Graduate School Form 30
Updated 1/15/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Aja Anjilivelil

Entitled
MODELING, SIMULATION, AND OPTIMIZATION OF TRAFFIC INTERSECTIONS USING PETRI NETS

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

Is approved by the final examining committee:

Lingxi Li Chair	
Brian S. King	
Maher E. Rizkalla	

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 of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Lingxi Li

Approved by: Brian S. King 6/8/2015
Head of the Departmental Graduate Program Date

MODELING, SIMULATION, AND OPTIMIZATION OF TRAFFIC INTERSECTIONS USING PETRI NETS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Aja Anjilivelil

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

August 2015

Purdue University

Indianapolis, Indiana

This thesis is dedicated to all those who believed in me....

For their endless love, support and encouragement.

ACKNOWLEDGMENTS

I would like to sincerely thank Dr. Lingxi Li for accepting me into his team and for giving me assistance through out my thesis. His advice and help was crucial for me to finish my work. Also, I would like to acknowledge my thesis committee members Dr. Brian King, Dr. Maher Rizkalla for their time and effort. Their guidance and help were invaluable right from the beginning of my study at IUPUI. Lastly, I am grateful for the assistance of Sherrie Tucker and all other faculties of the ECE department.

My friends have been a constant source of encouragement for me. And I was grateful to have them around during my study. I especially want to thank Omar Nezamuddin. He was always there whenever I needed any help.

Finally, I would like to thank my family for their endless love and support. Their patience and encouragement kept be motivated.

Thanks you all

Aja.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Anjilivelil, Aja M.S.E.C.E, Purdue University, August 2015. Modeling, Simulation, and Optimization of Traffic Intersections using Petri nets. Major Professor: Lingxi Li.

With the increasing number of vehicles on the road and the amount of time people spend driving their vehicles, traffic control and management has become an important part within logistics. Effective traffic control would involve traffic signal control and control over vehicle movement. Since Petri nets are versatile enough to represent traffic signals and traffic flow, it has become an important tool in urban traffic control. Many traffic systems are modeled using hybrid Petri nets. Chapter 1 briefly talks about traffic management systems and previous related work in the area of traffic control. Chapter 2 is a basic background on various Petri nets used in the study. The section also uses examples to demonstrate the working of Petri nets. Chapter 3 introduces the need for optimization in various industry. And then, it discusses different steps involved in optimizing a process. Chapter 4 discusses the existing model of two one-way intersection. In an effort to understand the model better, simulations are also carried out. Then, drawbacks of the existing model are discussed. This paves way for a new, improved, and realistic version of two one-way intersection. Various optimization steps discussed in Chapter 3 is used to optimize traffic light of the improved model. And then, a comparison between existing model and improved model is carried out. Chapter 5 expands the study of traffic models by connecting two different one-way intersection through a road (thus making it a network). Optimization and simulation of the connected-intersection model is also carried out. Chapter 6 is the summary which will provide a brief overview about each chapter.

1. INTRODUCTION

The number of vehicles on the road has steadily increased ever since automotive industry began to prosper. During early ages, traffic control was established mainly through traffic signals and signs. But with the increasing number of people/vehicles in cities/towns, traffic control and management required the use of much more sophisticated tools. Hence a lot of research have been carried out by independent organizations and/or government agencies in the field of traffic control ever since.

1.1 Traffic management

Earliest form of traffic management consisted of traffic signals and signs. But later on, as number of vehicles on road increased tremendously, various sophisticated algorithms and tools had to be used to accomplish the task. And much time have been spent on the research about optimizing these tools and algorithms. In an attempt to understand them better, a brief discuss about various traffic management strategies have to be carried out. There are couple of popular traffic management strategies in use. In its simplest form, traffic management in recent time consist of an optimization algorithm aiming to improve one or two key factors affecting traffic (namely duration of green signal and length of queue). A brief discussion about them is carried out later in this chapter.

According to the techniques used to manage the traffic, traffic management can be classified into two. (i) fixed time strategy (ii) traffic-response strategy. Fixed time strategy requires knowledge about existing layout of the intersections, roads connecting them, stop signs within an area, etc. to carry out its task. It also requires an optimization algorithm. The technique studies traffic pattern of the area (during peak time and otherwise). And this data is used to solve the optimization algorithm.

Based on the data, algorithm will come up with (i) optimum duration of green light (ii) optimum phase plan: total number of phases in a traffic cycle, vehicle movement from which direction should be allowed during a phase, which phase should follow the current phase etc. (iii) offset: time difference between various traffic signals in the area. However, the main draw back of such a system is that, it is not adaptive to current traffic situations. For instance, if there is an accident in the area, then the system will behave the exact same way it behaved before (duration of traffic light and offset won't change because of the accident). Hence, such type of traffic management is not advised for high traffic areas. Because any deviation from normal scenario would create inefficiency in the system. However, this method is perfect for low traffic roads because it does not require the use of any additional tools. The method will work well if you have an algorithm and the knowledge about traffic pattern. TRANSYT (Traffic Network Study Tool) [2] is a system which employs fixed timed strategy. Its main purpose is to determine signal timing and optimization. It features a generic algorithm which optimizes signal duration/phase plan/offsets. CRONOS [11] is another traffic management tool which uses fixed time strategy.

Traffic response strategy, on the other hand is an interactive system. This offers up-to-date traffic information and thus ensures predictability and reliability under all circumstances. This method also requires the knowledge of existing system layout (roads, intersections etc.). In addition to that, it employs sensors/detectors at every traffic signal/link/intersection. The data collected by the sensors are used to determine performance parameters (optimum duration of traffic lights). Since data collected by the sensors are live, the performance parameters calculated will be based on the current situation. And hence, the system is effective under all circumstances. SCOOT (Split Cycle Offset Optimization Technique) [1] is a leading tool which manages traffic using current data. The data collected by sensors/detectors within SCOOT controlled area is analyzed and based on that, necessary changes are made to traffic signal duration. And thus vehicles don't have to wait in queue for long. Since it responds to traffic fluctuations automatically, the technique is proven to reduce traffic delays considerably. Another example of an adaptive traffic control system is SCATS (Sydney Coordinated Adaptive Traffic System). This system is similar to SCOOT in the fact that it requires the knowledge of existing layout and it employs sensors to detect current traffic situation at a traffic signal. In addition, SCATS also maintains a library of solutions. So when SCATS encounters a scenario, it automatically selects a plan/solution from a library (of solutions) in response to the current traffic problem. Even though traffic-response strategy is highly effective, because of the added cost (of sensors/detectors), this technology is used only in high traffic areas.

In recent years, traffic management and control have expanded to addresses environmental issues, allocating the use of infrastructure to cars, bikes and, foot traffic and to various other areas. Many intelligent transportation systems (ITS) are capable of performing some of these tasks already.

1.2 Thesis contributions

Petri net is a modeling language which could be used to represent time driven or event driven systems. The simplicity with which a system could be represented using Petri net have made them a popular modeling tool in the field of ITS (intelligent transportation systems) , manufacturing etc. Many traffic systems have been modeled using Petri nets. The fact that Petri nets can represent both traffic signals and vehicle flow effectively, made the modeling of the traffic intersection easy and simple.

Optimization is a mathematical concept which involves selecting the best scenario out of a few choices. Engineering industry have been using optimization for a long time. In Engineering, optimization process is mainly used to maximize the output of a process while minimizing the effort. Optimization of traffic signals have been researched quiet a lot. And optimization of a traffic signal which was modeled using Petri net was the inspiration for this thesis.

This thesis mainly discusses traffic intersections which are modeled using Petri nets. Two one-way intersection have been chosen for simplicity. Work in this thesis mainly concentrates on three specific models. First model is the existing system (system which have fixed cycle time i.e, duration of red/green/yellow periods are constant). This is the base model. In this, vehicles arrive and depart the intersection at a constant rate. Working of the model have been discussed and then simulations have been carried out for better understanding of the model. Mathematical studies and objective function describing such a model have also been discussed. An algorithm to calculate the objective function value have been developed as a part of this thesis. Finally, in order to overcome the flaws of the existing model, we move on to the model proposed by Vandzquez, Sutarto, Boel, and Silva [8].

In reality, since traffic does not arrive or depart at an intersection at constant rate, the existing model is not much accurate. Vandzquez, Sutarto, Boel, and Silva [8] proposes a model much more realistic. In the new model, traffic arrival and departure at the intersection occurs in bursts. The burst happens when traffic light at the upstream intersection turns from red to green. At this time, all vehicles will leave the upstream intersection at once. Hence it arrives the downstream intersection as a burst. After a while, when majority of vehicles from upstream intersection have reached the downstream intersection, the burst will start to decay. And finally when all the vehicles have reached downstream intersection, the burst will die. This thesis explains the working of such a model. Mathematical formulas have been deduced to calculate the objective function of the model. In an effort to optimize such a traffic model (duration of green signal), an algorithm have been developed. Results of both the algorithms have been compared to analyze their performance quantitatively.

Third and final model is the model of a traffic system with two intersections connected through a road/link. Each one of the intersection have a traffic signal to control it. Like previous models, this model was also developed using Petri net. This model is a combination of the two previously discussed models. And hence, algorithms used to optimize previous models have been modified to optimization the new traffic model. To understand the working of the new model, simulations have been carried out as well. And finally, all models are compared to analyze their performance.

1.3 Thesis organization

This thesis is divided into six chapters. First chapter is a small introduction. It explains inspiration for the work. And it also gives an overview of forthcoming chapters. Chapter 2 is an introduction to the technology which is used in this study: Petri nets. It starts by explaining basic definitions and notation of Petri nets. Later, in the chapter, simple examples are used to explain their properties and dynamics. Different types of Petri nets (hybrid Petri net, timed hybrid Petri net) which are used in the thesis is also explained in this chapter. Their additional features and working are illustrated as well. Since much of the work in this thesis deals with optimization of traffic signals, Chapter 3 is an introduction to optimization process. Chapter explains various steps involved in optimizing a process. Thus Chapter 3 is a overview of process which is used to optimize our problem. Chapter 4 starts by explaining the model of a basic one-way traffic intersection (the intersection is controlled only by traffic light). Dynamics of the model is explained, followed by the simulations of the model. The model is studied mathematically, so that major parameters affecting its performance could be identified. These parameters are used to formulate the objective function. And then, an algorithm to evaluate its objective function is developed. Discussion then moves on to traffic intersection model proposed by Vandzquez, Sutarto, Boel, and Silva [8]. Working of the model is explained in order to understand the model better. Parameters affecting its performance are also identified. In an effort to optimize the traffic signal, mathematical deductions are carried out to obtain an objective function. Based on the working of the model, a program is developed to obtain an optimum value for the model. And finally, simulations of the model is also carried out. Chapter 5 expands the single intersection model by connecting two different single intersections. The Petri net model of connected-intersection is explained. Simulation of the model is also done. Then, traffic signal of the connected-intersection model is optimized by using the algorithms developed for the previous models. And finally, Chapter 6 is the conclusion. It summarizes all the work. And then presents with suggestions for future work.

2. INTRODUCTION TO PETRI NETS

2.1 Definition

Petri net was discovered by Carl Adam Petri for describing chemical processes. A Petri net (place/transition net) is a mathematical modeling language which could be used for describing/representing a system. It is a weighted directed bipartite graph consisting of two nodes namely (i) places (ii) transitions. These nodes are connected to each other by weighted directed arcs. So given a set N, a Petri net could be represented by,

$$N = (P, T, A, W)$$

where,

- P is finite set of places. Places are typically represented by circles.
  $P = \{p_1, p_2, p_3.....\}$

- T is finite set of transitions. Transitions are represented as bars.
  $T = \{t_1, t_2, t_3....\}$

- A is set of arcs which connects various nodes, i.e an arc connects a place to a transition or a transition to a place.

  $$A \subseteq (P \times T)\, U\, (T \times P)$$

  It is important to note that an arc never runs between two similar nodes, i.e there will not be a connection between two places or two transitions through a single arc.

- W is set of positive integers which represents the weight of the arc connecting various nodes. Every arc has a weight attached to it. If no weight is mentioned

for an arc, then weight is assumed to be one. Thus no arc has zero weight. And if weight of an arc between two nodes is more than one, then it indicates that there are multiple arcs between those nodes. For example, if an arc from p_1 to t_1 has a weight two, then it means that there are two arcs (each of weight one) between p_1 and t_1. The two arcs (each of weight one) could be replaced with a single arc of weight two.
$W = \{1,\ 2,\ 3.....\}$

The place from which an arc runs to a transition is called an input place of the transition. $I(t_j)$ is used to represent the set of input places to transition t_j. So,

$$I(t_j) = \{p_i \epsilon P \mid (p_i, t_j) \epsilon A\}$$

The places to which an arc runs from a transition is called output place of the transition. $O(t_j)$ is used to represent output places of the transition t_j. So,

$$O(t_j) = \{p_i \epsilon P \mid (t_j, p_i) \epsilon A\}$$

similarly,

$$I(p_i) = \{t_j \epsilon T \mid (t_j, p_i) \epsilon A\}$$

$$O(p_i) = \{t_j \epsilon T \mid (p_i, t_i) \epsilon A\}$$

represents input and output transitions to and from places. Fig. 2.1 shows a simple Petri net. Various nodes and components of the Petri net are as follows:

$$P = \{p_1, p_2, p_3\}$$

$$T = \{t_1, t_2, t_3\}$$

$$A = \{(p_1, t_1), (p_1, t_2), (t_1, p_2), (p_2, t_3), (t_3, p_3), (t_2, p_3)\}$$

$$W = \{1, 1, 2, 1, 1, 1\}$$

$$I(t_1) = \{p_1\}, \quad I(t_2) = \{p_1\}, \quad I(t_3) = \{p_2\}$$

$$I(p_1) = \phi, \quad I(p_2) = \{t_1\}, \quad I(p_3) = \{t_2, t_3\}$$

$$O(t_1) = \{p_2\}, \quad O(t_2) = \{p_3\}, \quad O(t_3) = \{p_3\}$$

$$O(p_1) = \{t_1, t_2\}, \quad O(p_2) = \{t_3\}, \quad O(p_3) = \phi$$

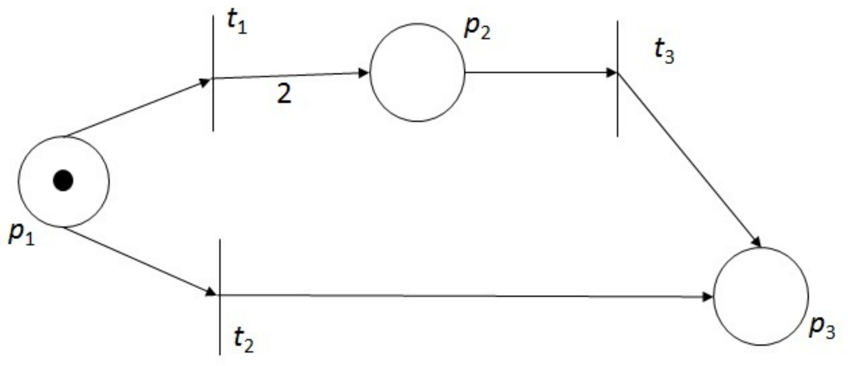


Fig. 2.1. A simple Petri net.

Since Petri net is a pictorial representation, it is quiet easy to deduce all the information and thus to understand its working. It is this simplicity which makes it a popular modeling tool. Following sections will discuss about Petri nets in detail.

2.2 Petri net marking

Petri net offers a graphical representation for step wise evolution of the system. In order to capture the evolution/change in the state of the system, Petri net uses a mechanism called tokens. Graphically, tokens are the finite number of black dots inside a place in the Petri net. Any distribution of tokens over the place will represent marking of the Petri net. Typically, marking of a Petri net is a column vector (known as marking vector) with each element of the vector representing tokens in the corresponding place. So $m(p_i)$ represents the number of tokens in place p_i. If a place does not have any tokens, then a zero will be entered in its place in the marking vector. And hence, number of elements in the marking vector will be equal to number of places in the Petri net. In Fig. 2.1 the initial marking of the system m_0 is $[1 \quad 0 \quad 0]^T$.

2.3 Petri net dynamics

A transition $t_j \epsilon T$ is said to be enabled if the number of tokens in each input place (p_i) of transition t_j is greater than or equal to the weight of the arc which connects place p_i to transition t_j, i.e transition t_j is enabled if and only if:

$$m(p_i) \geq W(p_i, t_j) \ \ \forall \ \ p_i \epsilon I(t_j) \tag{2.1}$$

Below example demonstrates Petri net dynamics. In Fig. 2.2, transition t_1 has three

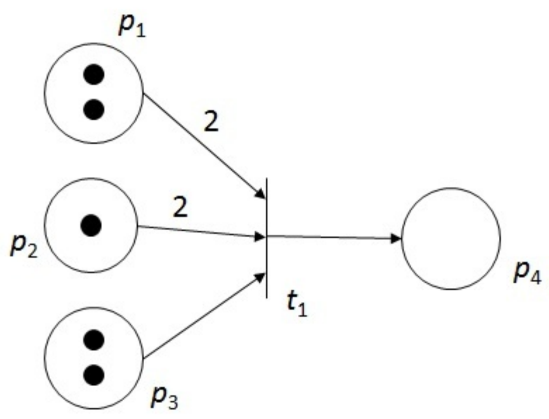


Fig. 2.2. Petri net dynamics (a).

input places. The number of tokens in p_2 is less than the weight of arc connecting p_2 and t_1. So, even though p_1 and p_3 satisfy the equation 2.1, t_1 is not enabled because p_2 does not satisfy equation 2.1.

Fig. 2.2 could be modified to increase the number of tokens in p_2. In Fig. 2.3, number of tokens in all the input places are greater than or equal to weight of the arc connecting the places to transitions. Hence they satisfy the equation 2.1. And so, transition t_1 is enabled.

Dynamics of a Petri net changes when an enabled transition fires. A transition can fire anytime after it becomes enabled. And when a transition t_j fires, following changes take place:

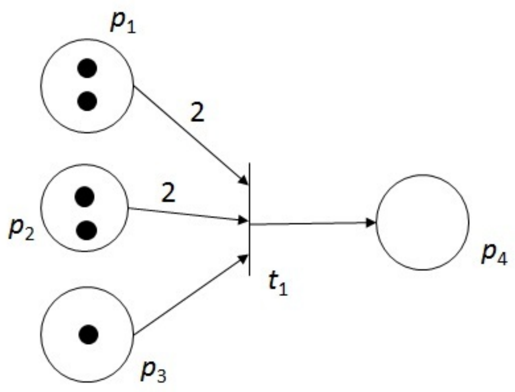


Fig. 2.3. Petri net dynamics (b).

- From each input place p_i, it removes as many tokens as the weight of the arc connecting the input place p_i to transition t_j.
- To each output place p_0, it deposits as many tokens as the weight of the arc connecting the transition t_j to place p_0.

It is worth to note that, while firing a transition, the number of tokens removed from an input place need not be same as the number of tokens deposited to output place. Tokens removed and deposited depends on the weight of the arc connecting places and transitions. So the total number of tokens may not be conserved during this process. In Fig. 2.3, after transition t_1 fires, the markings of the places will be $m(p_1) = 0$, $m(p_2) = 0$, $m(p_3) = 0$, $m(p_4) = 1$. Now, the input places don't have any tokens left. So transition t_1 is not enabled anymore (by default, weight of the arc from place to transition is one. Since places don't have any more tokens, equation 2.1 is not satisfied and hence transition t_1 is not enabled). In short, for a Petri net to fire once, at least one of the input place should have at least one token. Also, it is important to note that if a place serves as an input to more than one transition, then there is a good chance that more than one transition might be enabled at a particular time. However, only one among them will fire at a time. This might lead to some conflict among transitions. More details about such conflict is discussed in the forthcoming section. For now, let's consider an example to demonstrate Petri net evolution.

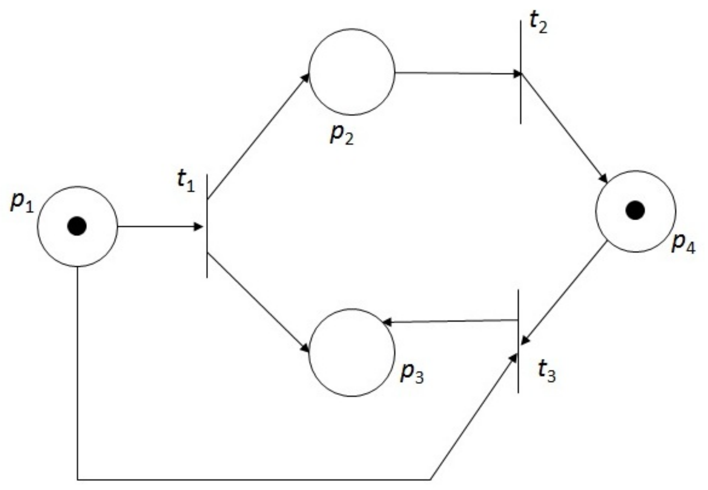


Fig. 2.4. Petri net firing.

1. Initial marking $m_0 = [1 \quad 0 \quad 0 \quad 1]^T$.
2. Use equation 2.1 to find out enabled transitions. Here transitions t_1 and t_3 are enabled. But only one transition will fire at a time.
3. Let transition t_1 fire.
   (a) When transition t_1 fires, it will
      i. Remove one token from place p_1 (input place).
      ii. Deposit one token each at p_2 and p_3 (output places).
   (b) The new marking will be $[0 \quad 1 \quad 1 \quad 1]^T$.
   (c) Use the marking from previous step and equation 2.1 to determine all the enabled transitions. Transition t_2 is the only transition which satisfies the equation 2.1. Hence t_2 is the only enabled transition.

   (d) Transition t_2 will fire. During the process, it will

      i. Remove one token from place p_2 (input place).

      ii. Deposit one token at place p_4 (output place).

   (e) The new marking will be $[0 \quad 0 \quad 1 \quad 2]^T$.

   (f) Use the marking from previous step and equation 2.1 to determine all the enabled transition. None of the transitions are enabled.

4. Let transition t_3 fire.

   (a) When transition t_3 fires, it will

      i. Remove one token each from p_1 and p_4 (input places).

      ii. Deposit one token to place p_3 (output place).

   (b) The new marking will be $[0 \quad 0 \quad 1 \quad 0]^T$.

   (c) Use the marking from the previous step and equation 2.1 to determine all the enabled transitions. None of the transitions are enabled.

5. End.

Since none of the transitions are enabled any more, no transitions will fire. And so, that will conclude the dynamics of Petri net in Fig. 2.4. The above process can be pictorially represented as shown in Fig. 2.5. It is called a reachability tree. It captures all the possible states which could be reached from the initial state. Reachability tree in Fig. 2.5 is very specific to the particular initial marking of $[1 \quad 0 \quad 0 \quad 1]^T$. The Petri net in Fig. 2.4 will have different reachability tree for another initial condition. For example, if $m(p_2) = m(p_3) = m(p_4) = 0$ and $m(p_1) = 1$ then reachability tree would be as shown in Fig. 2.6.

$$\begin{tikzcd}
 & {[1\ 0\ 0\ 1]^T} \arrow[dl, "t_1"'] \arrow[dr, "t_3"] & \\
{[0\ 1\ 1\ 1]^T} \arrow[d, "t_2"] & & {[0\ 0\ 1\ 0]^T} \\
{[0\ 0\ 1\ 2]^T} & &
\end{tikzcd}$$

Fig. 2.5. Reachability tree.

$$\begin{tikzcd}
{[1\ 0\ 0\ 0]^T} \arrow[d, "t_1"] \\
{[0\ 1\ 1\ 0]^T} \arrow[d, "t_2"] \\
{[0\ 0\ 1\ 1]^T}
\end{tikzcd}$$

Fig. 2.6. Reachability tree with different initial marking.

2.4 Incidence matrices

Given a Petri net with n places and m transitions, we can uniquely define three $n \times m$ matrices namely (i) output incidence matrix (ii) input incidence matrix (iii) incidence matrix.

Output incidence matrix, B^+, is an $n \times m$ matrix which captures the weight of the arc from a transition t_j to place p_i. If there is no arc from transition t_j to place p_i then b_{ij}^+ will be zero. So for Fig. 2.4,

$$B^+ = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Input incidence matrix, B^-, is an $n \times m$ matrix which captures the weight of the arc from place p_i to transition t_j. If there is no arc from place p_i to transition t_j then b_{ij}^- will be zero. So for Fig. 2.4,

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

And finally incidence matrix $B = B^+ - B^-$. So for Fig. 2.4,

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

Note that incidence matrices do not talk about tokens at all. It is a structural property. And it is independent of state of the system. These three matrices are very important for a Petri net because, state of a Petri net can be uniquely determined from these matrices.

2.5 State equation of a Petri net

As discussed previously, one way to analyze a Petri net is by using reachability tree. It will give all possible states which could be reached from the initial state. But analyzing a system using reachability tree will be a much more difficult task when the Petri net under consideration is large and complicated. Under such circumstances it is better to use state equation to find the marking of a system. This method finds out all the markings using a mathematical equation. Since markings are found out algebraically, the process is much easier and faster. The relationship between two concurrent/consecutive marking in a Petri net is established through state equation. If a Petri net has n places and m transitions then the state equation can be defined as:

$$M_{k+1} = M_k + B.X_k \tag{2.2}$$

where,

- M_{k+1} is an $n \times 1$ vector. It captures marking of the Petri net at time step $k+1$.
- M_k is an $n \times 1$ vector. It captures the marking of the Petri net at time step k.
- B is incidence matrix.
- X_k is an $m \times 1$ vector called firing vector. This vector has only one non-zero entry per step. And the non-zero entry will be 1. The non-zero entry will indicate the transition which will fire next.

So for the Fig. 2.4, marking at time step one when t_1 fires would be,

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

And after that, since t_2 is enabled, t_2 will fire. Marking after t_2 fires will be,

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} -1 & 0 & -1 \\ 1 & -1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & -1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 2 \end{bmatrix}$$

Since this is an algebraic method, it is much easier to find out all the markings using basic mathematics softwares. And hence this is a much more popular choice to find out all the markings of a Petri net.

2.6 Continuous Petri net

So far we discussed Petri nets which are event-driven. Such Petri nets change their state upon the occurrence of a discrete event. And hence they are called discrete Petri nets. But most physical systems changes their state with respect to time. For example vehicle movement, water flow, etc. Such systems cannot be represented using a discrete Petri net. We use continuous Petri net to represent such systems. Continuous Petri nets are time-driven. And unlike discrete Petri nets, their dynamics change with time. Major differences between a discrete Petri net and continuous Petri net are:

1. Places in a continuous Petri net are represented by double circle. And transitions are represented by rectangular box.

2. The number of tokens in a place need not be an integer. It could be any positive real number.

3. Weights of the arc need not be integers. Any positive real number could be the weight of an arc connecting continuous nodes.

4. A continuous transition has enabling degree (q). Enabling degree (q) has to be greater than zero. And a continuous transition t_j is q-enabled in a marking m_k if, for each input i to transition t_j,

$$q = min\left(\frac{m_k(p_i)}{B^{-}(p_i,t_j)}\right)$$

5. Continuous transitions have firing rates associated with them. If the firing rate associated with a transition t_j is α, then the transition t_j will fire α tokens/unit time. Consider the Fig. 2.7 below:

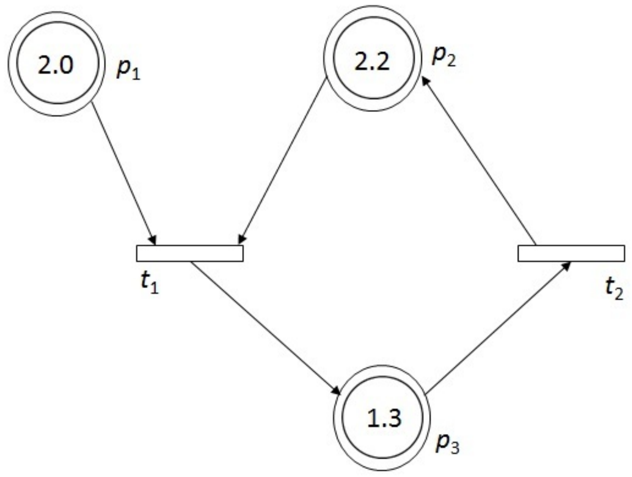


Fig. 2.7. Continuous Petri net.

- Here transition t_1 is 2-enabled and transition t_2 is 1.3-enabled.
- Let firing rate of t_1 be 0.2. Then, when t_1 fires, 0.2 tokens will be removed from p_1 and p_2.
- 0.2 tokens will be deposited in p_3. Therefore marking after t_1 fires will be $[1.8 \quad 2 \quad 1.5]^T$.

6. Since markings in a continuous Petri net changes constantly, a firing sequence would indicate a series of successive markings. And so there will be infinite number of reachable marking for a continuous Petri net. And it is not possible to capture all reachable state in a reachability tree. Instead macro marking is used to capture general marking of a continuous Petri net. If a Petri net has n places, then it will have a maximum of 2^n macro markings. A macro marking of a continuous Petri net with n places would be $[0 \quad 0 \quad 0]^T, [m_1 \quad 0 \quad 0]^T, [0 \quad m_2 \quad 0]^T, [0 \quad 0 \quad m_3]^T, [m_1 \quad m_2 \quad 0]^T, [m_1 \quad 0 \quad m_3]^T,$ $[0 \quad m_2 \quad m_3]^T, [m_1 \quad m_2 \quad m_3]^T$. Macro markings of continuous Petri net in Fig. 2.7 is shown in Fig. 2.8. Since it is not possible for m_2 to reach zero, the macro marking which has $m_2 = 0$ won't be considered.

$$\begin{tikzcd}
(m_1\ m_2\ m_3) \arrow[loop above, "t_1\text{ , }t_2"] \arrow[r, "(t_1)^{m1}"'] \arrow[d, "(t_2)^{m3}"'] & (0\ m_2\ m_3) \arrow[loop above, "t_2"] \arrow[d, "(t_2)^{m3}"] \\
(m_1\ m_2\ 0) \arrow[loop below, "t_1"] \arrow[r, "(t_1)^{m1}"'] & (0\ m_2\ 0)
\end{tikzcd}$$

Fig. 2.8. Macro marking of continuous Petri net.

7. State equation of a continuous Petri net is,

$$M = M_0 + B.X \tag{2.3}$$

where,

- M_0 is the initial marking.
- B is the incidence matrix.
- M is the marking which could be reached from M_0.
- X is the firing vector. Non-zero element (representing firing transition) in the firing vector X, will be the firing rate of the corresponding transition. So unlike discrete transition, non-zero element in X can be any positive real number.

2.7 Hybrid Petri net

So far we have discussed discrete Petri nets and continuous Petri nets. Discrete Petri net is used to represent discrete event. While continuous Petri net is used to represent a continuous process. But most process in the real world is a combination of discrete and continuous events. For example, a process trying to represent vehicles moving in and out of a 4-way intersection require both discrete and continuous nodes (discrete node to capture traffic lights and continuous nodes to capture vehicle movement). Such Petri net which uses discrete nodes and continuous nodes are called hybrid Petri nets. A hybrid Petri net is a sextuple. And it could be defined as:

$$Q = (P, T, Pre, Post, m_0, h)$$

where,

- $P = \{p^d, p^c\}$. Set of all discrete and continuous places.
- $T = \{t^d, t^c\}$. Set of all discrete and continuous transitions.
- $Pre =$ input incidence matrix. B^-.
- $Post =$ output incidence matrix. B^+.
- $m_0 =$ initial marking.
- $h =$ hybrid function which indicates whether the node is a discrete node or continuous node. p^d represent discrete places while p^c represent continuous place. Likewise, t^c, t^d represents continuous and discrete transitions respectively.

It is important to keep in mind that, in a hybrid Petri net the condition

$$B^+(p^d, t^c) = B^-(p^d, t^c) \tag{2.4}$$

has to be satisfied at all the time. i.e an arc connecting a continuous transition to a discrete place must have the same weight as the arc which connects the discrete place to a continuous transition.

Since hybrid Petri net is a combination of continuous and discrete nodes, the enabling condition of a transition (whether continuous or discrete) vary depending on the type of transition. For example, if the transition is discrete, then the enabling condition is same as the one described in equation 2.1, which is,

$$m(p_i) \geq W(p_i, t_j) \ \forall \ p_i \epsilon I(t_j)$$

But if the transition is continuous, the enabling condition depends on the input place. If the input place is discrete, then the enabling condition is,

$$m(p_i^d) \geq B^-(p_i^d, t_j^c) \ \forall \text{ discrete place } p^d \text{ to continuous transition } t^c.$$

And if the input places is continuous, then enabling condition is,

$$m(p_i^c) > 0 \ \forall \text{ continuous input place } p^c \text{ to continuous transition } t^c.$$

And also, an enabled continuous transition is said to be strongly enabled if marking of every continuous input places is greater than zero. Else it is called weakly enabled. The evolution/state equation of a hybrid Petri net is described by the equation:

$$M_{k+1} = M_k + B.s \tag{2.5}$$

All the terms of the equation 2.2 and equation 2.5 are the same, except that the components of firing vector in equation 2.2 are integers while those in equation 2.5 could be any positive real number. So for a hybrid Petri net, the non-zero element in the firing vector will indicate the transition which will fire. And if the transition is a continuous transition then, non-zero number will be the firing rate of the transition.

While modeling a system using hybrid Petri net, it is often advised to number all the discrete nodes before continuous nodes. That way, while creating the incidence matrix, all discrete nodes will appear before continuous nodes.

2.8 Timed Petri net

While using a Petri net to represent a system, the state of the system will be usually associated with the place of the Petri net and changes to the state (or an event) will be associated with the transition. If the system remains in a particular state for sometime, it is important to associate the duration of the state with the place. Similarly, if there is a delay associated with an event, then it has to be taken care by the transition. Such Petri nets with time/delay associated with place and/or transition is called is timed Petri net. The time/delay could be associated with discrete nodes or continuous nodes.

If the delay associated with the place is d_A then the token in the place must remain in the place for a duration of d_A time units before allowing firing of the transition associated with the place. If d_A is the time associated with a transition, then the transition will fire d_A time units after it has become enabled. If no delay is associated with a transition, then it will fire as soon as it becomes enabled. And if the delay associated with a place/transition is zero, then it is usually not specified.

Petri nets with deterministic delays associated with them are called deterministic-timed Petri net. If the delay follows a probability distribution, then the Petri net is called stochastic-timed Petri net. For such Petri nets, the time associated with transition t_j could be a random variable X with exponential distribution function.

2.9 Timed hybrid Petri net

As discussed in previous section, if there is time associated with the nodes of a Petri net, then the Petri net is called timed Petri net. Similarly, if there is time associated with nodes of a hybrid Petri net, then the hybrid Petri net is called timed hybrid Petri net. Such Petri nets have timed delays associated with place and/or transitions. Structure of timed hybrid Petri net is defined as,

$$Q = (P, T, Pre, Post, m_0, h, \lambda)$$

The only new component in the definition of a timed hybrid Petri net (compared to hybrid Petri net) is λ. Rest of the parameters have the same definition as before. In a timed hybrid Petri net, if the transition is discrete then λ is a positive integer represented by d_i. And d_i is the time associated with the discrete transition. The discrete transition will wait for d_i time units before firing, once it has become enabled.

If $d_i = 0$, then the discrete transition will fire as soon as it becomes enabled. If the transition is continuous, then λ is positive real number represented by U_i. And U_i is the maximum flow rate of the continuous transition. U_i is an important factor in determining the maximum firing speed of the continuous transition. Because, for a continuous transition, maximum firing speed is calculated from maximum flow rate by the equation shown below:

$$V_i = Ui \times D(t_i, m_k^d) \tag{2.6}$$

where,

- V_i = maximum firing speed.
- U_i = maximum flow rate of the continuous transition.
- $D(t_i, m_k^d)$ = enabling degree of the continuous transition t_i. If the continuous transition t_i has n discrete input places, then the enabling degree of transition t_i is the least number of tokens which will be transfered in one transaction. If the continuous transition does not have any discrete place as input , then the transition will fire at infinite speed. In order to avoid that, continuous transitions of a timed hybrid Petri net will always have at least one discrete place as its input.

The state equation of a timed hybrid Petri net is slightly different from that of a hybrid Petri net. Starting with the characteristic equation, for a timed hybrid Petri net,

$$s(t) = n(t) + \int_0^t v(u)du \tag{2.7}$$

where,

- $n(t)$ is the firing vector. This vector will have zero as its entry for every continuous transition. Only one element in the vector will have a non-zero entry. That entry will indicate the discrete transition which will fire.

- $v(u)$ is the firing speed of the continuous transitions. This vector will have zero as its entry for every discrete transition. Non-zero entry could be any positive real number and it indicates the firing speed of continuous transition which will fire.

And hence the state equation at time t for a timed hybrid Petri net is,

$$m(t) = m(0) + B\left(n(t) + \int_0^t v(u)du\right) \tag{2.8}$$

where,

- $m(0)$ is marking at time t = 0.
- B is the incidence matrix. If all the discrete nodes are numbered before continuous nodes, then B can be summarized as $B = \begin{bmatrix} B_D & B_{DC} \\ B_{CD} & B_C \end{bmatrix}$.

Where B_D captures weight of arcs among discrete nodes, B_C captures weight of arcs among continuous nodes, B_{CD} captures weight of arcs between continuous place and discrete transition and B_{DC} captures weight of arcs between discrete place and continuous transition. But since equation 2.4 has to be satisfied, B_{DC} will always be zero. Hence incidence matrix can be further reduced to $B = \begin{bmatrix} B_D & 0 \\ B_{CD} & B_C \end{bmatrix}$.

Below example of the timed hybrid Petri net will give a good understanding about its working. At time t= 0, discrete transition t_1 and continuous transition t_3 are enabled. Note that continuous transition t_3 is strongly enabled. Transitions t_2 and t_4 are not enabled because p_2 does not have any tokens. Since t_1 and t_3 are enabled at the same time, discrete transition will be eligible to fire (whenever a continuous and discrete transitions are enabled at the same time, firing of discrete transition takes precedence over continuous transition). But since there is a delay associated with t_1 ($d_1 = 90$), tokens from p_1 will not be transfered to p_2 till $t = 90$. So at $t = 0$, transition t_3 will fire. And firing will move tokens from p_3 to p_4 at 2 tokens/time unit. Hence at the end of $t = 35$ time units, all tokens from p_3 will be successfully

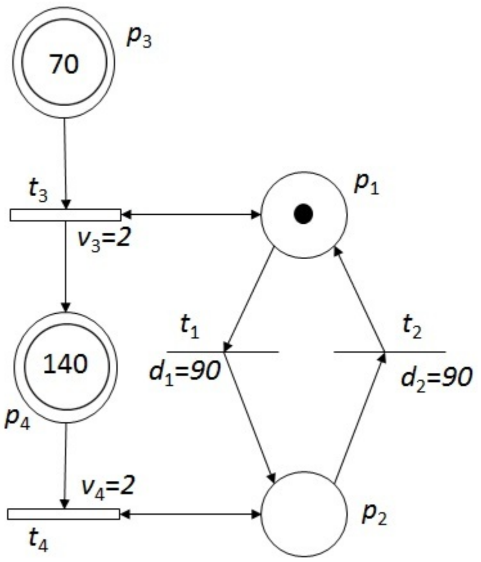


Fig. 2.9. Timed hybrid Petri net.

transfered to p_4. And p_4 will have $140 + 70 = 210$ tokens. From $t = 35$ to $t = 90$ no transitions will fire because,

- p_3 does not have tokens, so t_3 won't fire.
- t_4 and t_2 won't fire because there is no token at p_2.
- t_1 won't fire because it is waiting for $t = 90$.

When $t = 90$, token from p_1 will be transfered to p_2. Hence transitions t_2 and t_4 will become enabled. Since both transitions are active at the same time, firing of t_2 takes precedence. But since there is a delay with t_2 $(d_2 = 90)$, t_2 won't fire until $t = 180$. So t_4 will fire at $t = 90$. Since $v_4 = 2$, two tokens will be moved at a time from p_4 to p_3. After 90 time units, a total of 180 tokens will be transfered to p_3 from p_4. When $t = 180$, transition t_2 will fire. Hence there will be no token at p_2 (so t_4 and t_2 will no longer be active). But t_1 and t_3 will become enabled. Transition t_1 will fire only at time $t = 270$. So t_3 will fire at $t = 180$. And after 90 time units, 180 tokens would be transfered to p_4 from p_3. So at the end of $t = 270$, t_1 will fire (moving one token from

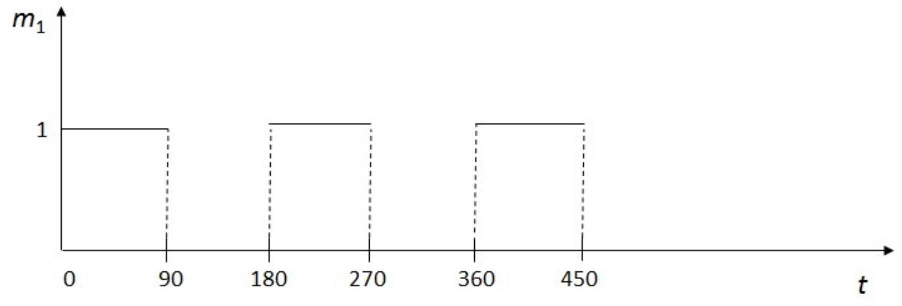


Fig. 2.10. Marking of timed hybrid Petri net (discrete node).

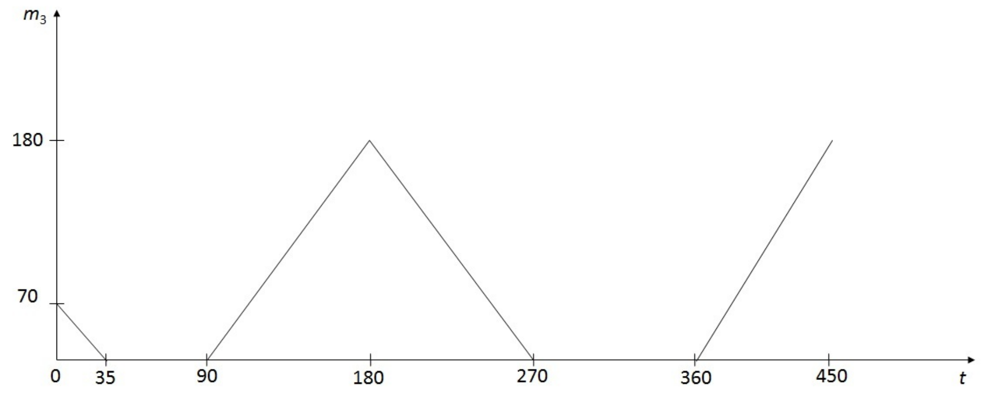


Fig. 2.11. Marking of timed hybrid Petri net (continuous node).

p_1 to p_2) and p_4 will again have 210 tokens (180 + leftover 30 from previous cycle). This process will continue.

All of the above sections introduces various Petri nets and their basic properties. But, in depth details about Petri net and it's working could be found in [3–7].

2.10 Conflict in a Petri net

As mentioned in some of the previous sections, if a place serves as input to two transitions, then there is a good chance that both the transition will be enabled at the same time. But in a Petri net, only one transition will fire at a time. Such a situation will result in a conflict. Conflict will arise to determine the transition which will fire first. In Fig. 2.4, at the initial marking $[1 \quad 0 \quad 0 \quad 1]^T$ transitions t_1 and t_3 are enabled. But only one will fire at a time. Nothing could be told with certainty as to which one will fire first. Both transitions have equal chances of firing first.

However, if there is a conflict between a continuous transition and a discrete transition, then the discrete transition takes precedence over continuous transition. In most cases, conflict among discrete transitions could be resolved by assigning delays to the transition. For example, if two discrete transitions are enabled by a place at the same time, then both will have equal chance of firing first. But if the designer wants t_1 to fire before t_2, then associating a delay with t_2 will make sure that t_1 will always fire first.

2.11 Simulation tool

Simulation is an important technique in understanding the behavior of a model during design phase. Only after the simulation of a model yields satisfactory results, the design process is carried forward. For simulating Petri nets described in this thesis, we use a tool called SimHPN. It is a MATLAB embedded software. The tool allows simulation of timed hybrid Petri net under different server semantics. To simulate a model, the tool requires knowledge of B^+, B^-, initial marking (m_0), firing rate (λ) of the transition, and the type of transition. These data could be given to the tool in many ways. It could be entered directly to SimHPN GUI, it could be given in the form of an input file or it could be entered from MATLAB prompt. Once the data is entered, the model could be simulated from SimHPN GUI. The simulation result (marking of a place or flow of a transition) could be seen in the GUI. The result could also be exported as a file. More information regarding SimHPN could be found at [10].

3. OPTIMIZATION

3.1 Objective

Optimization is a mathematical concept. It involves the process of selecting best available option from a set of options, under certain circumstances. In most situations an optimization problem involves maximizing or minimizing an objective function, subject to certain parameters. This is done by developing an optimization algorithm and repeating it multiple times (under various circumstances) till the desired result is achieved. A typical optimization problem can be represented as:

$$\min f(x)$$
$$\text{Subject to } g(x) \geq 0$$

3.2 Optimal problem formulation

Whenever there is scope to streamline a process and thus to increase its efficiency, an optimization problem arises. Simplest form of optimization process would involve comparing few solutions (formulated from prior knowledge of the problem). Further investigation is carried out regarding the feasibility of each solution. For each solution, the underlying objective function value is estimated. The estimates are compared to determine the best solution.

It is difficult to come up with a simple formulation process for all the problem, because the underlying function of the optimization problem and its design parameters vary from problem to problem. Thus we can only come up with a general model. The model must be solved using an algorithm for an optimal solution. Below flow chart shows various steps involved in an optimal design/solution formulation.

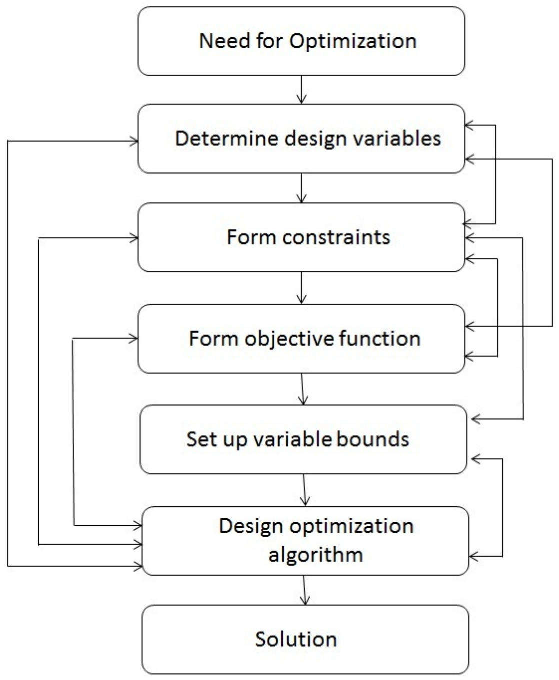


Fig. 3.1. Optimization process flow chart.

3.3 Design variables

First step in an optimal design problem is determining design variables. All variables which affect the process we are trying to optimize are good contenders for design variables. But only a few variables are highly sensitive to the proper execution of the design. These variables are selected as design variables. And these variables are varied during the design process.

It is always best to choose as few design variables as possible. The rest of the variables which affects the design (but still have minor impact on the design) is kept constant or they are varied in accordance to the design variable. At the end of the process, depending on the solution, the optimization process might be altered to include/delete/modify the existing design variables.

3.4 Constraints

In general, constraints represent some form of relationship between various parameters. In an optimization process, constraints indicate relationship between design variables and other parameters (which are not chosen as design variables). There is no exact rule about the number and nature of constraints that could be used in the optimization process. It varies from process to process. In most cases, there are two types of constraints (i) equality constraints (ii) inequality constraints. Out of the two, inequality constraints are the most commonly used constraints. Because inequality constraints are more relaxed. These type of constraints use inequality operator to express the relation between design variables and parameters which affect the process. $\geq, <, >$ and, $\geq$ are the most commonly used inequality constraint operators.

An example for an inequality constraint would be: in an internal combustion engine, during the compression and ignition stroke the pressure build up around the cylinder should be less than or equal to the maximum pressure of the material. This constraint could be translated into mathematical equation as,

$$P(x) \leq P_{allowable}$$

Another example would be: while designing a second order system, the natural frequency of oscillation should be greater than or equal to 3Hz. The constraint could be translated as,

$$f(x) \geq 3$$

On the other hand, equality constraints use equality operator to express relation between design variables and parameters which affect the process. And so, the functional relationship has to exactly match a value. An example of such a constraint would be: rise time of a system has to be 3 time units. So the constraint could be represented as,

$$t_r = 3 \tag{3.1}$$

Since equality constraints are more difficult to work with, they are often relaxed and transformed into inequality constraints. For example, equation 3.1 could be made into an inequality constraint as shown below,

$$t_r < 4$$
$$t_r > 2$$

3.5 Objective function

Objective function represents the process which we are trying to optimize. Most objective functions are expressed in terms of mathematical expressions. And the objective function represent the process in terms of design variables. In short, objection function is the mathematical representation of the process which is being optimized. And the main purpose of the optimization process could be minimization or maximization of the objective function. Example of an objective function would be,

$$\min(x^2 + 1)$$

General examples of objective functions would be: minimize cost, maximize efficiency, minimize time etc.

Most objective functions are minimization function. However, a maximizing function can be converted into minimizing function by multiplying the objective function by -1. Similarly, a minimizing function can be converted into maximizing function by doing the same process. But converting a minimizing function into maximizing function is seldom used.

Even though most process can be expressed in terms of mathematical expression, in some cases, certain processes might have to be approximated in order to get a mathematical expression. Designers are known to make assumptions while formulating objective function. As long as the assumptions are true, such approximations does not affect optimization. And technically, more than one objective function could be optimized at a time. But such optimization process tend to be complex. Hence, to make such optimization process simpler, the most important function will be chosen as objective function and the other functions will be included as a constraints.

3.6 Variable bounds

During the optimization process, the value of design variables are varied. So there is a good chance that the design variables will grow to extremes. In order to avoid such a situation, each of the design variables are varied within bounds (lower bound and upper bound). For example, while designing the controller for a second order system, if one of the design variable is rise time, t_r, then during optimization process, rise time will be varied within bounds. So, rise time will always stay within a lower bound and an upper bound. Varying rise time within bounds makes sure that the controller doesn't take too long to respond to a situation.

$$t_r^L \leq t_r \leq t_r^U$$

Although it is not mandatory to have variable bounds for each design variable, it is often a good practice to vary them within bounds. After finding the optimal solution, if the design variable does not fall within the variable limits, then the limits could be adjusted and the rest of the process could be continued.

If there are n design variables, then we can summarize the optimization problem as,

$$\begin{aligned} &\text{Minimize } f(x) \\ \text{Subject to } &g(x)_j \geq 0 \text{ for } j = 1, 2....j \\ &h(x)_k = 0 \text{ for } k = 1, 2....k \\ &x_i^L \leq x_i \leq x_i^U \text{ for } i = 1, 2....n \end{aligned}$$

3.7 Optimization algorithm

Algorithm used to optimize a process depends mainly on the function which is being optimized. However, objective functions can be broadly classified into:

- Linear progrmming (LP): such objective function will be a linear function. And the constraint set will contain linear terms. Solution to such problems could be found out by searching through a finite number of feasible points.
- Non-linear programming (NLP): objective function and/or the constraints set of such function will contain non-linear terms.

There is no single algorithm which will solve all optimization problem. The optimization algorithm to be used depends on the problem at hand. Since all the problems/algorithms are vastly different from each other, there are number of ways to classify optimization algorithms. Few of them are discussed below.

- Single variable optimization algorithm: as the name suggests, these algorithms will optimize an objective function which has only one variable. The function will be of the form,

  $$\min f(x)$$

  Different values of x (in an objective function) will yield different local maximum, local minimum, global maximum, global minimum. To find out these values, a single valued optimization algorithm will only search in one dimensional space. And the algorithm will either use the value of objective function to guide the search process (direct search) or it will use the derivatives of objective function to guide the search process (gradient search).
- Multi variable optimization algorithm: these algorithms will optimize an objective function with multiple design variables. And hence such algorithms will have to search in multiple dimensions. And as in previous case, the search can be a direct search or a gradient search. In many applications involving multi variable optimization, single variable optimization algorithms are used to search for optimum value in one dimension.
- Constrained optimization algorithm: in this method, algorithm will try to optimize the objective function in the company of constraints. Based on the number

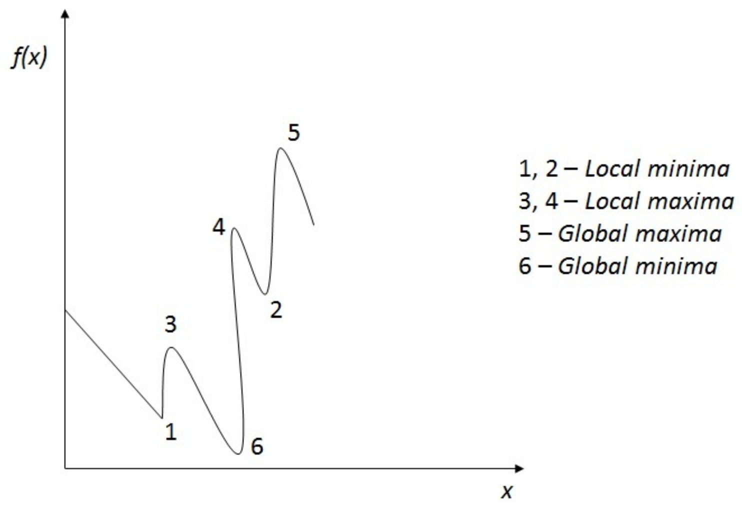


Fig. 3.2. Graphical representation of an objective function.

of dimensions (in space) in which search for optimum value is being carried out, this type of algorithm can employ single and multi variable algorithms simultaneously. If an optimization problem has n design variables and m constraints, then,

- Primal method: search for optimum value occurs in $n - m$ dimensional space.
- Penalty method: searches for optimum value in n dimensional space.
- Dual and cutting plane method: searches for optimum value in m dimensional space.
- Lagrangian method: searches in $n + m$ space.

There are many other optimization algorithms which does not fall into any of these categories. Details about them can be found in Chong, Zak [15].

4. HYBRID PETRI NET MODEL OF A TRAFFIC INTERSECTION

With the increasing amount of vehicles on the road, traffic control and traffic safety is an important aspect of transportation system. Introduction of traffic signs and signals have improved the safety of the system considerably. But controlling traffic during peak hours still remains a complicated issue. Major factor contributing to the problem is, lack of communication among traffic lights. Traffic light at one intersection do not convey the traffic volume to the traffic light at nearby intersection. Since traffic lights don't communicate with each other, the system won't work at its full efficiency. And as a result people lose a lot of time waiting at the intersection. For instance, if there are no vehicles waiting at an upstream intersection, there won't be any vehicle traveling from upstream intersection to downstream. Even then, the signal at downstream intersection would hold the traffic for a fixed time interval. If somehow the traffic signals were communicating to each other, then duration of signal at downstream intersection could have been reduced. And the process would have saved valuable time of vehicles waiting to cross the intersection. In this chapter, we will try to understand the traffic model in place. Following that, we will develop an algorithm which will optimize the duration of traffic signal.

A Petri net model which captures traffic flow and traffic signal uses continuous as well as discrete nodes. While continuous nodes capture vehicle flow, discrete nodes are used to capture signal change. Since the traffic signals hold their state for certain time, the discrete nodes (which represent the signals) are usually timed nodes. Fig. 4.1 shows an intersection of two one-way street controlled by one traffic light. The discrete nodes $p_1, p_2, p_3, p_4, t_1, t_2, t_3, t_4$ captures the signal change. A token in p_1 would indicate that intersection I is active (signal is green). And during this time intersection II will be inactive (signal is red). A token in p_3 would indicate that intersection II

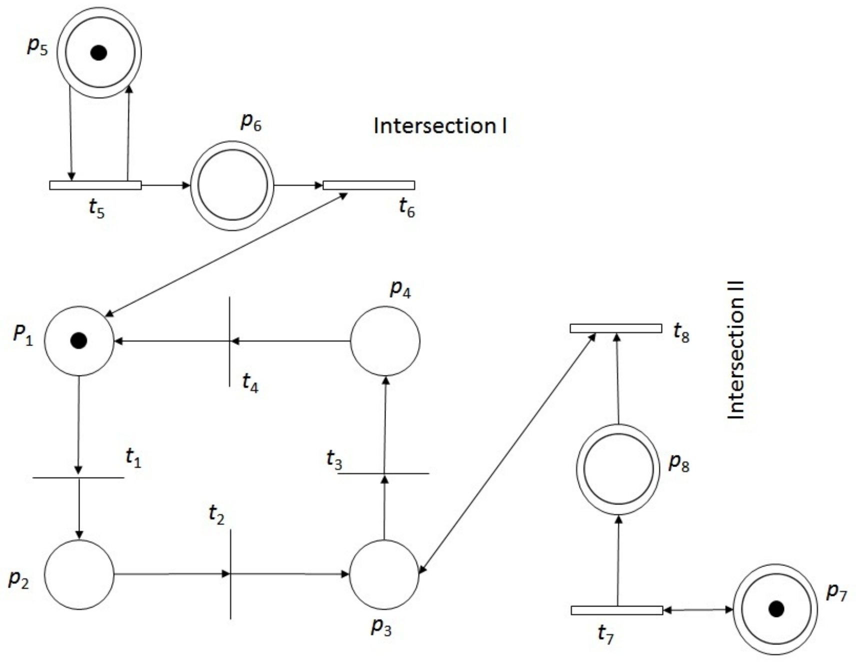


Fig. 4.1. Petri net model of two one-way intersection.

is active and intersection I is inactive. A token at p_2/p_4 would indicate yellow/red period for intersection I/II respectively. So during this period, no queue will be served. The transitions t_1, t_2, t_3, t_4 represent the switching of traffic signals. These transitions have deterministic time delays associated with them. Because of the deterministic time delay, their input places will remain active for the duration of delay.

The continuous nodes $p_5, p_6, p_7, p_8, t_5, t_6, t_7, t_8$ represent vehicles moving in and out of the intersection. Tokens in places p_6, p_8 represent vehicles waiting at the signal to cross the intersection. And transitions t_5, t_6, t_7, t_8 enables vehicle arrival and departure to and from the intersection. Queue I will be active only when transition t_6 is enabled. And transition t_6 will be enabled only if there is a token at p_1 (green signal for intersection I). Thus vehicles waiting at queue I will move towards the intersection only when its signal is green (p_1 has tokens). Similarly vehicles waiting at queue II will move towards the intersection only when its signal is green (p_3 has tokens). Tokens at p_5, p_7 replicate vehicles coming to intersection from upstream.

Since p_5 and p_7 always have token in them, their corresponding transitions will be always active. Hence vehicles will arrive at the intersection continuously at a constant rate (depending on the firing rate of t_5 and t_7). And finally, the incidence matrices of the model are,

$$B^+ = \begin{bmatrix} 0&0&0&1&0&1&0&0 \\ 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&1 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1&0 \end{bmatrix} \text{ and } B^- = \begin{bmatrix} 1&0&0&0&0&1&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&1 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix}$$

4.1 Background and related work

Lefeber, Rooda [9] studied a model similar to Fig. 4.1. Their model consists of two queues being served by a single server. And in their model, the arrival rate and service rate of the queues were constant. If you consider their model to Fig. 4.1, the tokens in p_6, p_8 could be regarded as two queues and the traffic signal can be considered as a single server. The arrival rate of tokens to p_6 and p_8 is constant (depends on the firing rates of t_5 and t_7) and the departure rate of tokens are also constant (firing rate of t_6 and t_8). The result of the study was characterized as the optimal steady state periodic orbit with cost function,

$$J = \frac{1}{T_{ss}} \int_0^{T_{ss}} [x_1(\tau) + x_2(\tau)] d\tau \tag{4.1}$$

where,

- x_1 and x_2 represents the queue (marking of p_6 and p_8).
- T_{ss} is the period of the orbit.

Since characteristics of model in Fig. 4.1 is similar to the model studied by Lefeber, Rooda [9], equation 4.1 will represent the objective function of the process defined by Petri net in Fig. 4.1. However, switching of traffic signal is modeled by discrete nodes, so the calculation of cost function has to be modified accordingly (cost function in the equation 4.1 is a continuous function. But switching of signal is not a continuous function. It is a discrete function). In order to accomplish this, cost function is calculated for each discrete state.

For example, when $p_1 = 1$ (transition t_1 is enabled). And if firing rates of t_5, t_6, t_7, t_8 are $\lambda_5, \lambda_6, \lambda_7, \lambda_8$ respectively, then marking of places p_6 and p_8 could be computed by,

$$m(p_6) = (\lambda_6 - \lambda_5)\Delta t + p_6^{prev} \tag{4.2}$$

$$m(p_8) = \lambda_7 \Delta t + p_8^{prev} \tag{4.3}$$

where,

- Δt is the time duration for which the system will remain the same discrete state.
- p_6^{prev}, p_8^{prev} are the initial values of p_6 and p_8. $[m(p_6), m(p_8)]$.

And so, the cost function can be written as,

$$\Delta J = \int_0^{\Delta t} W \begin{bmatrix} m(p_6) \\ m(p_8) \end{bmatrix} dt = w_1 \int_0^{\Delta t} m(p_6)dt + w_2 \int_0^{\Delta t} m(p_8)dt$$

where w_1 and w_2 are some optimization weights.

If $w_1 = w_2 = 1$, then,

$$\Delta J = \int_0^{\Delta t} W \begin{bmatrix} m(p_6) \\ m(p_8) \end{bmatrix} dt = \int_0^{\Delta t} m(p_6)dt + \int_0^{\Delta t} m(p_8)dt$$

And from equations 4.2 and 4.3,

$$\int_0^{\Delta t} m(p_6)dt = \left(\frac{\lambda_6 - \lambda_5}{2}\right)(\Delta t)^2 + (p_6^{prev})\Delta t$$

$$\int_0^{\Delta t} m(p_8)dt = \frac{\lambda_7}{2}(\Delta t)^2 + (p_8^{prev})\Delta t$$

Similarly, the cost function could be calculated for all discrete states. Once calculation for all cost functions are determined, we can write an algorithm to sum up all the above mentioned steps.

```
Begin algorithm
  1. Define the firing rates of all the continuous transition. Define time delays for all
  the discrete transitions. Define working variables and initialize arrays and flags.
  for all $t \leq T$ do
    2. Find the enabled discrete transitions.
    3. Find the remaining time at the current discrete state: $\Delta t$.
    4. Determine firing time of the current enabled discrete transition.
    5. Calculate the markings (queues) of $m(p_6), m(p_8)$ during time $\Delta t$.
    6. Calculate the incremental cost: $\Delta J$.
    7. Update cost function $J = J + \Delta J$.
    8. Update the firing time $t = t + \Delta t$.
    9. Fire the enabled discrete transition.
  end for
  10. Calculate the cost function, $J(\tau) = \frac{1}{T}.J$
  End algorithm
```

So far, determination of current discrete state and evaluation of cost function were done separately. But those two processes could be combined into a single process. Both processes when combined together can be elaborated as:

- Define the firing rate for continuous transitions as $(\lambda_5, \lambda_6, \lambda_7, \lambda_8)$. And define the time delays for discrete transitions as $(\theta_1, \theta_2, \theta_3, \theta_4)$. Define current time $t = 0$, cost function $J_{ac} = 0$, marking of place $p_1 = 1$, and cycle time T. And then, initialize/define necessary flags and define arrays.

- While $t \leq T$, do,

  – Determine the discrete place which has token. The place which has token will enable the corresponding discrete transition. Since only one discrete place will have token at a time, there will be only one enabled discrete transition at a time. And thus, this process makes sure that only one intersection will be active at a time.

Once current discrete state has been determined, the delay associated with current discrete state has to be looked at, because the newly enabled transition will fire only after a specific time interval (θ, it is a timed discrete transition).

Update the firing time of the enabled transition as: current time + delay associated with the transition. So,

- if $p_1 = 1$, then,
  * Enabled transition = 1
  * firing time $= t + Delays[j]$

Do the same check for p_2, p_3, and p_4 in order to find out current enabled discrete transition and to determine their firing time. Once the current discrete state is determined, compute the markings $[m(p_6), m(p_8)]$ and, cost function using equation 4.2, 4.3 as example.

if $p_1 = 1$ then,

$$m(p_6) = (\lambda_6 - \lambda_5)\Delta t + p_6^{prev}$$

$$m(p_8) = \lambda_7 \Delta t + p_8^{prev}$$

$$J_6 = \left(\frac{\lambda_6 - \lambda_5}{2}\right)(\Delta t)^2 + (p_6^{prev})\Delta t$$

$$J_8 = \frac{\lambda_7}{2}(\Delta t)^2 + (p_8^{prev})\Delta t$$

if $p_3 = 1$ then,

$$m(p_6) = \lambda_5 \Delta t + p_6^{prev}$$

$$m(p_8) = (\lambda_8 - \lambda_7)\Delta t + p_8^{prev}$$

$$J_6 = \frac{\lambda_5}{2}(\Delta t)^2 + (p_6^{prev})\Delta t$$

$$J_8 = \left(\frac{\lambda_8 - \lambda_7}{2}\right)(\Delta t)^2 + (p_8^{prev})\Delta t$$

if p_2 or $p_4 = 1$ then,

$$m(p_6) = \lambda_5 \Delta t + p_6^{prev}$$
$$m(p_8) = \lambda_7 \Delta t + p_8^{prev}$$
$$J_6 = \frac{\lambda_5}{2}(\Delta t)^2 + (p_6^{prev})\Delta t$$
$$J_8 = \left(\frac{\lambda_7}{2}\right)(\Delta t)^2 + (p_8^{prev})\Delta t$$

- Update cost function as $J_{ac} = J_{ac} + J_6 + J_8$.
- Reset temporary cost functions to zero. $J_6 = J_8 = 0$.
- Update current time as $t = t + \Delta t$.
- Mark the transition which fired as not an enabled transition. And make the downstream discrete transition as an enabled transition. So, if enabled transition is t_1 then, set $p_1 = 0$ and $p_2 = 1$. Carry out the same process for p_2, p_3, and p_4.

- End loop.
- Evaluate the final cost. $J = \frac{J_{ac}}{T}$.

Below example demonstrates working of the algorithm with an example. If the delays of discrete transitions are $\theta_1 = 20,\ \theta_2 = 5,\ \theta_3 = 40,\ \theta_4 = 5$. And the rates of the discrete transitions are $\lambda_1 = 1,\ \lambda_2 = 3,\ \lambda_3 = 1,\ \lambda_4 = 3$. Previous markings of p_6, p_8 are 10 and 20 respectively. And let cycle time, $T = 70$. For the initial condition $p_1 = 1$, transition t_1 is enabled. At time $t = 0$, transition t_1 is enabled. Delay associated with t_1 is 20 time units. Since, only one transition is enabled at a time, duration of the current state = delay of the enabled transition. Hence $\Delta t = 20$.

$$m(p_6) = (3-1)20 + 10 = 50$$
$$J_6 = \frac{(3-1)}{2}20^2 + (10 \times 20) = 600$$
$$m(p_8) = (1 \times 20) + 20 = 40$$
$$J_8 = \left(\frac{1}{2}\right) 20^2 + (20 \times 20) = 600$$

At the end of 20 time units, t_1 will fire and the process will transfer one token to p_2. Hence t_1 won't be enabled anymore. Since p_2 has a token, t_2 will be enabled. And the delay of t_2 is five time units. At $t = 20, \quad p_2 = 1, \quad \Delta t = 5$ time units. So,

$$\begin{aligned} m(p_6) &= (1 \times 5) + 10 = 15 \\ J_6 &= \left(\frac{1}{2}\right) 5^2 + (10 \times 5) = 62.5 \\ m(p_8) &= (1 \times 5) + 20 = 25 \\ J_8 &= \left(\frac{1}{2}\right) 5^2 + (20 \times 5) = 112.5 \end{aligned}$$

Five time units after, transition t_2 will fire. As a result of that, one token will be transferred from p_2 to p_3. So t_2 is no longer enabled. But t_3 is enabled. Delay associated with t_3 is 40. So, at $t = 25, \quad p_3 = 1, \quad \Delta t = 40$.

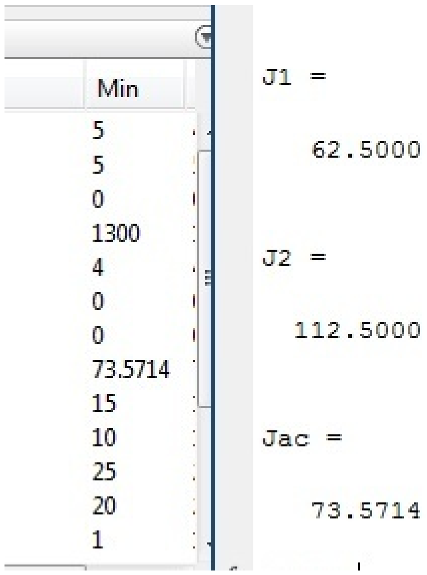
$$\begin{aligned} m(p_6) &= (1 \times 40)+10 = 50 \\ J_6 &= \left(\frac{1}{2}\right) 40^2+(10 \times 40) = 1200 \\ m(p_8) &= (3-1)40+20 = 100 \\ J_8 &= \frac{(3-1)}{2}40^2+(20 \times 40) = 2400 \end{aligned}$$

At the end of $t = 25 + 40 = 65$ time units, t_3 will fire. Hence one token will be transfered from p_3 to p_4. Thus t_3 won't be enabled anymore. But t_4 will be enabled. And delay of t_4 is five. So,

$$\begin{aligned} m(p_6) &= (1 \times 5)+10 = 15 \\ J_6 &= \left(\frac{1}{2}\right) 5^2+(10 \times 5) = 62.5 \\ m(p_8) &= (1 \times 5)+20 = 25 \\ J_8 &= \left(\frac{1}{2}\right) 5^2+(20 \times 5) = 112.5 \end{aligned}$$

Five time units after t_4 has become enabled, the transition will fire. Thus there won't be anymore token at p_4. Instead, the token will be available at p_1. Thus t_1 will be enabled again. And this cycle will continue.

Cost function value is the weighted sum of J's incurred during each discrete state. So, in this case, $J = \frac{(600+600+62.5+112.5+1200+2400+62.5+112.5)}{70} = \frac{5150}{70} = 73.5714$. To verify the result, program is executed for the same parameters. Below picture shows the result of execution.



```
Min
5
5
0
1300
4
0
0
73.5714
15
10
25
20
1
```

```
J1 =

   62.5000


J2 =

  112.5000


Jac =

   73.5714
```

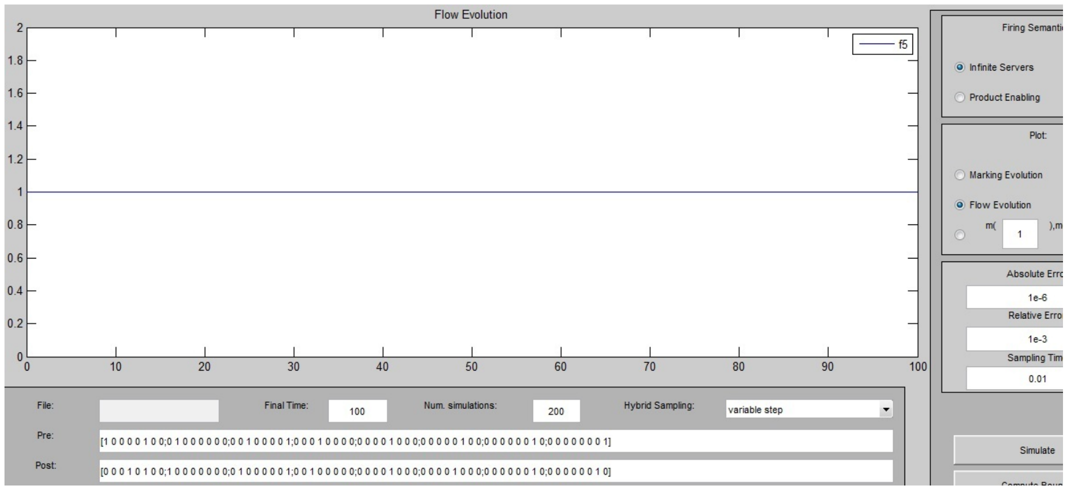
Fig. 4.2. Optimization result of constant arrival model.

4.2 Simulation results

In an effort to better understand the model behavior, Petri net model of the intersection (shown in Fig. 4.1)is simulated. Some of the simulation results are shown in this section.

Fig. 4.3 is the simulation result which shows arrival of vehicles at intersection I. It shows that vehicles arrive at the intersection at constant rate. The arrival rate of vehicle at intersection I, in this model, depends on firing rate of t_5. Same is the case for intersection II. Arrival at intersection II happens at constant rate. Hence graph of t_7 will yield the same result.

Fig. 4.4 is the simulation showing the flow of vehicle at intersection II (i.e t_8). Transition t_8 has two input places (i) continuous place p_8 (ii) discrete place p_3. Place



Infinite Servers [✓] Product Enabling

Marking Evolution [✓ Flow Evolution] m(1),m

Absolute Erro: 1e-6

Relative Erro: 1e-3

Sampling Tim: 0.01

File:

Final Time: 100

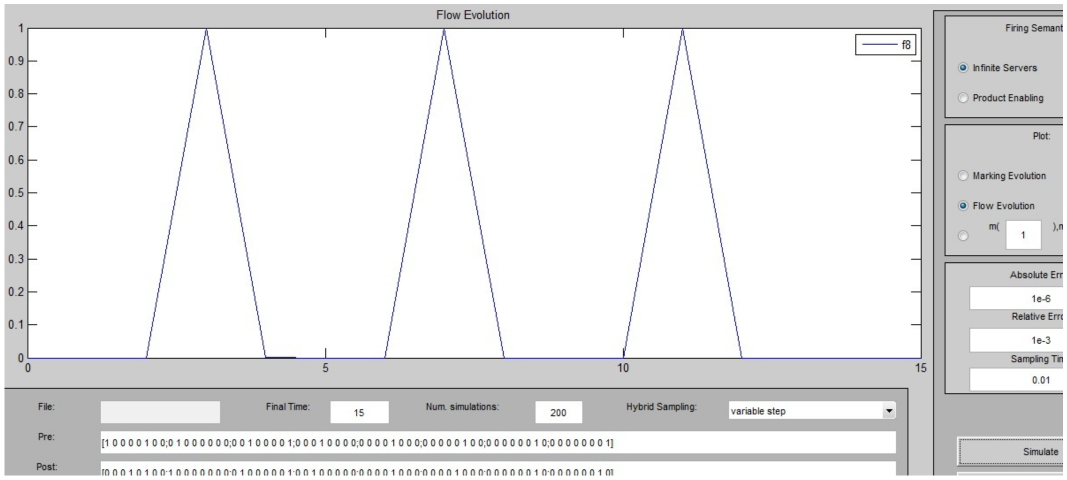
Num. simulations: 200

Hybrid Sampling: variable step

Pre: [1 0 0 0 0 1 0 0;0 1 0 0 0 0 0 0;0 0 1 0 0 0 0 1;0 0 0 1 0 0 0 0;0 0 0 0 1 0 0 0;0 0 0 0 0 1 0 0;0 0 0 0 0 0 1 0;0 0 0 0 0 0 0 1]

Post: [0 0 0 1 0 1 0 0;1 0 0 0 0 0 0 0;0 1 0 0 0 0 0 1;0 0 1 0 0 0 0 0;0 0 0 0 1 0 0 0;0 0 0 0 1 0 0 0;0 0 0 0 0 0 1 0;0 0 0 0 0 0 1 0]

Fig. 4.3. Result depicting constant arrival rate.



Infinite Servers [✓] Product Enabling

Marking Evolution [✓ Flow Evolution] m(1),m

Absolute Err: 1e-6

Relative Erro: 1e-3

Sampling Tin: 0.01

File:

Final Time: 15

Num. simulations: 200

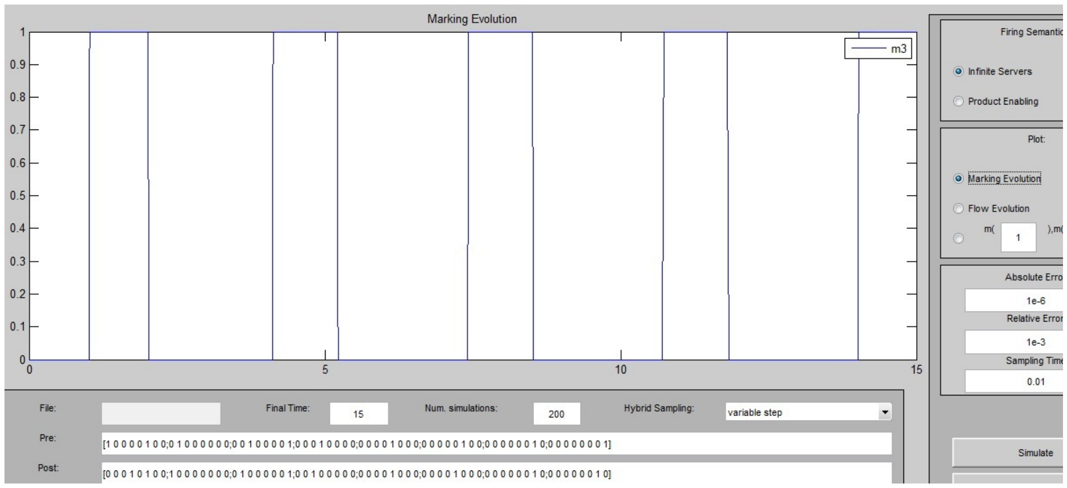
Hybrid Sampling: variable step

Pre: [1 0 0 0 0 1 0 0;0 1 0 0 0 0 0 0;0 0 1 0 0 0 0 1;0 0 0 1 0 0 0 0;0 0 0 0 1 0 0 0;0 0 0 0 0 1 0 0;0 0 0 0 0 0 1 0;0 0 0 0 0 0 0 1]

Post: [0 0 0 1 0 1 0 0;1 0 0 0 0 0 0 0;0 1 0 0 0 0 0 1;0 0 1 0 0 0 0 0;0 0 0 0 1 0 0 0;0 0 0 0 1 0 0 0;0 0 0 0 0 0 1 0;0 0 0 0 0 0 1 0]

Fig. 4.4. Result showing departure of vehicle from intersection.

p_8 will always have tokens in it. So firing of t_8 depends on presence of token at p_3. And p_3 will have token in the place only at specific intervals of time. Hence t_8 will fire only during those intervals. This is the reason for discontinuity in the graph of a continuous transition. And ultimately, since t_8 is a continuous transition, flow from the transition occurs continually. Hence the flow is in the form of increasing/decreasing ramp instead of a discrete waveform. Simulation of t_6 will also give similar result.



Firing Semantic

[✓ Infinite Servers] Product Enabling

Plot:

[✓ Marking Evolution] Flow Evolution m(1),m(

Absolute Erro: 1e-6

Relative Error: 1e-3

Sampling Time: 0.01

File:

Final Time: 15

Num. simulations: 200

Hybrid Sampling: variable step

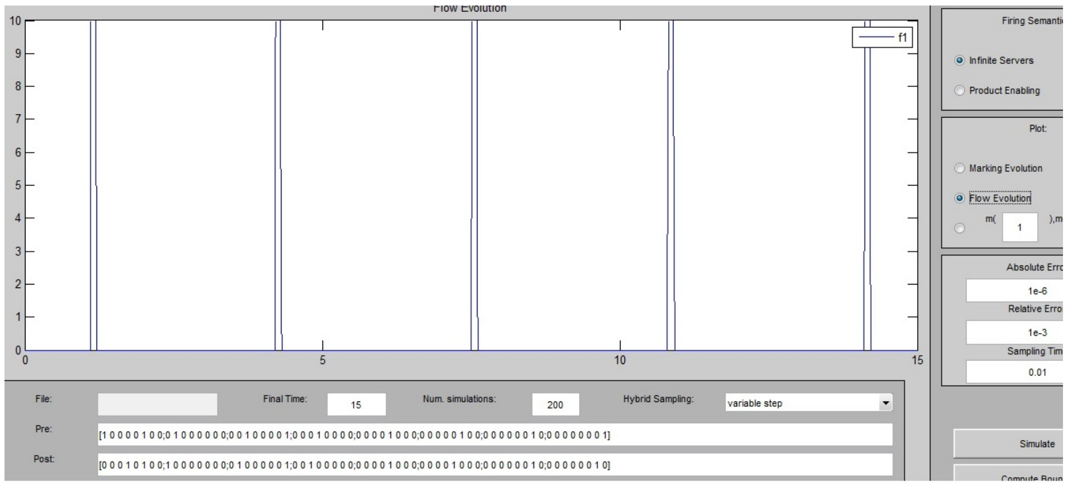
Pre: [1 0 0 0 0 1 0 0;0 1 0 0 0 0 0 0;0 0 1 0 0 0 0 1;0 0 0 1 0 0 0 0;0 0 0 0 1 0 0 0;0 0 0 0 0 1 0 0;0 0 0 0 0 0 1 0;0 0 0 0 0 0 0 1]

Post: [0 0 0 1 0 1 0 0;1 0 0 0 0 0 0 0;0 1 0 0 0 0 0 1;0 0 1 0 0 0 0 0;0 0 0 0 1 0 0 0;0 0 0 0 1 0 0 0;0 0 0 0 0 0 1 0;0 0 0 0 0 0 1 0]

Fig. 4.5. Result showing duration of traffic light.

Simulation in Fig. 4.5 shows the duration of a traffic light (i.e m_3). Since p_3 is a discrete place (with discrete input and output transition), graph will be in the form of a rectangular block (standard discrete waveform, having values one or zero). There is a delay associated with t_3, this will make the token to stay in p_3 till the delay time elapses. So for the duration of delay, graph will indicate one token in m_3. When the delay time elapses, token will be transfered to next place. And hence the graph will indicate zero token. The width of the graph depends on the delay associated with the transition t_3. Same can be expected from the graph of m_1.

Fig. 4.6 shows the flow of t_1. Since the transition is discrete, it will generate a standard discrete waveform. Transition will fire only when t_1 is enabled. In addition, it will wait for delay associated with t_1 to elapse. After that, the transition will fire. And the process will allow flow of token from p_1, through t_1, to p_2. Since it allows the flow of one token while firing, the waveform will be narrow. Same form of graph could be expected for the flow of t_2, t_3, t_4 as well.



[✓ Infinite Servers] Product Enabling

Marking Evolution [✓ Flow Evolution]

m(1),m

Absolute Erro: 1e-6
Relative Erro: 1e-3
Sampling Tim: 0.01

File:
Final Time: 15
Num. simulations: 200
Hybrid Sampling: variable step
Pre: [1 0 0 0 0 1 0 0;0 1 0 0 0 0 0 0;0 0 1 0 0 0 0 1;0 0 0 1 0 0 0 0;0 0 0 0 1 0 0 0;0 0 0 0 0 1 0 0;0 0 0 0 0 0 1 0;0 0 0 0 0 0 0 1]
Post: [0 0 0 1 0 1 0 0;1 0 0 0 0 0 0 0;0 1 0 0 0 0 0 1;0 0 1 0 0 0 0 0;0 0 0 0 1 0 0 0;0 0 0 0 1 0 0 0;0 0 0 0 0 0 1 0;0 0 0 0 0 0 1 0]

Fig. 4.6. Result showing firing of transition.

4.3 Traffic intersection with exponential arrival rate

As mentioned before, the arrival rate and service rate in Fig. 4.1 is constant. But in most situations, traffic intersections are controlled by traffic lights. And intersections are connected to each other through roads. Hence vehicles departing from one intersection will arrive at the neighboring intersection a little while later. When the traffic signal at the intersection turns from red to green, all the vehicles waiting to cross the intersection will leave the intersection. Since all vehicles starts leaving the intersection together, they all arrive at the neighboring intersection together. And hence, the traffic does not arrive at an intersection at a constant rate. It happens in batches (a batch arrives at an intersection shortly after the traffic light at upstream intersection turns green). So in an effort to make the traffic model more realistic, Vandzquez, Sutarto, Boel, and Silva [8] came up with a new model. Their model is shown in Fig. 4.7. The difference between model in Fig. 4.7 and model in Fig. 4.1 is in section I. Place p_5 of model (in Fig. 4.1) is replaced with section I in new model (in Fig. 4.7). There are two discrete places p_e, p_{ne} and two timed discrete transition t_9, t_{10} in the new section. A token in p_e means, transitions t_9 and t_1 are enabled. During this time, vehicles from upstream intersection will arrive at intersection 1

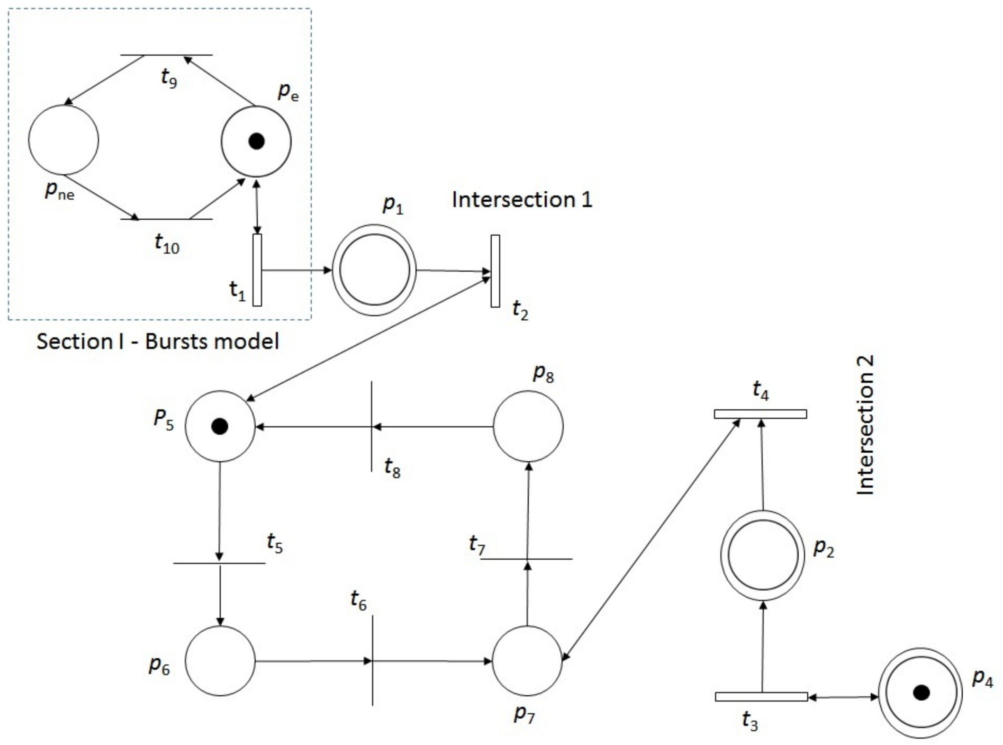


Fig. 4.7. Petri net model of two one-way intersection with exponential arrival rate.

with an arrival rate of λ. Since t_9 is a timed discrete transition, after a certain time θ_9 (delay associated with t_9) token from t_9 will be removed and will be deposited at p_{ne}. When this happens t_1 and t_9 will no longer be enabled (vehicles won't arrive at intersection 1). But at this time transition t_{10} will become enabled. No vehicle from upstream will arrive at intersection 1 during this time. If the deterministic time delay associated with t_{10} is θ_{10}, then θ_{10} time units after t_{10} became enabled, token will be removed from p_{ne} and it will be deposited back at p_e. And this cycle repeats. The whole process characterizes vehicles arriving at the intersection in batches. Because, the only time when vehicles are allowed to arrive at the intersection is when t_9 is enabled (for a duration of θ_9) as opposed to vehicles arriving continuously at the intersection (shown in Fig. 4.1). And if the arrival rate is λ_1, then the total number of cars arriving at the intersection from upstream will be $\theta_9\lambda_1$. The arrival at intersection 2 in the new model still occurs continuously and at a constant rate. The author has deliberately kept the arrival at constant rate to keep the model simple. Vehicle arrival at intersection 2 will occur in batches if p_4 is replaced with section I.

Since the model in Fig. 4.1 have constant arrival rate and the model in Fig. 4.7 do not have a constant arrival rate, equation 4.1 will not represent an optimal behavior of model shown in Fig. 4.7. In addition, the model in Fig. 4.1 has only one discrete structure (traffic light). And hence the discrete transitions will always fire in the following order $t_1, t_2, t_3, t_4, t_1, t_2$.... But for model in Fig. 4.7, there are two different discrete structures (traffic light, section I). Transition firing can follow many sequence. For example $t_9, t_5, t_6, t_{10}, t_9, t_7, t_8$.... or $t_5, t_9, t_6, t_{10}, t_7, t_8$... or $t_5, t_9, t_6, t_7, t_8, t_{10}$..... The transition firing sequence depends on the delay of the signal t_5 and t_7. And this uncertainty associated with transition firing sequence makes it harder to come up with a mathematical expression to represent the process.

4.4 Optimization of the signal

Since the firing sequence is not fixed for the model, a mathematical expression to represent the model is difficult to fabricate. Hence the optimal periodic orbit cannot be represented by the equation 4.1. In order to optimize the process of Fig. 4.7, we have to optimize the duration of traffic light (θ_5, θ_7). In order to optimize the process, we follow the same optimization steps mentioned in Chapter 3. So starting from the first step,

- Design variables - since we are trying to optimize duration of traffic signal, it is fair to make the assumption that θ_5, θ_7 are the major parameters affecting the optimization process. Hence θ_5, θ_7 are chosen as design variables.

- Constraint - there are not many parameters affecting the optimization process (other the duration of traffic signals). And the only constraint on the duration of traffic light (design variable) is that, it should be integers. i.e θ_5,θ_7 should be integers.

- Variable bounds - during the optimization process, θ_5,θ_7 will be varied within bounds. In other words, there exist a minimum value and a maximum value within which design variables(θ_5,θ_7) are varied. So Vandzquez, Sutarto, Boel, and Silva [8] has defined control value as,

$$cs = \{(\theta_5, \theta_7)\epsilon N \times N | \theta_5^{min} \le \theta_5 \le \theta_5^{max},\ \theta_7^{min} \le \theta_7 \le \theta_7^{max}\}$$

- Objective function - it is the cost function which we are trying to optimize

$$J(T, \theta_5, \theta_7) = \frac{1}{T}\int_0^T W \begin{bmatrix} m(p_1) \\ m(p_2) \end{bmatrix} dt \tag{4.4}$$

  where,

  - $m(p_1)$, $m(p_2)$ are length of the queues.
  - T is the fixed time horizon.

For a fixed time horizon T, and queue length $m(p_1)$ and $m(p_2)$, the design variables (θ_5,θ_7) are varied while computing the cost function in equation 4.4. Each time the cost function values are compared to find out the minimum cost value. And the minimum cost value determines optimal duration (θ_5^{opt}, θ_7^{opt}).

Thus given the time horizon T and current markings $m(p_1)$ and $m(p_2)$, the process will improve the duration of traffic signal over an interval of [current time, T time units from current time]. And also, the cycle time is not constant (because it depends on the values of (θ_5 and θ_7) obtained). The cost function of the model is computed for many combinations of (θ_5, θ_7). Hence the computational cost is high. In order to reduce that, the value of incremental cost function $J(T + \Delta T) - J(T)$ is calculated (where Δt is the time interval for which the system will remain in the same discrete state). Below process illustrates the calculation of cost function.

Let $(\theta_5, \theta_6, \theta_7, \theta_8, \theta_9, \theta_{10})$ be the delays of transitions $(t_5, t_6, t_7, t_8, t_9, t_{10})$. And let the delay values be $(20, 4, 35, 4, 10, 30)$ and $m(p_5) = m(p_e) = 1$ i.e, transitions t_1, t_5, t_9 are enabled. But $\min(\theta_5,\ \theta_9) = \min(20, 10) = 10$. So the system will remain in the same discrete state for 10 time units (after that the system will move to another discrete state). During this time, marking of p_1 and p_2 will change and it will follow a deterministic pattern. For instance if $(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$ are the firing rates of transitions (t_1, t_2, t_3, t_4), and if $\Delta t \leq t_c$ then the change in marking during Δt can be calculated by,

$$m_{\Delta t}(p_1) = (\lambda_1 - \lambda_2)\Delta t + m_0(p_1)$$

But if $\Delta t > t_c$, then,

$$m_{\Delta t}(p_1) = \frac{\lambda_1}{\lambda_2} + \left(1 - \frac{\lambda_1}{\lambda_2}\right) e^{-\lambda_2(t-t_c)}$$

where Δt_c is the time required by the first queue to reach a value of one. So,

$$t_c = \frac{(1 - m_0(p_1))}{\lambda_1 - \lambda_2}$$

and thus, the incremental cost function (as in previous model) is,

$$\Delta J = \int_0^{\Delta t} W \begin{bmatrix} m(p_1) \\ m(p_2) \end{bmatrix} dt = w_1 \int_0^{\Delta t} m(p_1)dt + w_2 \int_0^{\Delta t} m(p_2)dt \tag{4.5}$$

But, if $\Delta t > t_c$

$$\int_0^{\Delta t} m(p_1)dt = \frac{\lambda_1 - \lambda_2}{2}(\Delta t)^2 + m_0(p_1)\Delta t + \frac{\lambda_1}{\lambda_2}(\Delta t - t_c) + \left(\left(\frac{\lambda_1}{\lambda_2^2}\right) - \frac{1}{\lambda_2}\right)(e^{\lambda_2(t_c - \Delta t)} - 1) \tag{4.6}$$

And if $\Delta t \leq t_c$ then, markings can be found out by,

$$\int_0^{\Delta t} m(p_1)dt = \frac{\lambda_1 - \lambda 2}{2}(\Delta t)^2 + m_0(p_1)\Delta t$$

and since arrival rate at intersection 2 is kept constant,

$$\int_0^{\Delta t} m(p_2)dt = \frac{\lambda_3}{2}(\Delta t)^2 + m_0(p_2)\Delta t \tag{4.7}$$

will hold true under all circumstances.

Similarly, the expressions for markings $[m(p_1)$,$m(p_2)]$ and cost function (J) can be found out for all possible discrete states. Once the expressions are obtained, an optimization algorithm can be designed to minimize the cost function (and thus determine the optimum signal duration). The algorithm will sum up all the above mentioned steps.

```
Begin algorithm
  1. Define the firing rate for the continuous transition. Define the time delays for
  the discrete transitions. Define $t = 0$, $T$, $J_{ac} = 0$.
  for all t $\leq$ T do
    2. Find the enabled discrete transitions.
    if enabled-trans = $t_5$ or $t_7$ then
      3. Generate a random delay for $t_5$ and $t_7$.
    end if
    4. Determine the firing time of all the enabled transitions.
    5. Find the transition which will fire first(transition with the lowest firing time,
    from previous step, will fire first).
    6. Determine $\Delta t$. Transition will remain in the same discrete state during $\Delta t$.
    7. Calculate the markings (queues) of $m(p_1), m(p_2)$ during $\Delta t$.
    8. Calculate the incremental cost: $\Delta J$. If enabled transition is $t_5$ or $t_7$, do a
    linear search to determine the lowest $\Delta J$. If the current $\Delta J$ is the lowest, set
    delays of $t_5$ and $t_7$ as optimum signal duration.
    9. Update the firing time $t = t + \Delta t$.
    10. Fire the enabled discrete transition.
  end for
  11. Calculate the cost function, $J(\tau) = \frac{1}{\tau}.J_{ac}$
End algorithm
```

So far determining all discrete states, and calculation of the corresponding cost functions were done separately. But those two processes could be integrated into one as shown below:

- Define the firing rate for continuous transitions as $(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$. Define the time delays for discrete transitions as $(\theta_6, \theta_8, \theta_9, \theta_{10})$. Define current time $t = 0$, cost function $J_{ac} = 0$, cycle time T. Initialize necessary flags and define arrays.
- While $t \leq T$, do,
  - Determine the discrete places which has token. The place which has token will enable the corresponding discrete transition.
  - There will be multiple enabled transitions. If enabled transition is t_5 or t_7 then, randomly select a delay value (for t_5 and t_7) within an upper limit and lower limit. Let the randomly selected delays be θ_5 and θ_7.
  - Update the firing time of all the enabled transition as current time + delay associated with the transition. So,
  - if $p_5 = 1$, then,
    - Enabled transition[i] = 5
      - firing time $[i] = t + \theta_5$. This is the time at which this transition will fire.
      - $i = i + 1$

    Do the same check for p_6, p_7, p_8, p_e and, p_{ne}.
  - Compare firing time of all enabled transitions (determined in previous step). Transition with the lowest firing time will fire first. Rest of the transitions will fire later.
  - Δt = firing time − t. System will remain in the same discrete state for the duration of Δt.

After the current discrete state is established, compute markings $[m(p_1), m(p_2)]$, and cost function using equation 4.5, 4.6, 4.7 as example.
if $p_5 = 1$ then,

- $p_2^{cur} = \lambda_3 \Delta t + p_2^{prev}$

- $J_2 = \frac{\lambda_3}{2}(\Delta t)^2 + (p_2^{prev})\Delta t$

  if $p_e = 1$ then,

  * compute $t_c = \frac{1-p_1^{prev}}{\lambda_1-\lambda_2}$

    if $\Delta t \leq t_c$ then,

    · $p_1^{cur} = (\lambda_1 - \lambda_2)\Delta t + p_1^{prev}$

    · $J_1 = \frac{\lambda_1-\lambda_2}{2}(\Delta t)^2 + (p_1^{prev})\Delta t$

    if $\Delta t > t_c$ then,

    · $p_1^{cur} = \frac{\lambda_1}{\lambda_2} + \left(1 - \frac{\lambda_1}{\lambda_2}\right) e^{-\lambda_2(t-t_c)}$

    · $J_1 = \frac{\lambda_1-\lambda_2}{2}(\Delta t)^2 + (p_1^{prev})\Delta t + \frac{\lambda_1}{\lambda_2}(\Delta t - t_c) + \left(\frac{\lambda_1}{\lambda_2^2} - \frac{1}{\lambda_2}\right)(e^{\lambda_2(t_c-\Delta t)} - 1)$

  else (i.e, if p_e is not equal to 1)

  * $p_1^{cur} = \lambda_2\Delta t - p_1^{prev}$

  * $J_1 = \frac{\lambda_2}{2}(\Delta t)^2 - p_1^{prev}\Delta t$

Similarly, if $p_7 = 1$ then,

- $p_2^{cur} = (\lambda_4 - \lambda_3)\Delta t + p_2^{prev}$

- $J_2 = \frac{\lambda_4-\lambda_3}{2}(\Delta t)^2 + (p_2^{prev})\Delta t$

  if $p_e = 1$ then,

  * compute $t_c = \frac{p_1^{prev}-1}{\lambda_1}$

    if $\Delta t \leq t_c$ then,

    · $p_1^{cur} = (\lambda_1)\Delta t + p_1^{prev}$

    · $J_1 = \frac{\lambda_1}{2}(\Delta t)^2 + (p_1^{prev})\Delta t$

    if $\Delta t > t_c$ then,

    · $p_1^{cur} = \frac{1}{\lambda_1} + \left(1 - \frac{1}{\lambda_1}\right) e^{-\lambda_1(t-t_c)}$

    · $J_1 = \frac{\lambda_1}{2}(\Delta t)^2 + (p_1^{prev})\Delta t + \frac{1}{\lambda_1}(\Delta t - t_c) + \left(\frac{1}{\lambda_1^2} - \frac{1}{\lambda_1}\right)(e^{\lambda_1(t_c-\Delta t)} - 1)$

  And finally, if p_6 or $p_8 = 1$ then

  * $p_2^{cur} = (\lambda_3)\Delta t + p_2^{prev}$

        * $J_2 = \frac{\lambda_3}{2}(\Delta t)^2 + (p_2^{prev})\Delta t$
          if $p_e = 1$ then,
            · compute $t_c = \frac{p_1^{prev} - 1}{\lambda_1}$
              if $\Delta t \leq t_c$ then,
              $p_1^{cur} = (\lambda_1)\Delta t + p_1^{prev}$ and
              $J_1 = \frac{\lambda_1}{2}(\Delta t)^2 + (p_1^{prev})\Delta t$
              if $\Delta t > t_c$ then,
              $p_1^{cur} = \frac{1}{\lambda_1} + \left(1 - \frac{1}{\lambda_1}\right) e^{-\lambda_1(t-t_c)}$ and
              $J_1 = \frac{\lambda_1}{2}(\Delta t)^2 + (p_1^{prev})\Delta t + \frac{1}{\lambda_1}(\Delta t - t_c) + \left(\frac{1}{\lambda_1^2} - \frac{1}{\lambda_1}\right)(e^{\lambda_1(t_c - \Delta t)} - 1)$

    – Update cost function as $J_{ac} = J_{ac} + J_1 + J_2$. And reset temporary cost functions to zero. $J_1 = J_2 = 0$.
    – Update current time as $t = t + \Delta t$.
    – Mark the transition which fired as not an enabled transition. And make the downstream discrete transition as an enabled transition. So if enabled transition = 5 then, set $p_5 = 0$ and $p_6 = 1$. Do the same for p_6, p_7, p_8, p_e, and p_{ne}.

- End loop.
- Evaluate the final cost. $J = \frac{J_{ac}}{T}$.

The above process captures the vehicle movement in a more realistic manner. Hence the algorithm will have a better value for cost function. Below example illustrates the working of algorithm. If the delays of discrete transitions are $\theta_6 = 5,\ \theta_8 = 5,\ \theta_9 = 10,\ \theta_{10} = 30$ (note that delays associated with transition t_5 and t_7 are not mentioned here. Because those are the parameters which we are trying to optimize. And they will be randomly chosen by the algorithm). And if the rates of the discrete transitions are $\lambda_1 = 1,\ \lambda_2 = 3,\ \lambda_3 = 1,\ \lambda_4 = 3$. And if, previous markings of p_1, p_2 are 10 and 20 respectively. Let the initial condition be $p_5 = 1$ and $p_e = 1$ (t_5 and t_9 are enabled).

Whenever t_5 is enabled, algorithm choses a random value as the delay for t_5. For instance, let the random delay for t_5 be 17. Since two transitions are enabled at a time, the transition with minimum delay will fire first. Delay associated with $t_9 = 10$. So $\min[17, 10] = 10$. Hence t_9 will fire first. And current discrete state will have a duration of $\Delta t = 10$.

At t= 0. $p_5 = 1,\ p_e = 1$. So,

$$p_2 = (1 \times 10) + 20 = 30$$
$$J_2 = \left(\frac{1}{2} \times 100\right) + (20 \times 100) = 250$$
$$T_c = \frac{(1-10)}{(1-3)} = 4.5$$

Since $\Delta t > T_c$

$$\begin{aligned} J_1 &= \left(\frac{1-3}{2}\right) \times 10^2 + (10 \times 10) + \frac{1}{3}(10 - 4.5) + \left(\frac{1}{9} - \frac{1}{3}\right)(e^{3(4.5-10)} - 1) \\ &= -100 + 100 + 1.8333 + 0.2222 = 2.0555 \end{aligned}$$

After 10 time units, t_9 will fire. Token from p_e will be moved to p_{ne}. Hence p_e won't be enabled any more. Transition t_5 have been active for 10 time units now. So the remaining delay for $t_5 = 17 - 10 = 7$. Now transitions t_5 and t_{10} are active at the same time. Delay associated with both transitions (7 and 30 for t_5 and t_{10} respectively) are compared. Transition with minimum delay will fire first. So, t_5 will fire before t_{10}.

At t= 10, $p_5 = 1,\ p_{ne} = 1$. Transition t_5 will fire first. Hence duration of the current state will be $\Delta t = 7$.

$$p_2 = (1 \times 7) + 20 = 27$$
$$J_2 = \left(\frac{1}{2}\right) 7^2 + (20 \times 7) = 164.5$$

Since p_e has no token during this time, t_1 is not enabled. Hence no change will happen to the p_1 or J_1. Seven time units after, t_5 will fire (one token from p_5 will be moved to p_6). So now, the enabled transitions are t_{10} and t_6. But t_{10} have been active for seven time units now. The remaining delay for $t_{10} = 30 - 7 = 23$ and $\theta_6 = 5$. So

$\Delta t = \min[23, 5]$. At $t = 17,\ p_6 = 1,\ p_{ne} = 1$. Since transition t_6 has the minimum delay it will fire first. Hence duration of current discrete state will be $\Delta t = 5$.

$$p_2 = (1 \times 5) + 20 = 25$$
$$J_2 = \left(\frac{1}{2}\right) 5^2 + (20 \times 5) = 112.5$$

There will not be any change as far as p_1 and J_1 are concerned (t_1 is not enabled). At the end of $t = [17 + 5] = 22$ time units, t_6 will fire, transferring one token from p_6 to p_7. So transitions t_7 and t_{10} will become enabled. And as of now, t_{10} have been active for $7 + 5 = 12$ time units. So the remaining delay of t_{10} is $30 - 12 = 18$ time units. Delay associated with t_7 will be randomly generated by the system. In this case let it be 37. $\text{Min}[\theta_7, \theta_{10}] = \min[37, 18] = 18$. Remaining time at the current discrete state is $\Delta t = 18$.
At $t = 22,\ p_7 = 1,\ p_{ne} = 1$. Duration of current state is $\Delta t = 18$.

$$p_2 = (3 - 1)18 + 20 = 56$$
$$J_2 = \frac{(3 - 1)}{2} 18^2 + (20 \times 18) = 684$$

At the end of $t = 22 + 18 = 40$ time units, t_{10} will fire. During this process, one token will be transferred from p_{ne} to p_e. Thus only t_9 and t_7 will be enabled. But t_7 have been active for 18 time units. Remaining delay of t_7 is $37 - 18 = 19$ time units. So duration of current discrete state will be $\min[19, 10] = 10$.
At $t = 40,\ p_7 = 1,\ p_e = 1$. And $\Delta t = 10$.

$$p_2 = (3 - 1)10 + 20 = 40$$
$$J_2 = \frac{(3 - 1)}{2} 10^2 + (20 \times 10) = 300$$
$$T_c = \frac{(10 - 1)}{1} = 9$$

Since $\Delta t > T_c$

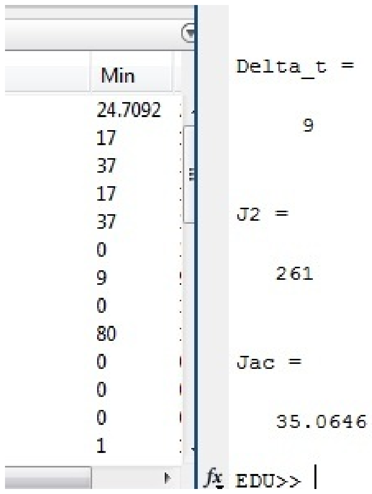
$$J_1 = \left(\frac{1}{2}\right) 10^2 + (10 \times 10) + \left(\frac{1}{1}\right)(10 - 9) + \left(\frac{1}{1} - \frac{1}{1}\right)(e^{1(9-10)} - 1) = 151$$

At the end of $t = 40 + 10 = 50$ time units, t_9 will fire. Hence one token will be moved from p_e to p_{ne}. So t_9 will no longer be enabled. But t_7 and t_{10} will become enabled. At this time t_7 have been active for $[18 + 10] = 28$ time units. So the remaining delay of t_7 is $37 - 28 = 9$ time units. And duration of the current state will be $\min[9, 30] = 9$. At $t = 50,\ p_7 = 1,\ p_{ne} = 1$ and $\Delta t = 9$.

$$p_2 = (3 - 1)9{+}20 = 38$$

$$J_2 = \frac{(3 - 1)}{2}9^2{+}(20 \times 9) = 261$$

And the cycle will continue till time (t) reaches cycle time (T). During the process, computed cost function values are summed up. And the last step of the algorithm is to compute the final cost value. $J = \frac{1}{T} \times$ sum of cost function value incurred during each discrete state. If the above algorithm was executed for a cycle time (T) of 55



```
Min
24.7092
17
37
17
37
0
9
0
80
0
0
0
1
```

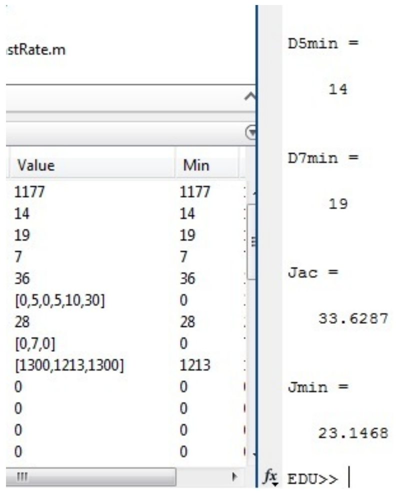
```
Delta_t =

     9


J2 =

   261


Jac =

   35.0646

EDU>>
```

Fig. 4.8. Program result of the illustrated example.

time units (and for the delay, firing rate values mentioned in the example), then cost function value would be,

$$J = 35.001009$$

Cost function value generated by a program (based on the above algorithm) is shown in Fig. 4.8. The program was executed for the same parameters (as used in the example). But the delay values of 17 and 37 for t_5, t_7 respectively, does not yield minimum value for cost function. Those values were chosen for illustrative purposes. Since the working of algorithm have been established, we can go ahead and execute it to find out optimal duration of traffic lights. As mentioned above, the algorithm will



```
stRate.m

Value               Min
1177                1177
14                  14
19                  19
7                   7
36                  36
[0,5,0,5,10,30]     0
28                  28
[0,7,0]             0
[1300,1213,1300]    1213
0                   0
0                   0
0                   0
0                   0
```

```
D5min =

    14


D7min =

    19


Jac =

   33.6287


Jmin =

   23.1468

EDU>>
```

Fig. 4.9. Optimization of two one-way intersection with exponential arrival rate.

select random duration values for t_5, t_7 and will compare the cost function value for each discrete state. Duration (θ_5, θ_7) with least cost function value will be the optimal signal duration value. Fig. 4.9 shows the result when algorithm was executed to find the optimum signal duration. And the duration of traffic signal which resulted in minimum cost function value of 23.1468 is 14 and 19 for t_5 and t_7 respectively. Table compares the result of both the algorithms. Comparison results clearly establishes

Table 4.1.
Optimization results of models (a).

Arrival/departure rate	Cost function value	Signal duration
Constant rate	73.5714	20,40
Exponential rate	23.1468	14,19

the fact that, model in Fig. 4.7 has much better result than model in Fig. 4.1. Cost function value and duration of signal is better for the model with exponential arrival rate. Hence implementation of the model would save valuable time.

5. HYBRID PETRI NET MODEL OF TWO INTERSECTIONS

Till now we were discussing optimization of a traffic signal at an intersection without implementing any new tools. Optimization of the model was solely based on mathematical deductions and calculations. The model with exponential arrival and departure rate improved the cost function value considerably. But work was mainly concentrated on just one intersection. In an effort to expand the model and manage the traffic of a wider area, another model has been designed. This chapter explains the working of new model in detail.

5.1 Model of two connected intersections

The new model is bigger in the sense that it has two separate intersections connected through a road/link. This way the new model is capable of controlling traffic of two different intersections. The first intersection where it controls the traffic is called upstream intersection, while the second intersection is called downstream intersection. Data of upstream intersection is used to optimize the signal duration of the new model. Fig. 5.1 shows the new Petri net model of the connected intersection. For the purpose of keeping it simple, model depicts one way streets. The upstream intersection is very much similar to model shown in Fig. 4.1. So the arrival and departure of vehicles at this intersection occurs at a constant rate. Places p_1, p_2, p_3, p_4 represent the traffic signal of upstream intersection. And the vehicle flow at this intersection is represented using nodes $p_{12}, p_{14}, t_{11}, t_{12}, t_{13}, t_{14}$. Downstream intersection is also modeled in a similar manner. Nodes p_7, p_8, p_9, p_{10} represent the traffic signal while nodes $p_{17}, p_{19}, t_{15}, t_{16}, t_{17}, t_{18}$ represent vehicle flow. Since vehicle flow is a continuous event, continuous node are used to model the event. On the other hand, discrete nodes

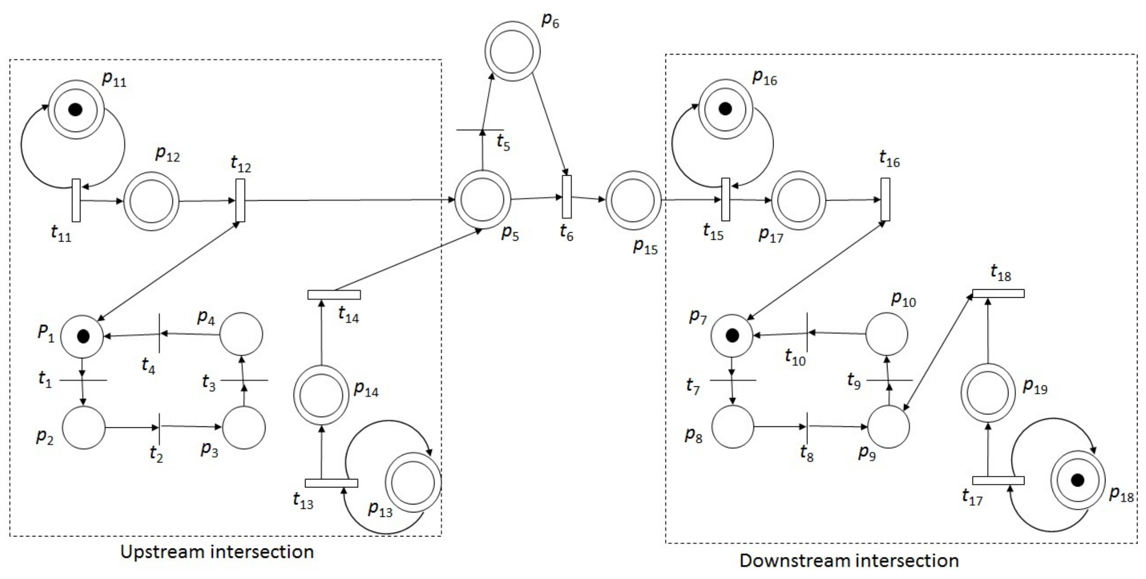


Fig. 5.1. Petri net model of two intersections connected through a link.

are used to model traffic signal because signal change happens at specific intervals of time. Upstream intersection is connected to downstream intersection through a road. All vehicles leaving upstream intersection will pass through this road to reach downstream intersection. But it serves a very important role in the model. This road introduces a delay. And it uses a combination of discrete and continuous nodes to accomplish the task.

Vehicles (tokens) leaving upstream intersection will reach place p_5 through transitions t_{12} and t_{14}. When vehicles reach p_5 transition t_5 becomes enabled. But there is a delay associated with the transition. If delay of t_5 is θ_5, then vehicles from upstream intersection will accumulate at p_5 for a period of θ_5 time units. After that, a token will be released to p_6. This makes t_6 enabled, thus allowing the movement of tokens (vehicles) from p_5 to p_{15}. From there all vehicles will reach downstream intersection, p_{17}, through t_{15} like a burst. So vehicles will depart from upstream intersection at constant rate. But the link (consisting of nodes p_5, p_6, t_5, t_6) will alter the process so that vehicles will arrive at downstream intersection in form of a burst.

Departure of vehicles from upstream intersection happens at constant rate. So, the switching delays of upstream intersection will be constant. But arrival of vehicles at downstream intersection happens in bursts. So, switching delays of the downstream intersection could be optimized using the same algorithm discussed in previous chapter. Since upstream intersection is much similar to model shown in Fig. 4.1, we can use algorithm for constant arrival rate model to calculate the parameters of the intersection. These parameters can be used as input to downstream intersection while optimizing the signal duration of the downstream intersection. Following steps explains how the process is carried out.

Begin algorithm

1. Write a function which will calculate the parameters of model shown in Fig. 4.1. Since vehicles arrive and depart at constant rate in this model, the traffic signal will have fixed delay. So, this function will calculate number of vehicles leaving the intersection based on fixed delay and constant arrival rate. This function will have no input arguments. But it will have two output arguments. For example: ConstantModel();

2. Write a function which will calculate the optimum switching delay based on the model shown in Fig. 4.7.
Since vehicles arrive as bursts in this model, the traffic signal duration will be optimized based on the input parameters. So this function will have two input arguments and two output arguments. For example: ExpModel(p_{12}, p_{14});

3. Start main program.

4. Call function to calculate the parameters of model shown in Fig. 4.1. For example: $[p_{12}, p_{14}]$ = ConstantModel();
Duration of traffic light is constant, so number of vehicles leaving the intersection could be calculated by mathematical deductions discussed in the constant arrival rate algorithm. That will be the output of this function.

5. Call function to calculate the optimum switching delay, based on the model shown in Fig. 4.7. For example: $[D_5, D_7]$ = ExpModel(p_{12}, p_{14});

This function will determine the optimum switching delay based on the input parameters determined by previous function for a fixed time horizon T.

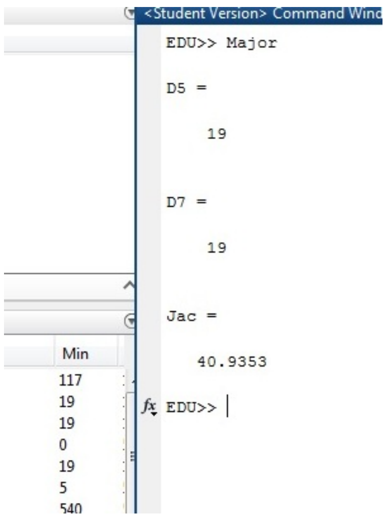
6. Apply the optimum delay to the system.

7. Calculate the parameters (markings, cost function etc.) of the downstream intersection based on the switching delay and exponential arrival rate.

8. After certain time, T_{upd} (lower than T), go to step 4.

End algorithm

So the process will estimate the parameters based on model in Fig. 4.1 (constant arrival rate model). And then it will optimize the signal duration based on exponential arrival rate model (Fig. 4.7). Because of the use of exponential arrival rate algorithm to optimize the signal, the system will have much better cost function and signal duration. And the process will keep updating the signal duration very often. Thus based on the data from upstream intersection, it will optimize the signal duration of downstream intersection. In other words, based on data from one point, the algorithm will predict the behavior of the system elsewhere. And the optimization of neighboring intersection does not require any added tools. Optimization is achieved through mathematical calculations. This makes the process cost effective as well. Fig. 5.2



```
EDU>> Major

D5 =

    19


D7 =

    19


Jac =

   40.9353

EDU>>
```

Fig. 5.2. Cost function value of two intersections connected through a link.

shows the cost function obtained by following the above method. The result shows an improvement in cost function value and a much better value for signal duration.

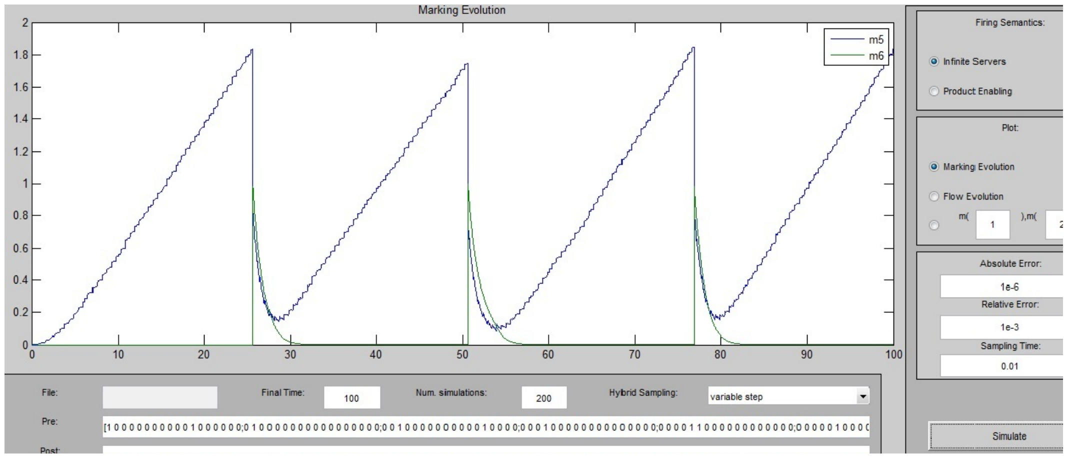
Table 5.1 compares the result obtained for the connected-intersection model, one-intersection model with exponential arrival/departure rate and, one-intersection model with constant arrival/departure rate for the same initial conditions. Since the algorithm used to optimize the signal of connected-intersection model is same as the one discussed for exponential arrival rate model, the resulting signal duration will be comparable to the signal duration of exponential arrival rate model. And for the same reason, cost function values of exponential arrival/departure rate model and connected-intersection model will be significantly less than constant arrival/departure rate model.

Table 5.1.
Optimization results of models (b).

Arrival/departure rate	Cost function value	Signal duration
Constant rate	73.5714	20,40
Exponential rate	23.1468	14,19
Exponential rate for connected-intersections	40.9353	19,19

5.2 Simulation results

In order to better understand the working of connected intersection model, simulations of the model were carried out. This section explain the simulation results. However, simulations of upstream intersections are not shown since they are already shown in previous chapter.



Firing Semantics: [✓ Infinite Servers] Product Enabling

Plot: [✓ Marking Evolution] Flow Evolution m(1),m(2

Absolute Error: 1e-6

Relative Error: 1e-3

Sampling Time: 0.01

File:

Final Time: 100

Num. simulations: 200

Hybrid Sampling: variable step

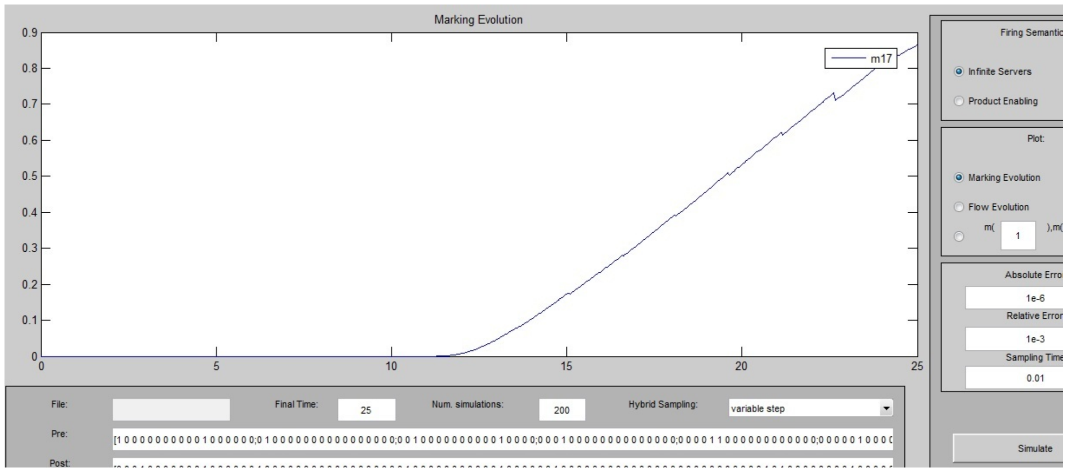
Pre: [1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0;0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0;0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 1 0 0 0 (

Post:

Fig. 5.3. Result showing working of link.

Since p_5 is fed by continuous transitions, marking of p_5 will increase continuously. This is shown by the steep curve in the graph. When the marking reaches one, t_5 will become enabled. But it will wait for delay time (associated with t_5) to elapse. Once it has elapsed, one token will be transferred to p_6. Since p_5 is a continuous transitions, it will have fractional amount of token left in it (even after the firing of t_5). Now, p_6 has one token. And p_5 also has fractional amount of token. This will enable t_6. Depending on the firing rate of t_6, tokens will be removed from p_5 and p_6. Since t_6 is a continuous transition, number of tokens will decrease continually at p_5 and p_6. When t_6 finishes transferring all token of p_6 to p_{15}, t_6 won't be active any more. So tokens from p_5 won't be transfered to p_{15}. Hence tokens inside p_5 will start accumulating. And the cycle will continue.

The Fig. 5.4 shows arrival of vehicles at downstream intersection (p_{17}). Vehicles leave upstream intersection at constant rate. But link/road connecting it to downstream intersection will convert the constant arrival rate to exponential arrival rate. So the arrival at downstream intersection happens in burst. This is supported by the graph. When vehicles starts arriving at downstream intersection, the graph will start to rise exponentially. After it reaches a maximum value, it will start to decrease gradually (indicating that almost all vehicles from upstream have reached downstream and



Firing Semantic

[✓ Infinite Servers] Product Enabling

Plot:

[✓ Marking Evolution] Flow Evolution

m(1),m(

Absolute Erro: 1e-6

Relative Error: 1e-3

Sampling Time: 0.01

File:

Final Time: 25

Num. simulations: 200

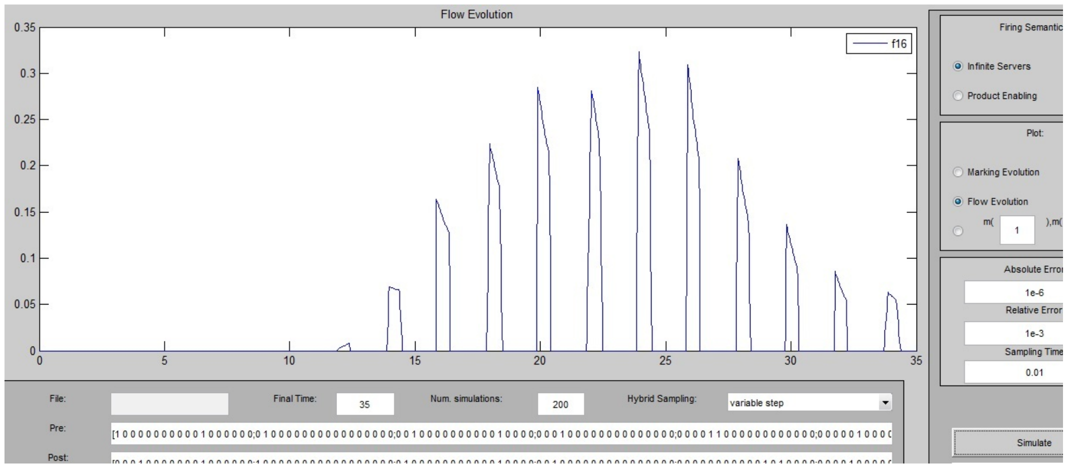
Hybrid Sampling: variable step

Pre: [1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0;0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0;0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 1 0 0 0

Post:

Fig. 5.4. Result showing arrival of vehicles at downstream intersection.

the burst is getting over). So the curve will continue decaying. And when the next burst arrives, the curve will rise exponentially and the cycle will repeat.



Firing Semantic

[✓ Infinite Servers] Product Enabling

Plot:

Marking Evolution [✓ Flow Evolution]

m(1),m(

Absolute Erro: 1e-6

Relative Error: 1e-3

Sampling Time: 0.01

File:

Final Time: 35

Num. simulations: 200

Hybrid Sampling: variable step

Pre: [1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0;0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0;0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 1 0 0 0

Post:

Fig. 5.5. Result showing departure of vehicles at downstream intersection.

Since vehicles arrive at downstream intersection in bursts, the number of vehicles at downstream intersection will rise from zero and it will keep on increasing till it reaches a peak value. Then the number of vehicles will begin to decrease gradually. So, for the same reasons mentioned for the previous graph, the exponential nature of vehicle arrival will be retained at the downstream intersection. Depending upon the presence of token at p_7, t_{16} will become enabled only at specific intervals. Spikes shown in the graph indicate time intervals during which t_{16} was enabled. Vehicles will cross the intersection only during those intervals.

6. CONCLUSION

Modeling and simulation is an important way of knowing system behavior even before putting the system to use. It helps to understand system behavior (under various circumstances) before actually making/using the system. Field of engineering uses modeling and simulation extensively. And Petri net is one such modeling tool. It is a popular tool because of its simplicity (systems modeled using Petri nets are represented graphically) and versatility (it can represent discrete as well as continuous events efficiently).

6.1 Summary

This thesis deals with Petri net models of traffic intersections and optimization of traffic signals. Chapter 1 is an introduction to thesis. Motivation for selecting the research area, work on related topics and preview of what has been done were explained in the chapter. Since majority of the work concentrates on Petri nets, Chapter 2 explains basics of Petri net. Definitions, notations, and working of Petri nets were illustrated using examples. Different types of Petri nets used in the work, and their properties were also discussed. Much of the work in this thesis also deals with optimization of traffic signal. And so, Chapter 3 was an overview about optimization process. It explained various steps involved in optimizing a process. Later on in the thesis, these steps were used to optimize traffic signal.

Chapter 4 started by explaining a two one-way intersections controlled by a traffic signal. Two different Petri net structures were used in this model. First one was a discrete structure (used to model traffic signal) while the second one was a continuous structure (used to model vehicle movement). Simulations of the model were carried out to understand its working. In this model, arrival and departure of vehicle occurred at constant rate. Mathematical description of the system was discussed and a program was developed to evaluate the mathematical function. But there were major flaws to the assumptions made in the model. To rectify the flaws, a new model was discussed. New model replicate arrival and departure of vehicles as occurring in bursts. This was a much more realistic approach. Working of the new model was discussed in detail. Mathematical deduction and programs were developed to evaluate the objective function of the model. The new model was optimized using the program developed. Chapter 5 extends the traffic model by connecting two different intersections via road. And this model was also simulated to better understand its working. Optimization of traffic signal had been carried out by combining the algorithm developed for the previous model.

6.2 Conclusion

Petri net is an efficient modeling tool. The fact that it is versatile and simple makes it a popular choice in traffic system modeling. While a hybrid Petri nets could be used to represent different events happening in the system, a timed hybrid Petri net could be used to induce time delay to various nodes of the model. The idea of vehicle arrival and departure occurring in an exponential manner (rather than at constant rate) helped make the system more real time and efficient. And hence such a model will have much better cycle time (duration of red/green/yellow light). With the implementation of this model, vehicles at the intersection will wait for shorter amount of time. And most importantly, the cycle time is not fixed. It varies from cycle to cycle. And hence wait time of vehicle at the intersection will also vary after each cycle.

Optimization is an important process in engineering. It helps to make sure that a system operates at its maximum possible efficiency under varying circumstances. Optimization of signal duration have made the traffic intersection model more efficient. Optimization process helped to make sure that vehicles at the intersection do not spend too much time waiting to cross the intersection. And further more, the same optimization algorithm have been used to optimize signal duration of a connected downstream intersection. Thus, the optimization algorithm improved the signal duration of one intersection, in addition, it improved the signal duration of a connected intersection without the use of any additional tool.

6.3 Future work

There is much room for expansion of the model shown in Fig. 4.7, Out of the two one-way intersection shown, vehicles arrive and depart at one intersection in an exponential manner while arrival and departure at the second intersection happens at constant rate. Second intersection could be modified to make arrival and departure at that intersection in exponential manner. This could be done by keeping intersection 1 intact while replacing intersection 2 with burst model. This model will be much more closer to real-time scenario. Further more, the model could be expanded by making the one-way streets into two-way street and thus model could represent two two-way intersection. The situation would prompt the use of burst model multiple times. Same could be done for model shown in Fig. 5.1. Model could be expanded to accommodate exponential arrival and departure at both sections of the upstream intersection. This would result in much more accurate data for downstream intersection. And also, one-way streets could be modified to two-way streets. Finally the model in Fig. 5.1, could be modified to connect it to multiple intersections. This would tackle traffic problems of much wider area. The same process used to optimize the model in Fig. 5.1, could be used to optimize each signal.

REFERENCES

REFERENCES

[1] D. I. Robertson, ”The scoot on-line traffic signal optimization technique,” *Traffic Eng. Control*, vol. 23, pp. 190-192, 1982.

[2] D. I. Robertson, ”Transyt method for area traffic control,” *Traffic Eng. Control*, vol. 10, pp. 276-281, 1969.

[3] M. Diaz, ”Petri Nets : Fundamental Models, Verification and Applications,” *Wiley-ISTE*, 2009.

[4] R. David, H. Alla, ”Discrete, Continuous, and Hybrid Petri Nets,” *Springer*, 1st edition., 2005.

[5] R. David, H. Alla, ”On hybrid petri net,” LAST DATE ACCESSED: June 8, 2015 http://webdiis.unizar.es/asignaturas/MSC/hybrid.pdf

[6] R. David, H. Alla, ”Continuous petri nets,” *Proc. 8th European Workshop on Application and Theory of Petri Nets*, 1987.

[7] R. David, H. Alla, ”Autonomous and timed continuous petri nets,” *11th International Conference on Application and Theory of Petri Nets*, pp. 367-386, 1990.

[8] C. R. Vandzquez, H. Y. Sutarto, R. Boel, and M. Silva, ”Hybrid petri net model of a traffic intersection in an urban network,” *2010 IEEE International Conference on Control Applications*, pp. 658-664, 2010.

[9] E. Lefeber, J. E. Rooda, ”Controller design for switched linear systems with setups,” *Physica A*, vol. 363, 4861, 2006.

[10] J. Julves, C. Mahulea, ”SimHPN: A MATLAB Toolbox for Hybrid Petri Nets USER MANUAL,” LAST DATE ACCESSED: June 8, 2015 http://webdiis.unizar.es/GISED/tools/contpn/SimHPNman.pdf vol. 1, 2012.

[11] F. Boillot, S. Midenet, J. Pierrele, ”The real-time urban traffic control system CRONOS: Algorithm and experiments,” *Transportation Research Part C: Emerging Technologies*, vol. 14, pp. 18-38, 2006.

[12] F. Boillot, J.M. Blosseville, J.B. Lesort, V. Motyka, M. Papageorgiou, S. Sellam, ”Optimal signal control of urban traffic networks,” *Road Traffic Monitoring, (IEEE Conf. Pub. 355)*, 1992.

[13] F. Balduzzi, A. Giua, G. Menga, ”First-Order Hybrid Petri Nets: A Model for Optimization and Control,” *IEEE transactions on robotics and automation*, vol. 16, no. 4, 2000.

[14] H. Yang, B. Jiang, V. Cocquempot, P. Shi, ”Fault tolerant control design via hybrid Petri nets,” *Asian Journal of Control*, vol. 12, no. 5, pp. 586-596, 2010.

[15] E.K.P. Chong, S.H. Zak, ”An Introduction To Optimization,” *John Wiley and Sons, Inc*, 4th edition., 2013.

[16] A.N.S. Moh, W. Ait-Cheik-Bihi, M. Wack, ”Modelling and analysis of a non-synchronized transport network using Petri nets and (max, plus) algebra,” *International Conference on Computers and Industrial Engineering*, pp. 6, 2009.

[17] B. Huang, C. Zhao, Y. Sun, ”Modeling of Urban Traffic Systems Based on Fluid Stochastic Petri Nets,” *Fourth International Conference on Natural Computation*, pp. 149 - 153, 2008.

[18] L. Bohang, Y. Xiaoyong, L. Qingbing, H. Shougang, ”An improved method for traffic control relying on close-loop control theory,” *International Asia Conference on Informatics in Control, Automation and Robotics*, pp. 48 - 50, 2010.

[19] A. Di Febbraro, D. Giglio, N. Sacco, ”Urban traffic control structure based on hybrid Petri nets,” *IEEE Transactions on Intelligent Transportation Systems*, pp. 224 - 237, 2004.

[20] J. Julvez, R. Boel, ”Modelling and controlling traffic behaviour with continuous petri nets,” *Proc. 16th IFAC World Congress*, 2005.

[21] T. Murata, ”Petri nets: Properties, analysis and applications,” *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541580, 1989.

[22] A. Di Febbraro, D. Giglio, ”Urban traffic control in modular/switching deterministic-timed petri nets,” *IFAC Symposium on Control in Transportation Systems*, vol. 11, pp. 153 - 158, 2006.