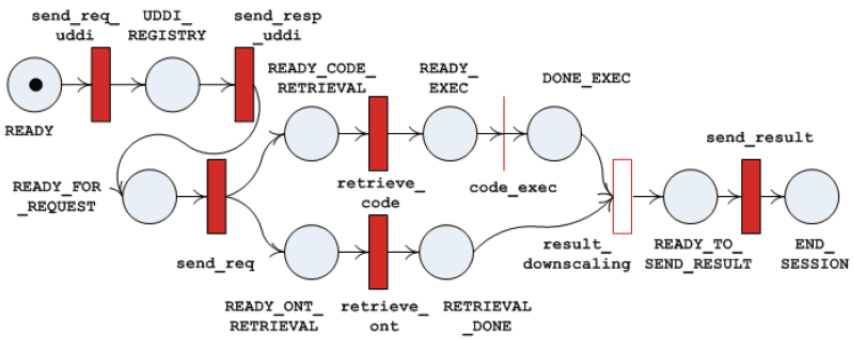

Fig. 6.1. Process Flow introduced in Hassan Jameel's work [9]

This kind of process flow could give us some inspirations of using our algorithm here. Though this Hassan Jameel's work, the process flow has been formed in certain rule and hard to be modified any more, we also could come up with some ideas to optimize some process flow in similar situation. For example, if we want to know how a system or program works but we could only observe its working states, we could use the algorithm discussed in this thesis to find a optimized estimation of its process flow. In that case, we can know approximately what the system or program looks like and help deeply understand it.

6.4 Summary

In this chapter, we introduced some practical use of Petri net and have also talk about how to use our ideas discussed in this thesis in these fields to help optimize their systems. Two main aspects are involved. For network structure, we could help reduce the number of middlewares and the unnecessary connections to reduce the time and money costs. At the same time, simplified systems will also have better efficient performance. For the process flow, we could help deeply understanding unknown systems and programs with only their working states which is also a efficiency-related problem and also could be a way to better assist the systems or programs with other resources.

7. CONCLUSION AND FUTURE WORK

In this thesis, we first proposed an idea that we could treat the length of observation sequences as constant when we use it as inputs to our algorithm. With this assumption, we introduced an algorithm used in Petri net. This algorithm could generate an optimized Petri net from given observation sequences in polynomial time. With some mathematical analysis and software code verification, we could see this algorithm could work. In that case, we also did some assumptions of using this algorithm in real life. Because Petri nets could be used to represent structure of systems and flow of operations, we could use this algorithm to optimize these systems that could help improving their performance.

On the other hand, this algorithm has some disadvantages. One of them is that there would be some estimation errors in this result. This is because we only get the approximation of the real Petri net. For this reason, we developed a new algorithm, which could perform better in time complexity. This algorithm will check whether the resulted incident matrices could work as required if new observations have been generated and decide whether we should run the first algorithm again to get new matrices.

Though this algorithm could work in polynomial time, it uses exhaustive method which could only be used in finite observation sequence length conditions. We can keep the finite sequence length condition, but use another method to find markings and V vectors with higher efficiency.

What's more, there should exist a method that could work in variable sequence length conditions. If we want to realize this, more assumptions need to be added to this problem to make it possible to select a constant number of markings and V vectors in every loop. If this modified algorithm could be proposed, it could be used in much wider applications.

Besides those, the method of updating the result of the algorithm is also a direction could be improved. In this thesis, we use another algorithm with enumeration method to check the B matrix in the result. We could make the algorithm update the result itself as the observation sequences grow to meet the requirement. In this way, it could get rid of the complicated enumeration and avoid reading the whole observation sequences again.

REFERENCES

REFERENCES

[1] A. M. M. M. Dotoli, M. P. Fanti and W. Ukovich, *On-Line Identification of Petri Nets with Unobservable Transitions.* Gteborg, Sweden: Proceedings of the 9th International Workshop on Discrete Event Systems, May 28-30, 2008.

[2] Y. C. Y. Qu, L. Li and Y. Dai, *Fault Detection in Acyclic Petri Net Models.* 978-1-4244-5182-1/10/ IEEE, 2010.

[3] L. Li and C. N. Hadjicostis, *Reconstruction of Transition Firing Sequences Based on Asynchronous Observations of Place Token Changes.* New Orleans, LA, USA: 46th IEEE Conference on Decision and Control, Dec.12-14,2007.

[4] J. Yan, *Reconstruction of Event Sequences Based on Asynchronous Observations in Sensor Networks.* Kota Kinabalu, Malaysia: ICUIMC(IMCOM)'13, January 17-19, 2013.

[5] A. Giua and C. Seatzu, *Observability of Place/Transition Nets.* IEEE TRANSACTIONS ON AUTOMATIC CONTROL, VOL. 47, NO. 9, SEPTEMBER 2002.

[6] L. Li and C. N. Hadjicostis, *Least-Cost Transition Firing Sequence Estimation in Labeled Petri Nets With Unobservable Transitions.* IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, VOL. 8, NO. 2, APRIL 2011.

[7] D. Lefebvre and A. E. Moudni, *Firing and Enabling Sequences Estimation for Timed Petri Nets.* IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICSPART A: SYSTEMS AND HUMANS, VOL.31, NO.3, MAY 2001.

[8] H. S. C. W B. Heinzelman, A L. Murphy and M. A. Perillo, *Middleware to Support Sensor Network Applications.* IEEE Network, January/February 2004.

[9] A. S. S. L. H. Jameel, U. Kalim and T. Jeon, *Mobile-to-Grid Middleware: Bridging the gap between mobile and Grid environments.* South Korea: Department of Computer Engineering, Kyung Hee University.

[10] B. C. F. Arichi, H. Kebabti and M. Djemai, *Failure Components Detection in Discrete Event Systems Modeled by Petri Net.* Algiers, Algeria: Proceedings of the 3rd International Conference on Systems and Control, October 29-31, 2013.

[11] Y. Ru and C. N. Hadjicostis, *Bounds on the Number of Markings Consistent With Label Observations in Petri Nets.* IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, VOL. 6, NO. 2, APRIL 2009.

[12] Y. Ru and C. Hadjicostis, *Reachability Analysis for a Class of Petri Nets.* Shanghai, P.R. China: Joint 48th IEEE Conference on Decision and Control and 28th Chinese Control Conference, December 16-18, 2009.