DECISION AND CONTROL IN DISTRIBUTED COOPERATIVE SYSTEMS

by

PRASANNA MOHAN BALLAL

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2008

ACKNOWLEDGEMENTS

In the past three years as a PhD candidate, there have been many people who have helped me choose the right direction and made this dissertation possible. This dissertation is dedicated to them.

I would like to begin by thanking Dr. Frank Lewis for his direction, assistance, guidance and most importantly his friendship throughout my days as a PhD researcher.

I wish to thank the members of my committee, Dr. Harry Stephanou, Dr. Dan Popa, Dr. Qilian Liang and Dr. Kamesh Subbarao, for their support, patience, and good humor. Special thanks should be given to my student colleagues who helped me in many ways, especially Draguna Vrabie and Pritpal Dang. Thanks for being there for me during my difficult times.

This work is dedicated to the greatest teachers of my life, my loving parents Kalanidhi and Mohan Ballal who have taken great pains to see me prosper in life. I thank them for believing in my abilities, and constantly providing the support I needed.

July 3, 2008

ABSTRACT

DECISION AND CONTROL IN DISTRIBUTED COOPERATIVE SYSTEMS

Prasanna Mohan Ballal, PhD.

The University of Texas at Arlington, 2008

Supervising Professor: Frank Lewis

This dissertation presents novel matrix-based approaches for decision and control in distributed cooperative systems such as wireless sensor networks. A novel matrix-based Discrete Event Controller has been implemented for task planning and resource dispatching in a network consisting of stationary ground sensors and mobile agents also known in the literature as mobile wireless sensor networks. The use of shared resources in such systems can sometimes cause a phenomenon called system deadlock, where all the processes in the system come to a standstill. This work presents a new matrix-based algorithm that has been implemented for deadlock avoidance in a system with shared resources and dynamic resource assignment. The analysis of deadlock is based on certain Petri Net objects in such systems called critical siphons and critical subsystems.

The analysis of deadlock avoidance becomes even more difficult when routing of tasks and resources are involved. The critical siphons and critical subsystems have to be redefined. This dissertation presents a new matrix-based approach for deadlock

avoidance in such systems. This is a generalized approach that can be used for systems with or without routing. This work also presents a method for tackling a certain pathological case called second order deadlocks.

In routing systems, dynamic decisions have to be made. In the presence of numerous agents which act as resources, a collective decision can be made based on the individual decisions of agents. This method is called data fusion. Dempster Shafer (DS) theory has been extensively used in the past for data fusion since it provides an excellent framework for conditions involving uncertainty. But the mathematics of computation involving DS belief functions is difficult to fathom because of the many summations over set inclusions and intersections. The equations are often difficult to comprehend and discourage readers due to their complexity, and are often difficult to implement using software. This dissertation provides a new matrix formulation for updating evidence and computing beliefs and plausibilities in DS theory. The work also shows how evidence theory can be used in routing systems for Condition Based Maintenance.

Finally, this work presents a framework for trust propagation and maintenance in a network of nodes or mobile agents that yields global consensus of trust under rich enough communication structure graphs. This work considers the case where the graph structure is a time-varying function of the trusts based on the graph connectivity. This makes the trust consensus scheme bilinear. This trust consensus is incorporated into cooperative control laws that depend on local information from neighboring nodes, yet yield team-wide desired behavior such as flocking and formations.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Distributed Cooperative Systems

Most of the systems and applications in real world are inherently distributed. This research considers distributed systems where autonomous participants pool together their local resources towards a global service. Such distributed systems provide extensibility, protection against failures, and an adequate architecture to cope with spatially distributed real-world applications. These systems are widely used in distributed databases, telecommunication, security, wireless sensor networks [7, 57, 65], and education. The challenge in developing cooperative distributed systems is in modeling and engineering the interactions between the entities of the system.

Modern design of large complex systems requires cooperation among many agents (humans or computers) that may be physically separated and/or operating under diverse environments. Implementing a distributed cooperative system is a huge technical challenge. Cooperative control of distributed multi-agent systems is organized into four main themes of cooperative control: distributed control and computation, adversarial interactions, uncertain management and complexity management.

Cooperation in distributed systems is very important since cooperation is not usually guaranteed by design and the result of non-cooperation could lead to a total system failure. Distributed cooperative systems make existing services better and

cheaper; the only gamble being that autonomy is such systems could imply lack of trust. In this research we consider distributed cooperative control and decision-making in wireless sensor networks as a case study.

1.2 The Problem

There has been increased research interest in systems composed of multiple autonomous agents such as mobile robots and stationary agents such as unattended ground sensors exhibiting distributed cooperative behavior. Emphasis has been put on developing wide area distributed wireless sensor networks with self-organization capabilities to cope with sensor failures, changing environmental conditions, and different environmental sensing applications. In particular, mobile wireless sensor networks hold out the hope to support self-configuration mechanisms, guaranteeing adaptability, scalability and optimal performance, since the best network configuration is usually time-varying and context dependent. This work provides solutions for task assignment and resource dispatching (i.e. supervisory control), deadlock avoidance, decision-making and trust establishment in mobile wireless sensor networks.

Different techniques are available to coordinate the task assignment and resource dispatching in mobile wireless sensor networks such as decentralized and centralized techniques. In decentralized coordination, the robots and sensors only have information about their local neighbors and do not have complete information of the entire network, whereas, in centralized technique, a supervisor controls the coordination of the robots and sensors. In the decentralized approaches, robots possess similar functionalities, perform similar tasks and just one mission at a time is usually

implemented. To overcome the inherent limitations of decentralized approaches, supervisory (centralized) control techniques are preferred. Some significant results in supervisory control have also been obtained using Petri Nets (PN). Nevertheless the implementation of high-level mission specifications is not straightforward, the dynamical description of the system is incomplete and a new design stage, almost from scratch, is required if objectives or resources change. Thus, there is a lack of supervisory control and decision-making techniques, which can sequence different missions according to the scenario (adaptability) and reformulate the mission if some of the robots fail (fault tolerance) in a predictable way and using a high-level interface.

Discrete Event Controller (DEC) was first used in manufacturing systems [30, 32, 47, 50] in order to sequence the most suitable tasks for each agent according to the current perception of the environment. A novel matrix formulation makes the assignment of the mission planning straightforward and easily adaptable if agents or applications change. It represents a complete dynamical description of the system that allows computer simulation analysis. The matrices are direct to write down given the sequence rules for a given task. Priority rules for efficiently dispatching shared resources and handling simultaneous missions can also be easily taken into account. Note that this controller is centralized but there could be many sensors in the network which have same functionalities. Selection of an ideal sensor to do the job can be done in a distributed fashion. This leads to the problem of decision-making. This dissertation is mainly concerned with the decision-making and control which involves combining decisions coming from several experts.

Decision-making becomes difficult when uncertainty is involved. Mobile wireless sensor networks present a range of challenges as they are closely coupled to the physical world with all its unpredictable variation, noise, and asynchrony; they involve many energy-constrained, resource-limited devices operating in concert; they must be largely self-organizing, self-maintaining and robust despite significant noise, loss, and failure.

In the case of shared resources, problems of deadlock can occur. Deadlock is a situation when all the tasks in a network come to a halt. Also, analysis of deadlock becomes difficult when routing of these shared resources is allowed. Routing of tasks and resources has to be done carefully, which again involves planning and decision-making.

Another issue in such systems is the establishment of trust among the nodes. In a distributed network, trust is interpreted as a set of relations among the nodes participating in the network activities. Trust establishment in distributed communication networks such as mobile ad hoc networks (MANETs), sensor networks and ubiquitous computing systems is considered to be more difficult than in traditional hierarchical structures such as the Internet and Wireless LANs centered on base-stations and access points. Given the presence of enemy components and the possibility of node compromise, a *trust consensus* must be reached by the network that determines which nodes to trust, which to disregard, and which to avoid. Trust algorithms for network nodes must be autonomous computationally efficient numerical schemes. However, existing schemes for control of dynamical systems on communications graphs (in the

style of work by [3, 23, 37, 52, 59, 60, 64]) do not take into account trust propagation and maintenance (such as work by [38, 86]). Yet it is a fact that biological groups such as flocks, swarms, herds, do have built-in trust mechanisms to identify team members, team leaders, and enemies to be treated as obstacles or avoided. Cooperative mission planning should involve decisions made in the context of the trust opinions of all nodes, and be based on performance criteria set by network team leaders. These performance criteria may change with time depending on varying mission objectives in the field.

1.3 Objective

The main objective of this research is to develop a matrix-based approach for decision and control in distributed cooperative systems with a case study on mobile wireless sensor networks. First, we develop a supervisory control for task planning and resource assignment using the DEC. Chapter 2 provides a brief introduction to Petri Nets (PN) and the DEC and shows how this matrix-based DEC can be used in distributed cooperative control. One can guarantee a smooth flow of logical operation in DE systems, including dynamic task sequencing and shared resource assignment, without blocking phenomena, including deadlock and others.

Chapter 3 presents a novel approach to implement on-line deadlock-free resource assignment for multi-agent systems with multiple missions. At each event occurrence, when resource changes are required, a greedy algorithm is first implemented by on-line updating of the resource requirements matrix [47]. The new resource assignment is accepted if it is compatible with a certain Maximum Work in Progress (MAXWIP) deadlock avoidance policy. Specifically, the discrete event system

representing the new mission plan has to satisfy two conditions: After the implementation of the new resource assignment, it is necessary to guarantee that (1) the system is not already in deadlock and (2) the new system is regular in a sense described in chapter 3. In order to check the latter condition the regularity test proposed in [32] is launched every time a new resource assignment is proposed. Having produced an allowable assignment of resources, the DEC described in chapter 2 is run to assign the next tasks in the multiple missions based on priority assignment policies.

Chapter 4 describes the deadlock avoidance scheme used in systems where routing decisions have to be made. The analysis of shared resources becomes even harder when choices are allowed for tasks. This means routing decisions have to be made [13]. These systems are called the Free Choice Multi-Reentrant Flow Lines (FMRF). FMRF systems are also known as S3PR (System of Simple Sequential Processes with Resources). In the case of FMRF, the known methods of deadlock avoidance provided for MRF do not work. There are many Deadlock Prevention (DP) methods for S3PR systems in the literature, but this work is mainly concerned with Deadlock Avoidance policies (online deadlock control). Certain objects such as Circular Waits, Critical Siphons, and Critical Subsystems have to be defined to avoid deadlock in FMRF [47, 50]. Chapter 4 describes how these objects can be computed for FMRF. Also, the chapter extends the matrix formulation in chapter 3 which can efficiently compute these objects for FMRF. This matrix formulation allows fast and efficient numerical computation techniques to be applied to Petri Net analysis. We show how a MAXWIP dispatching policy can be formulated for FMRF to avoid blocking

phenomena. Under this policy, deadlock in FMRF can be avoided by limiting the work in progress (WIP) in the Critical Subsystems of each CW. There is another type of deadlock called the second order deadlock [32, 50]. Chapter 4 provides a regularity test to find key resources in the FMRF systems that cause second order deadlocks.

Chapter 5 deals with decision-making involved in systems where routing is necessary. It provides a novel method for job and resource dispatching for condition-based maintenance using Free-Choice Petri Nets and Dempster-Shafer Evidence theory. In case of unreliable sensors in the system, there is uncertainty in decision-making. This uncertainty is modeled using Dempster Shafer Theory of combination (DS), which is used to decide which path to take in the event of choice decisions in the PN. The computations required in DS theory are difficult to perform due to the many required summations over set inclusions and intersections. Therefore, a new matrix formulation is given herein for efficiently combining evidence and finding belief, plausibility, and other quantities in Dempster Shafer theory. This, coupled with a previously developed matrix formulation for PN, makes it direct to implement the decision framework as a DEC using computer software. Examples are given showing how to apply the matrix formulation for decision-making in intelligent diagnostics.

Chapter 6 presents a framework for trust propagation and maintenance in a network of nodes or mobile agents that yields global consensus of trust under rich enough communication structure graphs. Most of the work in literature considers the graph to be static or have static weights. This chapter considers the case where the graph structure is a time-varying function of the trusts based on the graph connectivity.

This makes the trust consensus scheme bilinear. This trust consensus is incorporated into cooperative control laws that depend on local information from neighboring nodes, yet yield team-wide desired behavior such as flocking. Chapter 7 concludes the work with ideas for future research.

CHAPTER 2

MATRIX-BASED DISCRETE EVENT CONTROLLER

2.1 Introduction

The integration and cooperation of heterogeneous agents in a distributed system such as mobile wireless sensor networks is potentially very appealing, but requires a coordination supervisory controller suitable to sequence different missions according to the events and to the functionalities of the agents. In supervisory control, some significant results have been obtained using Petri Nets [40, 41], but the implementation of high-level mission specifications is not straightforward, the dynamical description of the system is incomplete and a new design stage, almost from scratch, is required if objectives or resources change.

In [32, 47] a matrix-based DEC has been proposed, proving to be very efficient in sequencing tasks in manufacturing environments [50]. The matrix formulation allows fast, direct design and reconfiguration of discrete event controllers. It provides a better dynamical description and a higher level interface than other popular tools for discrete event systems, such as Petri Nets. This chapter proposes the use of the matrix-based DEC as a central planner to produce high-level missions for a distributed network of stationary sensing units and mobile robots cooperating for a common goal. Its formulation allows one to switch between missions as priorities change or exception situations occur, and to accommodate node failures. Before moving on to the matrix-

based DEC, it is important to understand the concept of Petri Nets, Circular Waits, Marking Vector, Circular Blocking, Critical Subsystems and Siphons.

2.2 Petri Nets

A Petri Net (PN) a bipartite digraph (*P, T, I, O*), where *P* is the set of places, *T* is a set of transitions, *I* is the set of input arcs from places to transitions, and *O* is the set of outputs arcs from transitions to places [55]. One can represent *I* as an input incidence matrix which has $I(i,j) = 1$ if there is an input arc from place *j* to transition *i* and *O* as an output incidence matrix which has $O(i,j) = 1$ if there is an output arc from transition *i* to place *j*. The incidence matrix is defined as

$$W = O - I \tag{2.1}$$

Given a node *v* (either transition or place), we define $\bullet v$ as the pre-set of *v* (set of nodes with arcs to *v*) and $v \bullet$ as the post-set of *v* (set of nodes with arcs from *v*). Similarly, for a set of nodes $S = \{v_i\}$, define $\bullet S = \{\bullet v_i\}$ and $S \bullet = \{v_i \bullet\}$.

The following assumptions allow one to represent a discrete event system by a Petri Net:

1. There are no machine failures.
2. No pre-emption. A resource cannot be removed from a job until it is complete.
3. Mutual exclusion. A single resource can be used for only one job at a time.
4. Hold while waiting. A process holds the resources already allocated to it until it has all the resources required to perform a job.

2.3 Circular Waits

The following background is taken from [47, 48, 50]. We say resource r_i waits for resource r_j (denoted $r_i \rightarrow r_j$) if the availability of r_j is an immediate prerequisite for the release of r_i, i.e., $\bullet r_i \cap r_j \bullet \neq \phi$. A wait relation digraph is defined as $G_w = (R, A)$ where R is the set of nodes and $A = \{a_{ij}\}$ is the set of edges with a_{ij} drawn if $r_i \rightarrow r_j$ (i.e. each a_{ij} represents a transition in $\bullet r_i \cap r_j \bullet$). In G_w, define an R-path between r_i and r_k as a set of R-places such that $r_i \rightarrow r_j \rightarrow \ldots \rightarrow r_k$. Then r_i is said to wait over an R-path for r_k, denoted $r_i \mapsto r_k$, if there is an R-path between r_i and r_k. A circular wait (CW) is a set of resources $C \subset R$, with |C|>1, such that for any ordered pair $\{r_i, r_j\} \subset C, r_i \mapsto r_j$. A CW always contains at least one shared resource.

The simplest CW is a set of resources $C \subset R$, such that for some appropriate re-labeling, one has $r_1 \rightarrow r_2 \rightarrow \ldots \rightarrow r_q \rightarrow r_1$, with $r_i \neq r_j$ for $i \neq j, 1 \leq i, j \leq q$. This will be referred to as a simple circular wait. A simple circular wait is a simple circuit in the graph and is a CW not containing any other CW. Consider a wait relation graph G_w and a CW $C \subset G_w$. Then for every $r \in C$, there exists at least one simple circular wait $\sigma \subset C$ such that $r \in \sigma$. A CW is a strongly connected sub graph in the digraph G_w and can be obtained by taking unions of non-disjoint simple CW.

Given a CW $C = \{r_i\}$, one can partition the set of transitions $\bullet r_i$ as $\bullet r_i = \bullet r_{io} \cup \bullet r_{i+}$, where $\bullet r_{io} = \{x \in T \mid \bullet x \cap C \neq \phi\}$, the set of input transitions of r_i with input arcs from some other $r_j \in C$, and $\bullet r_{i+} = \{x \in T \mid \bullet x \cap C = \phi\}$, the set of input

transitions of r_i with no input arcs from any other $r_j \in C$. We loosely say that the transitions $\bullet r_{io}$ are 'in the CW C'.

The job set of CW $C = \{r_i\}$ is given by $J(C) = \bigcup_{i=1}^{n} J(r_i)$. Partition this as $J(C) = J(C)_+ \cup J(C)_0$, where $J(C)_0 = \{p \in J(C) \mid p\bullet \in \bullet r_{io}, r_i \in C\}$ and $J(C)_+ = \{p \in J(C) \mid p\bullet \in \bullet r_{i+}, r_i \in C\}$.

2.4 Marking Place Vector

A place $p \in P = J \cup R \cup PI \cup PO$ is said to be marked when it contains a token, which depending on the place containing it indicates an ongoing job, the existence of an available resource, a part in, or a product out. In a FMRF, the initial marking vector denoted as m_0 assigns tokens only to *R* and *PI*-places. It is assumed that the *PO*-places are always empty.

Given $p \in P$, $m(p)$ denotes the marking of *p*, i.e. the number of tokens in *p*. Given a set of places *S*, $m(S)$ denotes the number of tokens in *S*. A set of places is said to be unmarked or empty if none of its places has any tokens.

Given the set of places *P*, the PN place vector, or p-vector, $\vec{p}$ has dimension of $|P|$, and one element corresponding to each place. Let the set of all places be $P = \{p_1, p_2, ..., p_Q\}$. Then the place vector has *Q* elements. Any set of *m* places $\{p_{i_k} \in P \mid k = 1,2,...,m\}$ can be represented as a |*P*|-vector $\vec{p}$ having *m* entries $\vec{p}_{i_k} = 1$

and zero entries otherwise. The PN marking vector $m(\vec{p}) = [m(p_1) \quad m(p_2) \cdots]^T \in N^{|P|}$, with the natural numbers *N*= {*0, 1, 2...*}, gives the number of tokens in each place.

Define the PN transition vector, or *x*-vector, *x* to have dimension of $|T|$, and one element x_i corresponding to each transition. Let the set of all transitions be $T = \{x_1, x_2, ...x_L\}$. Then the transition vector has *L* elements. A set of *m* transitions $\{x_{i_k} \in T \mid k = 1,2,...,m\}$ can be represented as an *L*-vector *x* having *m* entries $x_{i_k} = 1$ and zero entries otherwise.

The well-known PN marking transition equation [55],

$$m^+(\vec{p}) = m(\vec{p}) + W^T x \tag{2.2}$$

gives the new marking vector $m^+(\vec{p})$ in terms of the previous marking vector $m(\vec{p})$ and the transitions that have fired appearing as *1*'s in *x*.

A *p*-invariant is defined as a set of resources and places that is in the null space of *W*. i.e.

$$Wp=0. \tag{2.3}$$

Note that if *p* is a *p*-invariant, then $p^T m^+(\vec{p}) = p^T m(\vec{p}) + p^T W^T x = p^T m(\vec{p})$ so that the number of tokens in a *p*-invariant is constant. One type of *p*-invariant is given by any resource plus all of its jobs, $r \cup J(r)$.

2.5 Circular Blocking

A circular blocking CB is a circular wait that is empty and will always remain so [32, 47, 50, 92]. That is, for a CW *C*= {r_i}:

1. $m(C)=0$, and
2. no tokens will ever be added to C.

In this situation, one is said to have deadlock, where the resources in the CW are waiting for each other and will never again become available. If a resource on a given part path is involved in a CB, then all downstream activity along that part path will eventually end. That is, after some time, the downstream jobs on that part path will never again be performed. Let $\overline{C}=\{C_i\}$ be a set of disjoint CW. Then, $\overline{C}$ is said to be in CB if each CW C_i is in CB.

2.6 Siphons

The analysis of CB and deadlock can be carried out formally using the notion of siphon. A siphon is a set of places having the property that its input transition set is contained in its output transition set i.e.

$$\bullet S \subset S \bullet \tag{2.4}$$

A siphon has the key property that, once it is unmarked, it remains so.

A minimal siphon of a CW C is the smallest siphon containing the CW. Define a critical siphon for a CW C as a smallest siphon which has the property that a CW is a CB if and only if the critical siphon is empty.

Next section defines a type of PN structure which is quite often used in wireless sensor networks for resource allocation and task planning, namely multiple reentrant flow lines.

2.7 PN Analysis of Multiple Reentrant Flow-Lines

A special case of PN is the multiple reentrant flow-line system (MRF), see figure 2.1. For MRF systems, we partition the set of places, $P = J \cup R \cup PI \cup PO$, with the places in *J, R, PI*, *PO* representing respectively, the jobs performed, the availability of resources, input of parts, and output of products. Each part path starts with a *PI*-place and terminates with a *PO*-place. We denote the set of job places *J* for part type *j* as J_j so that $J = \cup_j J_j$. Let the set of transitions along part path *j* be x_{j1}, $x_{j2},\ldots,x_{jLj}$, with x_{j1} and x_{jLj} being the initial and terminal transitions respectively.

R(*p*) is the set of resources needed by job *p*. For any resource $r \in R$ define the jobs performed by *r* as $J(r)$. We partition the resource set *R* as R_s and R_{ns}, with R_s being the set of shared resources, i.e. those needed for more than one job, and R_{ns} being the set of non-shared resources. Then, $|J(r)| = 1$ if $r \in R_{ns}$ and $|J(r)| > 1$ if $r \in R_s$, with |S| denoting the cardinality of a set *S* (i.e. the number of elements).

We formally define MRF systems as a class of systems satisfying the following properties (ϕ being the empty set):

Properties of MRF

1. $p \in P, {\bullet}p \cap p{\bullet} = \phi$
2. on part path j, $x_{j1}{\bullet} \cap P \setminus J = \phi$ and ${\bullet}x_{jLj} \cap P \setminus J = \phi$
3. $\forall p \in J, \quad {\bullet\bullet}p \cap R = p{\bullet\bullet} \cap R := R(p)$ with $|R(p)| = 1$
4. $\forall p_i, \quad p_k \in J_j, \quad i \neq k, \quad p_i \neq p_k$
5. $\forall p_i \in J_j, \quad p_k \in J_l, \quad j \neq l, \quad p_{ji}{\bullet} \cap p_{lk}{\bullet} = \phi$

7. $R_s \neq \phi$

This means that there are: (1) no self loops, (2) each part path has a well-defined beginning and an end, (3) every job requires only one resource with no two consecutive jobs using the same resource, (4) there are no part path loops, (5) for any two distinct jobs on different part paths there is no assembly, i.e. two part paths cannot merge into one, (6) there are shared resources.

According to property 3, $R(p) = \bullet\bullet p \cap R = p \bullet\bullet \cap R$, with the cardinality $|R(p)| = 1$; Under the foregoing assumption, one has $J(r) = r \bullet\bullet \cap J = \bullet\bullet r \cap J$.

In MRF, one has $|p\bullet| = 1$. A transition $x \in p\bullet$ is said to be a posterior transition of p. A decision place has multiple posterior transitions, i.e. $|p\bullet| > 1$. The resources used by decision places are called decision resources.

Figure 2.1 shows a sample MRF. In figure 2.1, the part paths are independent and neither split nor recombine. R1 is a shared resource along a single part path, and R2 is a shared resource between two part paths.

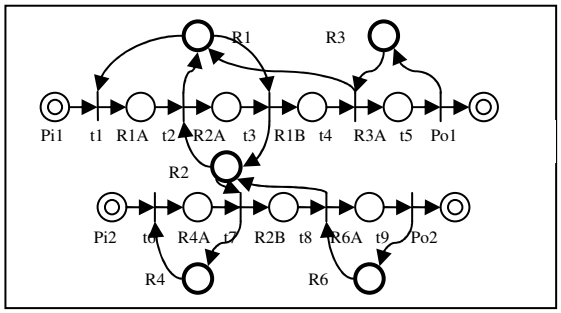


Figure 2.1 MRF system

2.8 Matrix Formulation for PN Objects

PN provides great pictorial insight and mathematical techniques for analysis, but they have sometimes had the deficiency of not providing an efficient computational framework for simple computer-based analysis. A matrix framework for computing structural objects of a PN can correct these deficiencies [47].

Commensurate with the partitioning $P = J \cup R \cup PI \cup PO$ with *J, R, PI*, and *PO* being the sets of job places, resource places, input places, and output places respectively, partition the place vector as $\vec{p} = [v \;\; r \;\; pi \;\; po]^T$, with $v \in J, r \in R, pi \in PI, po \in PO$.

Similarly, partition the input incidence matrix as $I = [F_v \quad F_r \quad F_i \quad F_o]$. Note that there are no input arcs to transitions from the places in *PO*, so that $F_o = 0$, a matrix of zeros. Partition the output incidence matrix as $O = [S_v^T \quad S_r^T \quad S_i^T \quad S_o^T]$. Note that there are no output arcs from transitions to the places in *PI*, so that $S_i = 0$. These sub-matrices are Boolean matrices having entries of 0 or 1.

2.8.1 Or/And Algebra For Computing PN Objects

Given Boolean matrices $A = [a_{ij}]$ and $B = [b_{ij}]$, define a logical or/and matrix algebra wherein addition operations are replaced by logical 'or' and multiplication operations by logical 'and'. That is, the matrix product is defined by $C = A \otimes B$ with $c_{ij} = (a_{i1} \wedge b_{1j}) \vee (a_{i2} \wedge b_{2j}) \vee (a_{i3} \wedge b_{3j}) \vee \cdots$, with $\wedge$ denoting logical 'and' and $\vee$ denoting logical 'or'. The matrix sum is defined by

$C = A \oplus B$ with $c_{ij} = a_{ij} \vee b_{ij}$. Note that these matrix products are easily performed using standard software programs including MATLAB®, etc.

Then one has computational methods for computing PN objects based on the following results in [2].

Lemma 2.1: Matrix computation of PN pre and post sets for places.

1. Let *v* represent a set of job places. Then

   a. $v\bullet$ is represented by the vector $F_v \otimes v$

   b. $\bullet v$ is represented by the vector $S_v^T \otimes v$

2. Let *r* represent a set of resource places. Then

   a. $r\bullet$ is represented by the vector $F_r \otimes r$

   b. $\bullet r$ is represented by the vector $S_r^T \otimes r$ ■

Lemma 2.2: Matrix computation of PN pre and post sets for transitions:

1. Let *x* represent a set of transitions. Then

   a. $x\bullet \cap J$ is represented by the vector $S_v \otimes x$

   b. $x\bullet \cap R$ is represented by the vector $S_r \otimes x$

   c. $\bullet x \cap J$ is represented by the vector $F_v^T \otimes x$

   d. $\bullet x \cap R$ is represented by the vector $F_r^T \otimes x$ ■

This allows one to formulate the desired scheduling strategies for MRF systems using matrices. In the next section we describe the discrete event controller for MRF systems.

2.9 Discrete Event Controller

An efficient Discrete event controller (DEC) based on matrices was first introduced in [47] and it has been in constant development [1, 2, 28, 29, 50, 82]. The DEC is completely based on matrices and it has important advantages in design, flexibility, computer simulation and online supervisory control of DE systems. The DEC has also been implemented on a practical robotic cell in [50]. This section presents a matrix-based discrete event controller for modeling and analysis of complex interconnected DE systems with shared resources, routing decisions, and dynamic resource management in a mobile wireless sensor network. This approach provides a rigorous, yet intuitive mathematical framework to represent the dynamic evaluation of DE systems according to linguistic if-then rules such as "If <conditions hold> then <consequences>".

Multi-agent systems such as one composed of mobile robots and wireless sensor network face problems of coordination. One can write down a set of if-then rules to define the mission planning of the sensor agents, such as:

Rule i: If <sensor 1 has completed task1 (data acquisition), robot 1 is available and a fire hazard is detected > then <robot 1 starts task2 and sensor 1 is released>

These linguistic rules can be easily represented in mathematical form using matrices. Following the same notation used in [1, 2, 47], let *r* be the vector of resources used in the system (i.e. mobile robots and UGSs), *v* the vector of tasks that the resources can perform (i.e. go to a given target, perform data acquisition, and deploy UGS), *u* the vector of input events (i.e. occurrence of sensor detection events) and *y* the vector of

completed missions (outputs). Finally, let x be the state logical vector of the rules of the DE controller, whose entry of '1' in position i denotes that rule i of the supervisory control policy is currently activated. Then we can define two different sets of logical equations, one for checking the conditions for the activation of rule i (matrix controller state equation), and one for defining the consequences of the activation of rule i (matrix controller output equation). In the following, all matrix operations are defined to be in the or/and algebra, where + denotes logical or and 'times' denotes logical and.

The controller state equation is

$$\bar{x} = F_v \bar{v} + F_r \bar{r} + F_u \bar{u} + F_{ud} \bar{u}_d \tag{2.5}$$

where x is the task or state logical vector, F_v is the task sequencing matrix, F_r is the resource requirements matrix, F_u is the input matrix. F_{ud} is the conflict resolution matrix and u_d is the conflict resolution vector. They are used to avoid simultaneous activation of conflicting rules, as will be shown later. The current status of the DE system includes task vector v, whose entries of `1' represent `completed tasks', resource vector r, whose entries of `1' represent `resources currently available', and the input vector u, whose entry of 1 represent occurrence of a certain predefined event (fire alarm, intrusion etc.). The over bar in equation (1) denotes logical negation so that tasks complete or resources released are represented by '0' entries. F_v is the task sequencing matrix [47], and has element (i,j) set to '1' if the completion of task v_j is an immediate prerequisite for the activation of logic state x_i. F_r is the resource requirements matrix [47] and has element (i,j) set to '1' if the availability of resource j (robot or UGS) is an immediate prerequisite for the activation of logic state x_i.

On the ground of the current status of the DE system, equation (2.5) calculates the logical vector x, i.e. which rules are currently activated. The activated rules determine the commands that the DEC has to sequence in the next iteration, according to the following equations

$$v_s = S_v x \tag{2.6}$$

$$r_s = S_r x \tag{2.7}$$

$$y = S_y x \tag{2.8}$$

S_v is the task start matrix and has element (i,j) set to '1' if logic state x_j determines the activation of task i. S_r is the resource release matrix and has element (i,j) set to '1' if the activation of logic state x_j determines the release of resource i. S_y is the output matrix and has element (i,j) set to '1' if the activation of logic state x_j determines the completion of mission i.

The task start equation (2.6) computes which tasks are activated and may be started, the resource release equation (2.7) computes which resources should be released (due to completed tasks) and the mission completion equation (2.8) computes which missions have been successfully completed.

Vector v_s, whose `1' entries denote which tasks are to be started, and vector $r_s$, whose `1' entries denote which resources are to be released, represent the commands sent to the DE system by the controller. '1' entries in vector y denote which missions have been successfully completed.

Equations 2.5-2.8 represent the rule-base of the supervisory control of the DE system. All the coefficient matrices are composed of Boolean elements and are sparse, so that real time computations are easy even for large interconnected DE systems.

The task sequencing matrices (F_v and S_v) are direct to write down from the required operational task sequencing. On the other hand, the resource requirements matrices (F_r, S_r) are written down based on the resources needed to perform the tasks and are assigned independently of the task sequencing matrices. Matrix F_{ud} in equation (2.5) is used to resolve conflicts of shared resources, i.e. conflicts deriving by the simultaneous activation of rules, which start different tasks requiring the same resource. Matrix F_{ud} has as many columns as the number of tasks performed by shared resources. Element (i,j) is set to '1' if completion of shared task j is an immediate prerequisite for the activation of logic state x_i. Then an entry of '1' in position j in the conflict resolution vector u_d, determines the inhibition of logic state x_i (rule i cannot be fired). It results that, depending on the way one selects the conflict-resolution strategy to generate vector u_d, different dispatching strategies can be selected, in order to avoid resource conflicts or deadlocks.

To use the DEC as a Supervisory Controller for task assignment and resource dispatching in mobile wireless sensor networks, one needs to have an architecture that is modular, flexible and adaptable. One such architecture consists of three layers, namely agent control layer, network control layer and organization control layer. The important aspect of this architecture is that improvements and updates on one layer results in minor changes in other layers, making the system intelligent and adaptable.

The first layer (agent control level) deals with the control of each agent (being either a UGS or a mobile robot), keeping into account its peculiar functionalities. At this level one defines the processing capabilities of the UGSs (e.g. signal processing) and the control algorithms for the behavior of each robot (e.g. reach the target, follow another robot etc.). The second layer (network control level) deals with the implementation of communication protocols for energy efficient data transmission between the UGS, robots and the supervisor. The third layer (organization control level) consists of matrix-based DE supervisory controller whose matrix formulation allows one to employ a high-level human interface to define the mission planning, the resource allocation and the dispatching rules. The supervisor is in charge of sequencing the tasks each agent has to perform according to the perception of the environment; assuming that the agent level controllers correctly perform the assigned tasks and that the communication protocol for each agent perfectly works.

The complete architecture is shown in the figure 2.2.

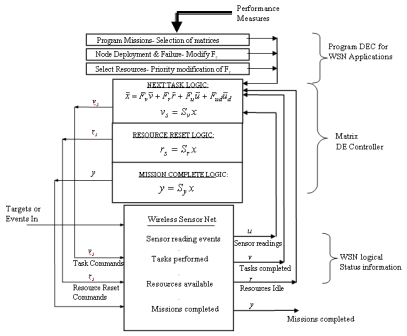


Figure 2.2 Complete System Architecture

Thus, using this architecture, a complex system can be decomposed into missions, tasks and rules for task sequencing, resource dispatching and conflict resolution (figure 2.3).

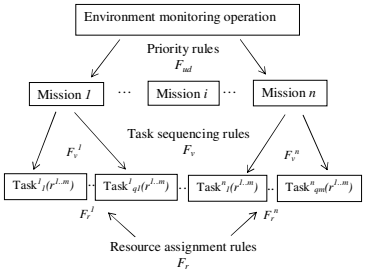


Figure 2.3 Sequencing of missions

This block diagram can be represented using matrices using the technique of DEC. Suppose that there are m resources r^j $j=1…m$ (mobile robots and stationary sensors) each one capable of performing p_j tasks, and define n different missions, each one composed of q_i tasks. For each mission, there are corresponding set of matrices F_v^i, F_r^i, S_v^i, S_r^i which represent the coordination rules of the agents in the execution of the tasks. In order to take into account the priority among missions, there is a global conflict resolution matrix $F_{ud.}$. After assigning a priority order k to each mission, calculate for every resource j and every mission i, a matrix ($F_{ud}^i\left(r_j\right)$), creating a new column for every '1' appearing in the *jth* column of F_r^i. Then one constructs the global conflict resolution matrix of resource r_j ($F_{ud}\left(r_j\right)$) inserting each $F_{ud}^i\left(r_j\right)$ matrix in position *(i,k)*.

As shown in figure 2.4 the matrix formulation of the overall environment monitoring operation is then obtained by stacking the set of matrices together. The correspondence between figure 2.3 and the matrices is very obvious.

$$F_{ud} = \left[F_{ud}(r_1)...F_{ud}(r_j)...F_{ud}(r_m)\right]$$

$$q_1 \quad ... \quad q_i \quad ... \quad q_n \qquad\qquad m$$

$$F_v = \begin{bmatrix} F_v^1 & & & & \\ & ... & & & \\ & & F_v^i & & \\ & & & ... & \\ & & & & F_v^n \end{bmatrix} \quad F_r = \begin{bmatrix} F_r^1 \\ ... \\ F_r^i \\ ... \\ F_r^n \end{bmatrix}$$

Figure 2.4 Matrix formulation

In WSN, two issues have to be tackled i.e. adaptability and scalability. Adaptability and scalability are crucial requirements to guarantee optimal performances for agent team operations. These issues can be tackled by using the principle of DEC.

Adaptability: Adaptability is the ability of an agent team to change its behavior according to the dynamical evolution of the environment. Following methods make the system adaptable using DEC.

1. Implementation of distributed algorithms:

Adaptability can be resolved both at the agent control level and at the supervisor control level. In the agent control level, individual agents are autonomous and perform tasks using their perception of the environment. In the framework of the DEC, these operations can be considered as a generic (fully decentralized) mission *i* (or part of it) composed of simultaneous tasks. Therefore there is enhanced adaptability decision, at

the supervisor level (on the grounds of the present situation), which decentralized mission has the priority (changing F_{ud}^{i}) and which resources should be used (changing F_{r}^{i} and S_{r}^{i}).

2. Dynamic reallocation of resources:

A dynamic reallocation of the agents to missions can be performed by rearranging the '1' relative to similar resources in the matrices F_r and S_r when new missions (or new agents) are added. Due to the matrix representation of the mission plans, these objectives can be pursued using computationally efficient algorithms.

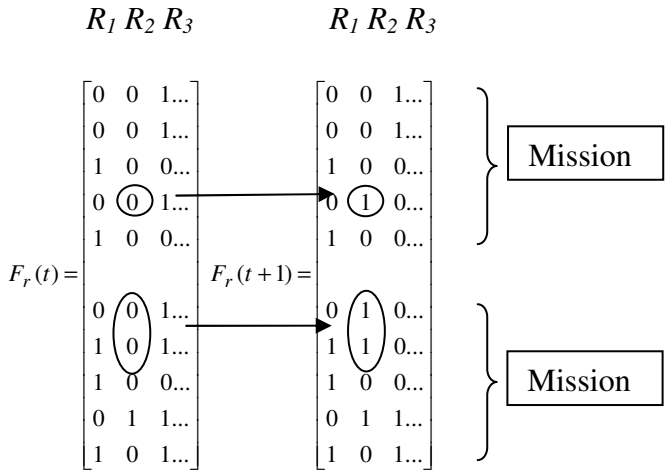


Figure 2.5 Reallocation of resources through matrix operations

Figure 2.5 shows an example of reallocation of tokens among three similar resources (R_1, R_2 and R_3) in the case of two missions. After the reallocation, the workload of the resources is more balanced since each resource performs a similar number of tasks (equal to the number of '1' in the corresponding column).

3. Combining multiple plans for the same mission:

In certain circumstances, different sequences of tasks can be used to implement the same mission. A computationally efficient algorithm can be used to combine the plans together and derive one single compact matrix representation for the DEC. In this way, the DEC automatically sequences the most suitable succession of tasks depending on the current available resources.

4. Priority among missions:

Another way to adapt to the WSN control scenario is to make the set of mission priority rules adaptable. For example, suppose that resource r_1 is shared among three different missions whose priority rank is *3, 1, 2*. After defining the conflict resolution matrix of r_1 for each mission ($F_{ud}^1(r_1)$, $F_{ud}^2(r_1)$, $F_{ud}^3(r_1)$), the overall conflict resolution matrix of r_1 ($F_{ud}(r_1)$) is built as shown in figure 2.6.

$$F_{ud}(r_1) = \begin{array}{r} \\ mission^1 \\ mission^2 \\ mission^3 \end{array} \begin{array}{c} \begin{array}{ccc} priority^1 & priority2 & priority3 \end{array} \\ \begin{bmatrix} 0 & F_{ud}^1(r1) & 0 \\ 0 & 0 & F_{ud}^2(r1) \\ F_{ud}^3(r1) & 0 & 0 \end{bmatrix} \end{array}$$

Figure 2.6 Initial priorities

If the priority of the missions changes in *2, 3, 1* then one can have the following configuration as shown in figure 2.7.

$$F_{ud}(r_1) = \begin{array}{c} \\ mission^1 \\ mission^2 \\ mission^3 \end{array} \begin{array}{c} \begin{array}{ccc} priority^1 & priority2 & priority3 \end{array} \\ \begin{bmatrix} 0 & 0 & F_{ud}^1(r_1) \\ F_{ud}^2(r_1) & 0 & 0 \\ 0 & F_{ud}^3(r_1) & 0 \end{bmatrix} \end{array}$$

Figure 2.7 Changed priorities

Thus, a change of priority results in a simple permutation of the block matrices F_{ud}^i for each resource.

Scalability: Scalability defines the possibility to add and remove agents. One can use the DEC to tackle scalability at the supervisor level, updating the matrix based representation of the missions to take into account the failure of agents as well as the adding of new ones.

If a new agent is added to the system, a new column is added in the matrices F_r and $S_r^{'}$ (S_r transpose). Then, dispatching algorithms (based on matrix operations) can be applied to rearrange the tasks among resources. In a similar fashion, an agent failure can be tackled rearranging the tasks among the resources so that the column vectors relative to the failed resources in F_r and S_r' are null. In the following example, a simple algorithm is used for reallocating (off-line, i.e. when no missions are in progress) resources after agent failure. The mission planning is revised in such a way that predefined back-up agents execute the tasks of the failed agents. In the matrix formulation this is equivalent to move the elements equal to one in the matrices F_r^i and S_r^i from the column of the failed resource to the column of the back-up resource.

This can be achieved through a simple linear combination of the columns of F_r^i and S_r^i respectively. Thus,

$$F_r^{i,new} = F_r^{i,old} \cdot B^i \tag{2.9}$$

$$S_r^{i,new} = S_r^{i,old} \cdot B^i \tag{2.10}$$

where B^i is a square matrix of dimension equal to the number of the resources of the system. The diagonal elements of B^i (a_j) are parameters which are equal to 1 if resource r_j is working properly and 0 otherwise. On row j the element (j,j) is equal to a_j and the element (j,k) is equal to $1- a_j$, where k is the column of matrix F_r^i corresponding to the back-up resource of agent j for mission i. If a_j =0, the *jth* column of $F_r^{i,new}$ will be null (meaning that agent j is not supposed to perform any task) and the *kth* column will have '1's in correspondence of the tasks for which resource j was required. If no failure occurs, B^i is the identity matrix and mission plans are not changed. Clearly, in the definition of the matrix B^i one has to make sure that each back-up agent does not perform any simultaneous task with the resource they are supposed to substitute. For example, suppose there are three agents, and that, for a certain mission *i,* the resource requirement matrix and the back-up matrix B^i are as shown in figure 2.8.

$$F_r^{i,old} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \quad B^i = \begin{pmatrix} a_1 & 1-a_1 & 0 \\ 0 & a_2 & 0 \\ 1-a_3 & 0 & a_3 \end{pmatrix}$$

Figure 2.8 Initial resource matrix

The B^i matrix corresponds to the case where, in mission *i*, agent 2 is the backup of agent 1, agent 2 has no back-up and agent 1 is the back up of agent 3. If agent 1 fails (a_1=0) whereas agent 2 and 3 work properly (a_2=a_3=1),

$$B^i = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ and } F_r^{\,i,new} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

Figure 2.9 Changed resource matrix

i.e., in mission *i*, agent 1 has been replaced by agent 2.

Another method to cope up with agent failure is by routing resources, which means that one has to define a set of multiple resource choices initially for certain critical tasks. The routing resources automatically assign to the task the first available resource of the corresponding set providing redundancy and robustness against agent failures. This novel matrix formulation supports task routing efficiently, since there is no need to distinguish between physical and logical resources. Routing of resources will be explained later in the chapter.

2.10 Implementation on Sentries & UGS

It is well known that a matrix approach can be used to describe the marking transitions of a Petri Net using the PN transition equation

$$m(t+1) = m(t) + (S' - F) \cdot x(t) \qquad (2.11)$$

where *S* and *F* are the output and input incidence matrix respectively. This equation gives a useful insight on the dynamics of discrete event systems but does not provide a complete dynamical description of DE systems.

Observe that the vector *x* in equation (2.11) is the same as in equation (2.5), then one may identify *x* as the vector associated with the PN transitions and *u, v, r,* u_d as associated with the places. Then it follows that,

$$m(t) = \left[u(t)', v(t)', r(t)', u_d(t)'\right]' \tag{2.12}$$

$$S = \left[S_u', S_v', S_r', S_{u_d}', S_y'\right]' \tag{2.13}$$

$$F = \left[F_u', F_v', F_r', F_{u_d}', F_y'\right]', \tag{2.14}$$

Therefore, we can use equation (2.5) to generate the allowable firing vector to trigger transitions in equation (2.11). The combination of the DEC and the PN marking transition equation, therefore, provides a complete dynamical description of the system.

In order to take into account the time durations of the tasks and the time required for resource releases, one can split *m(t)* into two vectors, one representing available resources and current finished tasks ($m_a(t)$) and the other representing the tasks in progress and idle resources ($m_p(t)$)

$$m(t) = m_a(t) + m_p(t) \tag{2.15}$$

This is equivalent to introducing timed places in a Petri Net and to dividing each place into two parts, one relative to the pending states (task in progress, resource idle) and the other relative to the steady states (task completed and resource available). As a consequence, we can also split equation (2.15) into two equations

$$m_a(t+1) = m_a(t) - F \cdot x(t) \tag{2.16}$$

$$m_p(t+1) = m_p(t) + S' \cdot x(t) \tag{2.17}$$

When a transition fires a token is moved from $m_p(t)$ to $m_a(t)$ where it may be used to fire subsequent transitions. Therefore equations (2.5), (2.16) and (2.17) represent a complete description of the dynamical behavior of the discrete event system and can be implemented for the purposes of computer simulations using any programming language (e.g. Matlab® or C). In the case of a mobile wireless sensor network, where experiments on wide and hostile areas can be really complex and challenging, it allows one to perform extensive simulations of the control strategies and then test experimentally only those that guarantee the most promising results.

Consider an experimental scenario; A network consisting of two mobile robots and two wireless sensors. Two different missions have been implemented to show the potentialities of the proposed DEC. In the first mission, after one of the sensors launches an intruder alert, the network automatically reconfigures its topology to further investigate the phenomenon. In the second mission, one of the sensors detects a huge vibration indicating that there is an earthquake. The procedure for implementing the supervisory control policy consists of three different steps. First of all one defines the vector of resources r present in the system and the tasks they can perform. In ARRI DIAL test-bed there are two robots (R_1 and R_2), each one able of performing certain number of tasks (say 4 or 5), and two stationary sensors (UGS_1, UGS_2), each one able of performing one task (i.e. taking measurement). The resource vector is $r = [R_1, R_2, UGS_1, UGS_2]$.

Then for each mission i, define the vector of inputs u^i, of outputs y^i and of tasks v^i, and the task sequence of each mission (refer table 2.1 and 2.2 for mission 1 and

mission 2), and write down the if-then rules representing the supervisory coordination strategy to sequence the programmed missions (table 2.3 and table 2.4). In the definition of the rule bases particular attention has to be devoted to the definition of consecutive tasks performed by the same resources. If the consecutive tasks are interdependent (e.g. *go to sensor 2* and *retrieve sensor 2*), the corresponding resource should be released just at the end of the last of the consecutive tasks. Instead, if the tasks are not interdependent, before starting the new consecutive task, the DEC releases the corresponding resource and makes sure that no other missions are waiting for it. If after a predetermined period of time no other missions request that resource, the previous mission can continue. Finally translate the linguistic description of the coordination rules into a more convenient matrix representation, suitable for mathematical analysis and computer implementation. As an example, the following shows a derivation of the matrix formulation from the rule-base of mission 1 (table 2.1).

For example, considering the rule-base of mission1 (table 2.2), one can easily write down the F_v^1 and F_r^1 matrices considering that $F_v^1(i, j)$ is '1' if task j is required as an immediate precursor to rule i and $F_r^1(i, j)$ is '1' if resource j is required as an immediate precursor to rule i.

If one sees Mission 2, Robot 1 is shared resource. Hence one needs to construct the F_{ud} matrix. The S_v^1 matrix is built considering which tasks should be executed after a rule fires. The S_r^1 matrix is built considering that $S_r^1(i,j)$ is 1 if resource i has

to be released after rule *j* has been fired. In the same way, the set of matrices relative to mission 2 can be built.

Table 2.1 Mission 1 Task sequence

mission1	**Notation**	**Description**
Input 1	U^1	*UGS1* launches earthquake alert
Task 1	*T1a*	*UGS2* takes measurement
Task 2	*T2a*	*R1* goes to *UGS1* and takes measurement
Task 3	*T3a*	*UGS1 takes measurements again*
Task 4	*T4a*	*R2* goes to *UGS2* and takes a measurement.
output	Y^1	False Alarm, Mission 1 completed

Table 2.2 Mission 1 Rule-base

Mission1-operation sequence	
Rule1 x_1^1	If input then T1a
Rule2 x_2^1	If T1a then T2a
Rule3 x_3^1	If T2a then T3a
Rule4 x_4^1	If T3a then T4a
Rule5 x_5^1	If T4a then y

Table 2.3 Mission 2 Task sequence

Mission2	**Notation**	**Description**
input	U^2	*UGS2* detects intruder
Task 1	*T1b*	*UGS2* takes measurement again
Task 2	*T2b*	*UGS1* takes measurement
Task 3	*T3b*	*R1* goes to *UGS2*

Table 2.3- *continued*

Task 4	*T4b*	*R2* goes to *UGS1*
Task 5	*T5b*	*R1* goes to the door
output	Y^2	Intruder detected, Mission 2 completed

Table 2.4 Mission 2 Rule-base

Mission2- operation sequence	
Rule1 x_1^2	If *input then T1b*
Rule2 x_2^2	If T1b then T2b
Rule3 x_3^2	If T2b then T3b
Rule4 x_4^2	If T3b then T4b
Rule5 x_5^2	If T4b then T5b
Rule6 x_6^2	If T5b then y^2

$$F_v^1 = \begin{array}{c} \\ x_1^1 \\ x_2^1 \\ x_3^1 \\ x_4^1 \\ x_5^1 \end{array} \begin{array}{c} \begin{array}{cccc} \text{T1a} & \text{T2a} & \text{T3a} & \text{T4a} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \qquad F_r^1 = \begin{array}{c} \\ x_1^1 \\ x_2^1 \\ x_3^1 \\ x_4^1 \\ x_5^1 \end{array} \begin{array}{c} \begin{array}{cccc} \text{R1} & \text{R2} & \text{UGS1} & \text{UGS2} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

(a) (b)

Figure 2.10 Mission1 job sequencing matrix F_v^1 (a), resource requirement matrix F_r^1 (b)

$$S_v^1 = \begin{array}{c} \\ T1a \\ T2a \\ T3a \\ T4a \end{array} \begin{array}{c} \begin{array}{ccccc} x_1^1 & x_2^1 & x_3^1 & x_4^1 & x_5^1 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \qquad S_r^1 = \begin{array}{c} \\ R1 \\ R2 \\ UGS1 \\ UGS2 \end{array} \begin{array}{c} \begin{array}{ccccc} x_1^1 & x_2^1 & x_3^1 & x_4^1 & x_5^1 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

(a) (b)

Figure 2.11 Mission1 Task start matrix S_v^1 (a) and resource release matrix S_r^1 (b)

One can implement the control system directly on the WSN test-bed. Figure 2.16 shows the actual experimental utilization time trace of the agents, assigning higher priority to mission 1. Notice that the time duration of the real WSN runs in terms of discrete-event intervals. This is a key result since it shows that the DEC allows one to perform a "simulate and experiment" approach for a WSN, with noticeable benefits in terms of cost, time and performance.

$$F_v^2 = \begin{array}{c} \\ x_1^2 \\ x_2^2 \\ x_3^2 \\ x_4^2 \\ x_5^2 \\ x_6^2 \end{array} \begin{array}{c} \begin{array}{ccccc} \text{T1b} & \text{T2b} & \text{T3b} & \text{T4b} & \text{T5b} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \qquad F_r^2 = \begin{array}{c} \\ x_1^2 \\ x_2^2 \\ x_3^2 \\ x_4^2 \\ x_5^2 \\ x_6^2 \end{array} \begin{array}{c} \begin{array}{cccc} \text{R1} & \text{R2} & \text{UGS1} & \text{UGS2} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

(a) (b)

$$F_{ud}^2(R1) = \begin{array}{c} \\ x_1^2 \\ x_2^2 \\ x_3^2 \\ x_4^2 \\ x_5^2 \\ x_6^2 \end{array} \begin{array}{c} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \end{array} \quad \text{(c)}$$

Figure 2.12 Mission2 Task sequencing matrix F_v^2 (a), resource requirement matrix F_r^2 (b) and conflict resolution matrix $F_{ud}^2(R1)$ (c)

$$S_v^2 = \begin{array}{c} \\ T1b \\ T2b \\ T3b \\ T4b \\ T5b \end{array} \begin{array}{c} \begin{array}{cccccc} x_1^2 & x_2^2 & x_3^2 & x_4^2 & x_5^2 & x_6^2 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \qquad S_r^2 = \begin{array}{c} \\ R1 \\ R2 \\ UGS1 \\ UGS2 \end{array} \begin{array}{c} \begin{array}{cccccc} x_1^2 & x_2^2 & x_3^2 & x_4^2 & x_5^2 & x_6^2 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

(a) (b)

Figure 2.13 Mission2 Task start matrix s_v^2 (a) and resource release matrix s_r^2 (b)

$$F_v = \begin{array}{c} \\ x^1 \\ x^2 \end{array} \begin{array}{c} \begin{array}{cc} F_v^1 & \end{array} \\ \begin{pmatrix} F_v^1 & 0 \\ 0 & F_v^2 \end{pmatrix} \end{array} \quad F_r = \begin{array}{c} \\ x^1 \\ x^2 \end{array} \begin{array}{c} \begin{array}{c} F_r^1 \end{array} \\ \begin{pmatrix} F_r^1 \\ F_r^2 \end{pmatrix} \end{array}$$

$$S_v = \begin{array}{c} \\ v^1 \\ v^2 \end{array} \begin{array}{c} \begin{array}{cc} S_v^1 & \end{array} \\ \begin{pmatrix} S_v^1 & 0 \\ 0 & S_v^2 \end{pmatrix} \end{array} \quad S_r = r\begin{pmatrix} S_r^1 & S_r^2 \end{pmatrix}$$

Figure 2.14 Overall monitoring operation- Matrix formulation matrices F_v, F_r, S_v, S_r

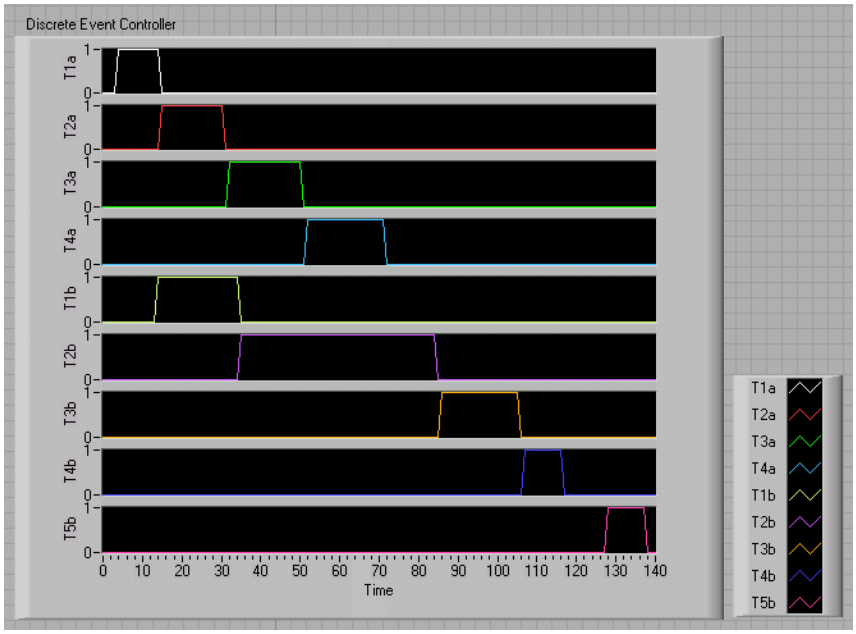


Figure 2.15 Utilization time trace of the WSN- Experimental results

2.11 Decision for routing of resources

"Reentrant flow lines" are of great importance in manufacturing systems [34] where the resources needed for each job are pre-defined. Resource assignment and dispatching for such systems is well understood [32, 47]. But in the case of mobile WSN there are many resources such as distributed sensors and it is not known beforehand which sensor is most useful for resolution of certain events. Dynamic sensor selection is a special sort of *routing problem* [4, 13, 55], or free-choice Petri Net, which requires highly complex decision-making. Therefore, one can use a new method for dealing with dynamic sensor selection using a novel Dynamic Priority Assignment Weighting Matrix.

Greedy activity/ resource selector algorithm [9] can be used for dynamic selection of resources most appropriate for a task in the DEC format. This can be done in the following method:

For each task that has a choice of resources to use, define a Dynamic Priority Assignment Matrix (DPAM) according to the example:

$$D_c = \begin{array}{c} \\ task\ 1 \\ task\ 2 \\ task\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} res.1 & res.2 & res.3 \end{array} \\ \begin{bmatrix} 0 & 1 & 0.6 \\ 1 & 0 & 0.4 \\ 0.1 & 0 & 1 \end{bmatrix} \end{array}$$

which indicates that task 1 may be efficiently performed by resource 2, or less efficiently by resource 3. The numerical entry in position *(i,j)* is between 0 and 1, and indicates the efficiency with which resource *j* performs task *i*, with 0 indicating that resource *j* cannot perform task *i*, and 1 indicating that resource *j* performs task *i* with

maximum efficiency. Note that this matrix indicates that task 1 may be performed with *either* resource 2 *or* resource 3, in contrast to the matrix F_r, where multiple entries of 1 in a row indicate that *all* those resources are required for that task.

According to greedy dispatching policies [9], one selects the resource to perform a given task according to the immediate 1-step look ahead maximum payoff. Thus, depending on the DPAM, at each step, for the free-choice tasks modify the resource matrix F_r to have 1's in the entries corresponding to the maximum values of the DPAM in each row. This effectively selects the current most efficient resource to perform each task. Then, compute the DEC equations (2.5)-(2.8) to determine which tasks to start and which resources to reset.

The DPAM dynamically updates based on the evaluation information from the task on how well the assigned resource performed, after the tasks are over. Thus, resources that perform well will be assigned next time to that task. In this way, the DPAM ensures that an optimal resource is always assigned to a particular task to improve the overall efficiency of the system.

The next chapter describes the concept of dynamic resource allocation in MRF systems with deadlock avoidance.

CHAPTER 3

MULTIPLE REENTRANT FLOW-LINES WITH DYNAMIC RESOURCE ALLOCATION

3.1 Need for Dynamic Resource Assignment

The coordination of a multi-robot system for the execution of cooperative tasks is a very active research field [6, 10, 11, 26, 27]. In particular, related literature has extensively tackled the problem of dynamic resource assignment, to on-line allocate a resource to a task according to predefined criteria (see e.g. [26] for a thorough overview). However in most cases the robot team is in charge of executing one mission at a time and no shared resource conflicts arise. If multiple missions are executed simultaneously, then system deadlock or conflicts might arise if the resource assignment is not properly made [17, 32, 47, 50]

The main objective of this chapter is to implement dynamic coordination of a mobile sensor network in presence of multiple missions. Therefore it is necessary to develop a control strategy in charge of simultaneously performing dynamic resource assignment and solving on-line shared resource conflicts.

In related literature the implementation of multiple missions for multi-robot systems has not been adequately studied. However, since the peculiarities of MSN require multiple competing missions and network topology changes, including mobility and addition/removal of nodes, dynamic resource assignment algorithms should be included in the framework of the DEC.

This chapter presents a novel approach to implement on-line deadlock-free resource assignment for multi-robot systems with multiple missions. At each event occurrence, when resource changes are required, a greedy algorithm is first implemented by on-line updating of the resource requirements matrix [47]. The new resource assignment is accepted if it is compatible with a certain MAXWIP deadlock avoidance policy specified herein. Specifically, the discrete event system representing the new mission plan has to satisfy two conditions: After the implementation of the new resource assignment, it is necessary to guarantee that (1) the system is not already in deadlock and (2) the new system is regular in a sense described herein. In order to check the latter condition the regularity test proposed in [32] is launched every time a new resource assignment is proposed. Having produced an allowable assignment of resources, the Discrete Event Controller (DEC) described in Chapter 2 is run to assign the next tasks in the multiple missions based on priority assignment policies.

3.2 One Step Look-ahead Deadlock Avoidance

The matrix approach in the discrete event controller presented in Chapter 2 provides a rigorous, yet an intuitive mathematical framework to represent the dynamic evolution of DE systems according to linguistic if-then rules. As shown in [47], the presented matrix constructions can also be efficiently used to implement deadlock avoidance policies for discrete event multi-robot systems.

3.2.1 Circular Waits

As mentioned in Chapter 2, for any two resources r_i and r_j, r_i is said to wait for r_j, denoted $r_i \rightarrow r_j$, if the availability of r_j is an immediate requirements for the release of r_i. Circular waits (CW) among resources are a set of resources r_a, r_b,… r_w whose wait relationship among them are $r_a \rightarrow r_b \rightarrow \ldots \rightarrow r_w$ and $r_w \rightarrow r_a$. The simple circular waits (sCW) are primitive CWs which do not contain other CWs. For a complete analysis of the deadlock structures, all the CWs need to be identified, not only the sCWs.

In order to avoid deadlocks, we have to monitor those tasks of the MSN whose completion activate rules which consume resources in a CW. The task set of a CW *C*, *J(C)*, is the set of tasks which need at least one of the resources of *C* to be started. Under the assumptions previously presented, a deadlock condition occurs if and only if there is an empty circular wait [92]. For these systems, an empty CW can only be caused by activation of tasks of the corresponding critical subsystem. Using the matrix formulation of (1)–(4) and some matrix manipulations, we can come up with a compact matrix representation of critical subsystems J_o as follows [47].

$$J_o = ({}_dCF_v) \wedge (C_dF_v) = (C_dS_v^T) \wedge (C_dF_v) \tag{3.1}$$

where each entry of '1' in position (*i, j*) means that task *j* is included in the critical subsystem of CW *i*. ${}_dC$ and C_d are called the input and output rules of a CW and have '1' entry in position *(i, j)* if the j^{th} rule increases or reduces the number of available resources in the i^{th} CW respectively.

A simple deadlock avoidance strategy consists in not allowing the number of activated tasks of the critical subsystem to become equal or greater than the number of available resources in the i^{th} CW C_i (MAXWIP policy [47]).

$$m(J_o(C_i)) < m_o(C_i) \tag{3.2}$$

Therefore, we can conveniently update the conflict resolution input u_d to inhibit rules which, if activated, would violate condition (9) and lead to deadlock conditions.

Our dispatching policy follows three main steps:

i. Based on the structure of the system defined by matrices F and S, we calculate the CWs, their corresponding critical subsystems and the number of available resources $m_o(C_i)$ in the ith CW C_i.

ii. For every DE-iteration, we calculate from the current marking vector, $m_{current}$, the corresponding possible successor-marking vector, $m_{possible}$. Equation (6) provides this possible successor $m_a(t+1)=m_{possible}$; $m_a(t)=m_{current}$; $m_{possible}$ is readjusted keeping into account possible shared resource conflicts (on-line computation). If the selected $m_{possible}$ does not satisfy condition (9), then it is necessary to eliminate the task that is attempting to cause a deadlock, inhibiting the corresponding rule. This is done by conveniently updating vector u_d. Then the algorithm restarts from step 2 (on-line computation).

3.3 Deadlock-free Dynamic Resource Assignments

The concept of dynamic resource assignment is highly substantial in multi-robot systems to adapt to unstructured and dynamic environments [26]. In other words, to

improve the coordination of a mobile sensor network, it is necessary to continuously update the set of matrices defined in the DEC based on new environmental conditions.

To cast the dynamic selection of resources most appropriate for a task into the DEC format, one may use the Greedy activity/resource selector algorithm from [9] to on-line modify the resource assignment matrix F_r as follows.

For each task that has a choice of resources to use, define a Dynamic Priority Assignment Matrix (DPAM) (Chapter 2) according to the example:

$$D_c = \begin{array}{r} \\ task\,1 \\ task\,2 \\ task\,3 \end{array} \begin{array}{ccc} res.1 & res.2 & res.3 \\ 0 & 1 & 0.7 \\ 1 & 0 & 0.5 \\ 0.2 & 0 & 1 \end{array}$$

which indicates that task 1 may be efficiently performed by resource 2, or less efficiently by resource 3. The numerical entry in position (*i, j*) is between '0' and '1', and indicates the efficiency with which resource *j* performs task *i*, with '0' indicating that resource *j* cannot perform task *i*, and '1' indicating that resource *j* performs task *i* with maximum efficiency. Note that this matrix indicates that task 1 may be performed with *either* resource 2 *or* resource 3, in contrast to the matrix F_r, where multiple entries of '1' in a row indicate that *all* those resources are required for that task.

According to greedy dispatching policies [29], one selects the resource to perform a given task according to the immediate 1-step look ahead maximum payoff. The algorithm looks for an ideal resource for a particular task. If it does not find a resource, it waits for a resource that is most suitable and available for that particular task. The task is not started till the resource is found. Therefore, depending on the

DPAM, at each event step, for the free-choice tasks, the resource matrix F_r is modified to have 1's in the entries corresponding to the maximum values of the DPAM in each row. This effectively selects the current most efficient resource to perform each task. Then, the DEC equations (1)-(4) are computed to determine which tasks to start and which resources to reset.

After the tasks have been performed, the DPAM is dynamically updated based on evaluation information from the task on how well the assigned resource performed. Thus, resources that perform well will be assigned next time to that task. The implementation of this resource assignment policy optimizes the single association resource/task without taking into account the global effect of all the associations. In multi-mission systems this way of proceeding may lead to shared resource conflicts and deadlocks. In [32] a similar approach was used which did not take into consideration the dynamic resource assignment problem.

In [28], the implementation of the deadlock avoidance policy assumed fixed structure of the matrices *F* and *S*. However the MAXWIP deadlock avoidance policy presented in Section 3.2 is based only on the current status of the system, i.e. on the current configuration of matrices *F* and *S*. Therefore it is possible, after every iteration, to update the resource requirement matrices of the DEC using the DPAM and then check if the resource assignment conflicts with the MAXWIP deadlock avoidance policy. If a conflict arises, then a new resource assignment is presented until the requirements of the MAXWIP policy are met. Figure 3.1 shows a flowchart

representing the procedure to update the DEC to adapt to the changing operating conditions while guaranteeing a deadlock free dispatching.

The procedure is as follows:

1) After a new transition of the DEC, if a resource reallocation or a priority change is needed, the mission plans can be accordingly updated by redefining matrices F_r, S_r and F_{ud} through a human operator or an automatic decision making algorithm (DPAM). The only constraint to be observed is that it is not possible to reassign the resource to a task currently in progress.

2) We then calculate the new set of circular waits CW and the new sets of critical subsystem J_o. At this point, before applying the MAXWIP policy, two conditions must be met.

First of all we have to apply Gurel's regularity test described in [32], in order to be sure that no key resources [50] are present in the system after applying the new resource assignment. The output of the Gurel's algorithm is a matrix Res_{cw} which is a matrix of critical resources. There is a certain pathological case that requires extreme care in the process of deadlock avoidance and dispatching. This situation is called second level deadlock (SLD) (see [19, 20, 21, 22] for more details). SLD is not a circular wait even if necessarily evolves into a deadlock in the near future. SLD exists on the presence of critical resources known as bottlenecks ([50]) and key resources. Bottleneck resources are identified by analyzing interconnectivities in circular wait relationships. Secondly, we have to make sure that the system is not already in a deadlock situation.

If at least one of these two conditions is not satisfied then another resource assignment attempt is made and algorithm restarts from step 1.

3) If the system is regular and not currently in deadlock, then the new resource assignment is actually implemented. The MAXWIP policy is compatible with the new system.

This procedure can be seen as a constrained optimization in resource assignment. It ensures that all the tasks are performed using the best resources available which do not cause the occurrences of deadlocks.

3.4 Simulation Results

Matlab simulation results are proposed to illustrate the proposed control approach. Two missions have been implemented for a sensor network composed of 7 mobile robotic sensors. The two missions encompass a sequence of 4 and 7 tasks respectively. The proposed algorithm can be easily extended to networks with several resources.

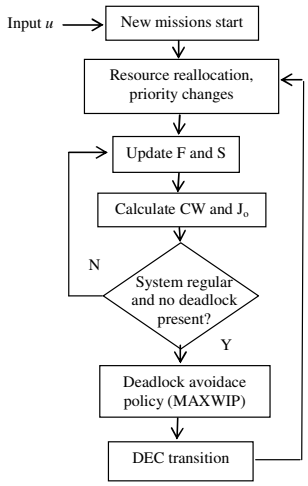


Figure 3.1 Flow chart representation of the deadlock-free dynamic resource assignment algorithm

3.4.1 Resource Assignment-Attempt 1

Consider figure 3.2. Suppose that at a certain instant of time, a new resource assignment is performed and the resource requirement matrix is accordingly updated. Also, suppose that resources *R1*, *R2*, *R3* and *R4* are currently busy, i.e. the resource vector is $r = [0\ 0\ 0\ 0\ 1\ 1\ 1]$.

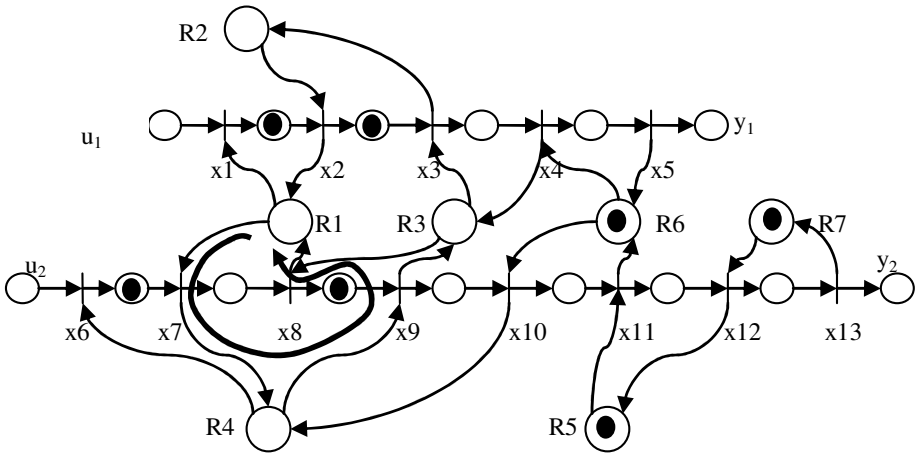


Figure 3.2 Petri net representation of the system after attempt 1

The new resource matrix F_r would then be:

$$F_r' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Using the equation presented in previous section we can then calculate the matrices of circular waits, critical subsystems and critical resources respectively corresponding to the new resource assignment:

$$CW = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} J_o = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathrm{Re}\, s_{cw} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Analyzing vector *r* and matrix *CW*, it results that resources R1, R3, and R4, which compose the first circular wait (first row of matrix CW), are all busy. If this resource assignment was accepted, the circular wait would be empty and the system would be in a deadlock (see figure 3.3).]. For sake of clarity Figure 2 reports the Petri Net corresponding to this first candidate resource assignment highlighting the empty circular wait.

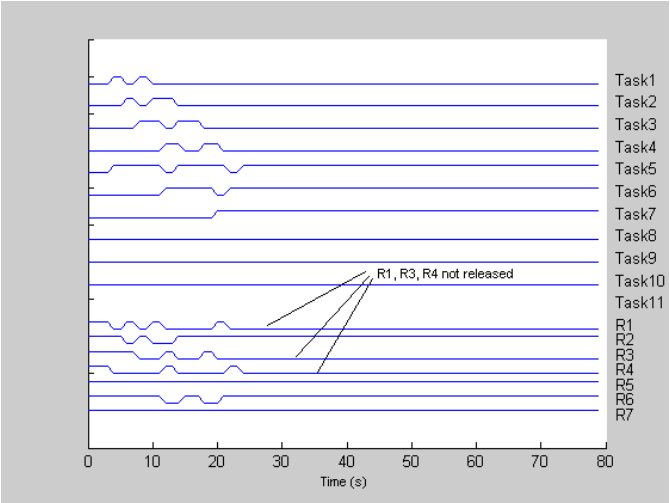


Figure 3.3 Event time trace resource assignment attempt 1: system is in deadlock

3.4.2 Resource Assignment- Attempt 2

The greedy algorithm makes a second attempt for a new resource assignment. The resource requirement matrix, the circular waits, critical subsystems and critical resources become:

$$F_r' = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$CW = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \quad J_o = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

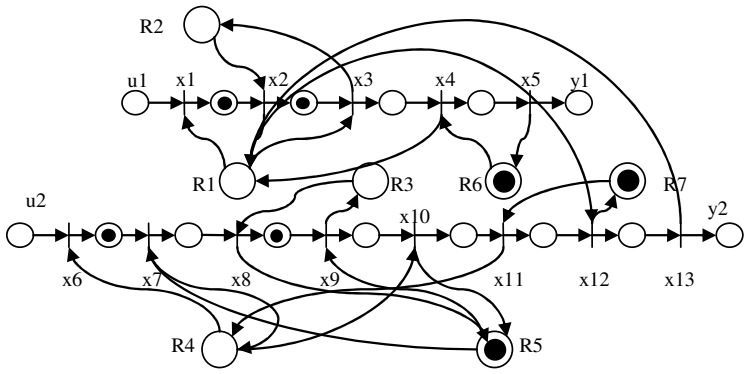


Figure 3.4 Petri net after second attempt of resource assignment: system is not regular

Analyzing vector *r* and matrix *CW*, it is possible to note that this assignment would result in a deadlock (compare with the corresponding Petri Net in figure 3.4 for tasks 1 and 2). Also, by applying the Gurel's test for regularity [32] to calculate the matrix of critical resources Res_{cw} for the new candidate system, it results that R_4 is a critical resource

$$\mathrm{Re}\, s_{cw} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The system is therefore irregular and a second order deadlock is present: the MAXWIP policy cannot be applied.

3.4.3 Resource Assignment- Attempt 3

Therefore the proposed resource assignment is discarded and the greedy algorithm comes up with a new assignment (see figure 3.5).

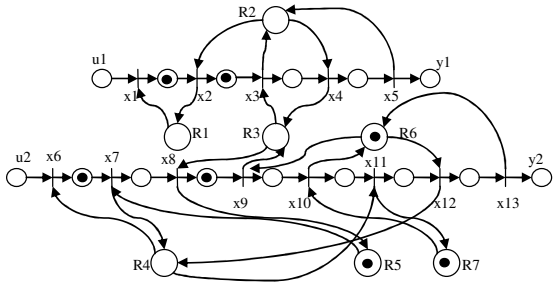


Figure 3.5 Resource assignment attempt 3: all requirements are met, the new configuration is accepted

$$F_r' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

The circular waits, critical subsystems and critical resources in this case are:

$$CW = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad J_o = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

$$\mathrm{Re}\, s_{cw} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The obtained system is regular (see Res_{cw} matrix) and does not present empty circular waits (compare vector r and matrix C_w). This configuration is accepted and a successful implementation of the tasks with the new resource assignment can finally take place (figure 3.6).

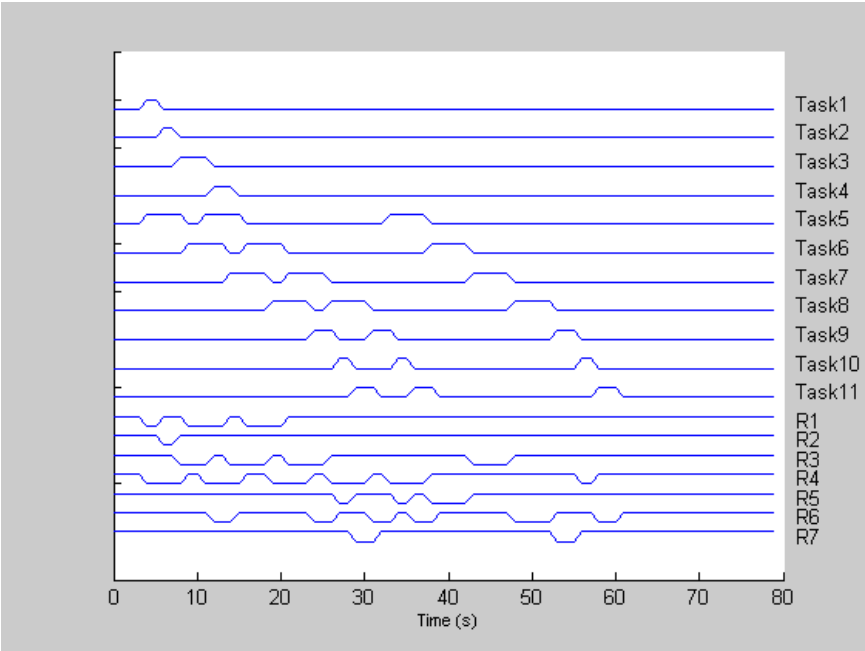


Figure 3.6 Event time trace resource assignment attempt 3: deadlock free dynamic resource assignment

A good application for this technique is monitoring of a warehouse using ground sensors and mobile robots [29]. Appropriate allocation of resources is critical in such applications.

3.5 Implementation Results

To show the effectiveness of the proposed strategy, the same scenario discussed in Section 3.4 is implemented at Automation and Robotics Research Institute (ARRI). The test-bed consists of four Acroname® Garcia robots which act as mobile resources while three Crossbow® Mica2 sensors act as stationary resources. The mobile robots are also endowed with Crossbow® sensors, so that we have a test-bed consisting of mobile and stationary sensors. Also, MIT cricket sensors (ultrasound ranging) are used for localization of the mobile resources. All the resources can communicate with each other for transmisson of data to and from the base-station. The greedy resource allocation is based on how close the resource is to the current task. Figure 3.7 shows the existing test-bed at ARRI.

Figure 3.7 Test-bed at Automation & Robotics Research Institute

When the number of tasks and resources are increased, the formulation of the required system matrices becomes cumbersome. The LabVIEW® toolkit developed at ARRI generates the matrices with a default conflict resolution matrix automatically when the resources are assigned to the tasks. The user can either manually change the resource allocation or the resource allocation can be changed automatically by the greedy resource assignment. Figure 3.8 and 3.9 show the front panel of the toolkit. The toolkit also generates the Petri Net for better visualization.

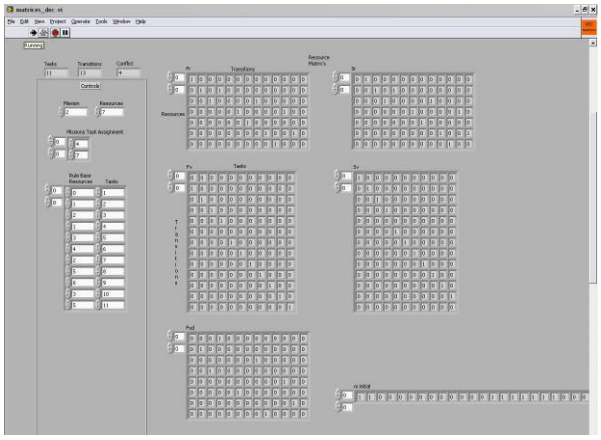


Figure 3.8 Automatic generation of matrices

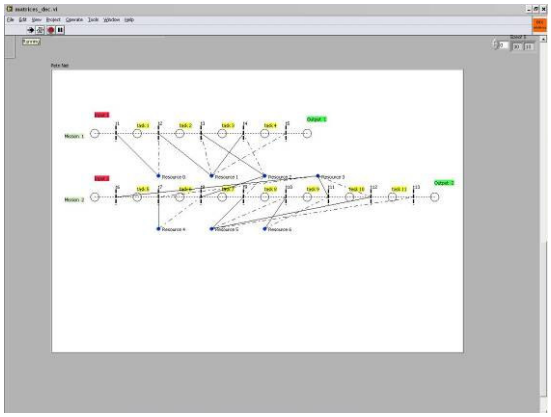


Figure 3.9 Automatic Petri-Net generation

Suppose the rule-base of the two missions discussed in section 3.4 is defined as shown in table 3.1.

Table 3.1 Rule-Base for the two missions in Section 3.4

TASK NO.	**TASKS/EVENTS**	**RESOURCES**	
Mission 1 INPUT	Event detected at Door1	First Attempt	Final Attempt
Task 1	Get Data from Door 1	*R1*	*R1*
Task 2	Robot 1 (R2) starts monitoring its round	*R2*	*R2*
Task 3	Get data from Machine 1	*R3*	*R3*
Task 4	Collect data from Door1 and send data back to Base station	*R6*	*R2*
Mission 1 OUTPUT	Base station receives data		
Mission 2 INPUT	Machine malfunction detected		
Task 6	Get data from Machine 3	*R4*	*R4*
Task 7	Get data from Machine 2	*R1*	*R5*
Task 8	Get data from Machine 1	*R3*	*R3*
Task 9	Get data from Machine 2 again	*R4*	*R6*
Task 10	Get Data from Door 2	*R6*	*R7*
Task 11	Get Data from Door 3	*R5*	*R4*
Task 12	Send all information back to the Base station	*R7*	*R6*
Mission 2 OUTPUT	Base station receives data		

Here *R2, R3, R4* and *R6* are mobile robots while *R1*, *R5* and *R7* are stationary sensors. As the robots change position while performing tasks, they become suitable for other tasks. Thus, based on the position of the resources with respect to the tasks, the greedy algorithm reassigns the resources.

As shown in figures 3.10 and 3.11, the highlighted tasks are the tasks currently being executed (tokens are in the tasks). Figure 3.2 shows the first attempt of resource assignment in table 3.1.

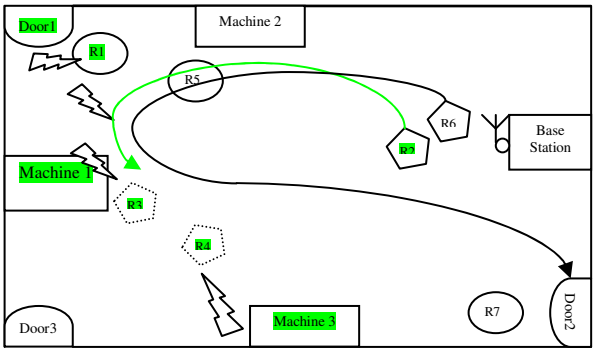


Figure 3.10 Initial Resource assignment

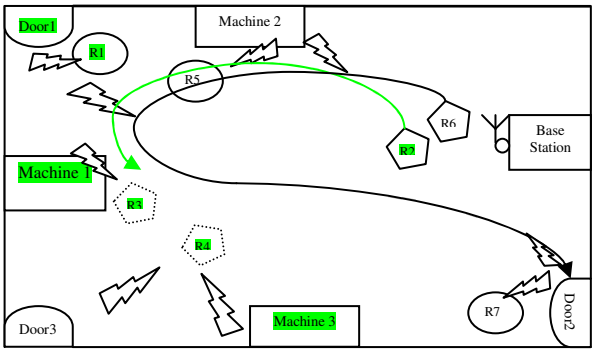


Figure 3.11 Final Resource assignment

Currently *R1, R2, R3* and *R4* are busy, and if the first attempt assignment shown in table 1 is made, deadlock will occur (see figures 3.2 and 3.3). As the robots change their locations, the assignment shown in figure 3.11 seems more appropriate. This assignment does not create any system deadlock. The advantage of the toolkit is that the resource assignment is done in run-time ensuring that the best set of resources is used for the tasks with no blocking phenomenon.

The next chapter considers deadlock avoidance in systems where routing of tasks are allowed.

CHAPTER 4

FREE-CHOICE MULTIPLE REENTRANT FLOW-LINES

4.1 Introduction

Resource assignment and task sequencing play important roles in applications involving mobile wireless sensor networks, manufacturing systems, and other decision resource systems. But, the use of shared resources in these discrete event systems (DES) creates major problems while sequencing tasks. If the assignment of the resources is not correctly made, serious problems might arise. Such problems include blocking and system deadlock [1, 2, 17, 18, 19, 20, 21, 22, 32, 39, 45, 46, 47, 49, 54, 56, 61, 62, 92, 93, 96], which are dangerous situations that eventually stop all the activity in the flow line involved.

In the past decade, considerable research has been done on developing deadlock control policies and routing in PN structures [4, 13, 32, 47, 50]. [97] provides an up-to-date survey on deadlock control policies used in PN. Vishwanadham *et al*. [89] suggested the use of PN for deadlock control. They define two main approaches for tackling the deadlock problem, namely, deadlock prevention (DP) [17, 35, 36, 44, 46] and deadlock avoidance (DA) [32, 47, 50, 54]. In DP methods, the system model is modified off-line so that the resulting controlled model is deadlock-free. DP approach makes use of adding extra control places to the original PN structure to prevent deadlock. DA algorithms check the system flow on-line to see that the system does not

fall in a deadlock state. DP and DA approaches require setting up a suitable control policy to regulate the shared resource allocation in the system. This chapter mainly focuses on the DA approach for a class of PN called FMRF. This is an online deadlock avoidance policy which does not require the addition of extra control places to the original PN structure.

Several existing approaches for DA in the literature are designed for the case of Multi Reentrant Flow Line systems (MRF) [1, 32, 47, 50, 87], where resources are shared and can perform more than one task. MRF systems are also known as S2LSPR (System of Simple Linear Sequential Processes with Resources) where flexible routing of tasks is restricted [54]. Analysis of deadlock avoidance and prevention is well understood for MRF. An important aspect in deadlock avoidance strategies in MRF is the concept of a Circular Wait (CW) [32] among the resources. It has been shown in [47, 92] that deadlock occurs in MRF when blocking develops in a CW.

The analysis of shared resources becomes even harder when choices are allowed for tasks. This means routing decisions have to be made [4, 13, 53]. These systems are a generalized case of MRF systems called the Free Choice Multi-Reentrant Flow Lines (FMRF). FMRF systems are also known as S3PR (System of Simple Sequential Processes with Resources). In the case of FMRF, the known methods of deadlock avoidance provided for MRF do not work. There are many DP methods for S3PR systems in the literature, but we are mainly concerned with DA policies (online deadlock control).

It should be noted that in Free-Choice PN, choices for tasks are allowed [2, 4, 13] whereas there are some other PN structures called Choice-Free PN where task choices are not allowed. Also, FMRF structure differs from PN structures such as ES3PR (Extended S3PR) [35, 36] or S3PGR2 (System of Simple Sequential Processes with General Resource Requirements) [54]. Such structures make use of more than one resource for a given task. Also, this work deals with deadlock avoidance in Free-Choice Petri Nets and not on the routing of resources employed in such systems. The routing mechanism is well studied in the literature and also features in the work of Park and Reveliotis [54], Lawley [42], etc. This work deals with deadlock avoidance and resource dispatching (not allocation) in systems where job path choices are allowed.

A matrix-based discrete event controller has been proposed, proving to be very efficient in computing PN objects and in sequencing tasks in manufacturing environments as well as sensor networks [1, 28, 29, 32, 47, 48, 50] for MRF systems. This chapter provides an extension of the matrix-based DA results of MRF systems to FMRF systems.

PN objects such as Circular Waits, Critical Siphons, and Critical Subsystems have been defined to avoid deadlock in MRF [32, 47, 50]. Due to the existence of decision place in FMRFs, which are followed by two or more decision branches, these objects cannot be used for deadlock avoidance there without redefinition.

This chapter shows how these objects can be computed for FMRF. The key issue is that it is necessary to redefine, or generalize the definition of, critical subsystems to FMRF. Also, we extend a matrix formulation which can efficiently

compute these objects for FMRF. This matrix formulation allows fast and efficient numerical computation techniques to be applied to PN analysis. We show how a MAXWIP dispatching policy can be formulated for FMRF to avoid blocking phenomena. Under this policy, deadlock in FMRF can be avoided by limiting the work in progress (WIP) in the Critical Subsystems of each CW. There is another type of deadlock called the second order deadlock [32, 47]. In this chapter we provide a regularity test to find key resources in the FMRF systems that cause second order deadlocks.

The rest of the chapter is organized as follows. In Section 4.2 we describe the properties that characterize FMRF/S3PR systems using Petri Nets. In Section 4.3 we show the correlation between circular waits and structures referred to as Critical Siphons and Critical Subsystems. We generalize the notion of critical siphon to the case of FMRF. Section 4.4 presents a matrix formulation over an or/and algebra that makes it efficient and direct to compute the Petri Net objects needed for deadlock avoidance. In Section 4.5, we illustrate the new notions by computing them for three FMRF/ S3PR examples. The first example admits deadlock while the second does not. The third example is an FMRF system that contains a key resource.

4.2 Petri Net Analysis of FMRF Systems

4.2.1 Definition of FMRF Systems

More general than MRF are the free-choice multiple reentrant flow-line systems (FMRF), which have no predetermined resource allocation for the jobs. That is, several

different resources may be capable and available to perform a specific job. Then, routing decisions may be required along the part path about which resources to use for the next job, see figure 3.1.

We formally define FMRF systems as a class of systems satisfying the following properties (ϕ being the empty set):

Properties of FMRF

1. $p \in P, \bullet p \cap p \bullet = \phi$

2. on part path j, $x_{j1} \bullet \cap P \setminus J = \phi$ and $\bullet x_{jLj} \cap P \setminus J = \phi$

3. $\forall p \in J, \quad \bullet\bullet p \cap R = p \bullet\bullet \cap R = R(p)$ with $|R(p)| = 1$

4. $p \in J, \quad |p \bullet| \geq 1$

5. $\forall p_i, \quad p_k \in J_j, \quad i \neq k, \quad p_i \neq p_k$

6. $\forall p_i \in J_j, \quad p_k \in J_l, \quad j \neq l, \quad p_{ji} \bullet \cap p_{lk} \bullet = \phi$

7. $R_s \neq \phi$

This means that there are: (1) no self loops, (2) each part path has a well-defined beginning and an end, (3) every job requires only one resource with no two consecutive jobs using the same resource, (4) there may be some jobs that can be done by different resources (i.e. routing decisions may have to be made), (5) there are no part path loops, (6) for any two distinct jobs on different part paths there is no assembly, i.e. two part paths cannot merge into one, (7) there are shared resources.

According to property 3, $R(p) = \bullet\bullet p \cap R = p \bullet\bullet \cap R$, with the cardinality $|R(p)| = 1$; Under the foregoing assumption, one has $J(r) = r \bullet\bullet \cap J = \bullet\bullet r \cap J$.

Property 4 distinguishes MRF from FMRF systems. A FMRF system can have $|p\bullet|>1$ for some $p\in J$, so that routing decisions are needed. We call such job places 'decision places'. In MRF, one has $|p\bullet|=1 \ \forall p\in J$. That is, MRF are a special class of FMRF. A transition $x\in p\bullet$ is said to be a posterior transition of *p*. A decision place has multiple posterior transitions, i.e. $|p\bullet|>1$. The resources used by decision places are called decision resources.

Figure 4.1 shows a sample FMRF. In figure 4.1 the part paths split at decision places. The decision places are B1 and B2, each followed by two transitions. A choice is required there to decide which resource (R1 or R2, respectively R3 or R4) to use for the next job.

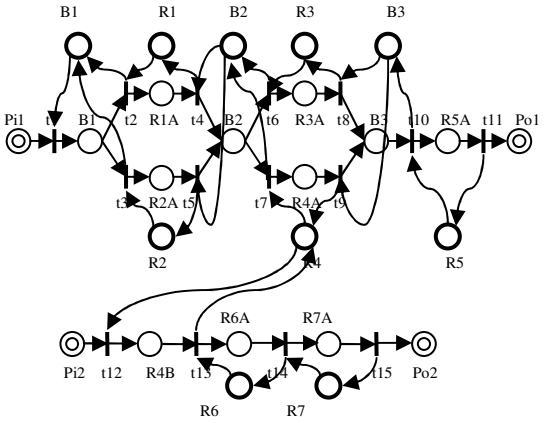


Figure 4.1 FMRF system

4.2.2 Circular Waits in FMRF Systems

In chapter 2 we defined $J(C)_+$ and $J(C)_0$. Note that for FMRF, one may have $J(C)_+ \cap J(C)_0 \neq \phi$. In fact, if $p\in J(C)$ is a decision place with *C* a simple

CW, i.e. $|p\bullet| > 1$ so that it has more than one posterior transition, then, $p \in J(C)_+$ and $p \in J(C)_0$. This is precisely what distinguishes deadlock avoidance in MRF from deadlock avoidance in FMRF systems.

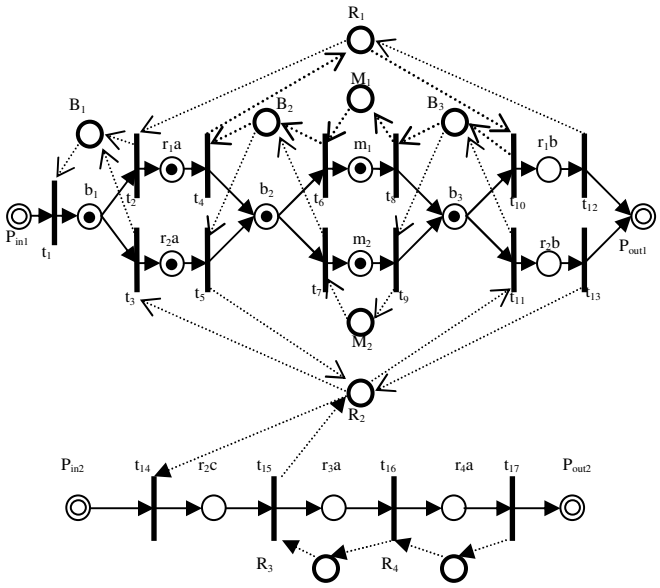


Figure 4.2 Sample FMRF system

Figure 4.2 shows a FMRF system. Here, *R1-B2-M1-B3* is a *CW* C. Here, *r1a*, *m1*, *b2* and *b3* are in $J(C)_0$ and *r1b* is in $J(C)_+$. *b2* and *b3* are decision places and *B2* and *B3* are the respective decision resources. In FMRF systems, due to their construction, the decision places (*b2* and *b3* in this example) are also a part of $J(C)_+$.

4.2.3 Siphons in FMRF Systems

The analysis of CB and deadlock can be carried out formally using the notion of siphon. A siphon is a set of places having the property that its input transition set is contained in its output transition set i.e.

$$\bullet S \subset S \bullet \tag{4.1}$$

A siphon has the key property that, once it is unmarked, it remains so.

A minimal siphon of a CW *C* is the smallest siphon containing the CW. Define a critical siphon for a CW *C* as a smallest siphon which has the property that a CW is a CB if and only if the critical siphon is empty. It is shown in [32, 47] that a minimal siphon in MRF is a critical siphon.

For MRF systems, for every CW *C*, the set of places $\hat{S}_c$ defined by

$$\hat{S}_c = C \cup J(C)_+ \tag{4.2}$$

is a minimal siphon as well as a critical siphon, where $J(C)_+$ was defined earlier [32, 47].

In the case of FMRF systems, this object is still a minimal siphon, but it cannot be used in deadlock analysis due to decision places. Hence it is not a critical siphon as is now shown.

Lemma 4.1: For FMRF systems, $\hat{S}_c$ is a minimal siphon for the CW *C*.

Proof: To prove that $\hat{S}_c$ is a siphon, we need to show $\bullet \hat{S}_c \subseteq \hat{S}_c \bullet$, i.e., every transition having an output place in $\hat{S}_c$ has an input place in $\hat{S}_c$. It is known that, for every $r_i \in C$, there exists $r_j \in C, i \neq j$, such that $\bullet r_i \cap r_j \bullet \neq \phi$. So, if, $r_i \in C \cap R_{ns} \cap \bar{B}_r$ with B_r being the decision resources, then $\left| \bullet r_i \right| = 1$, and there exists some $r_j \in C, i \neq j$, such that $\bullet r_i \in \{r_j \bullet\}$. On the other hand, $r_i \in C \cap (R_s \cup B_r)$, then for every $x \in \{\bullet r_i\}$, either

$x \in \{r_j \bullet\}$, for some $r_j \in C, i \neq j$, or $x \in p \bullet$, for some $p \in J(C)_+$. Moreover, $\forall p \in J(C)_+, \bullet p \in \{r \bullet\}$ for some $r \in C$ by Property 3. This proves that $\bullet \hat{S}_c \subseteq \hat{S}_c \bullet$, therefore by construction, S_c is a siphon.

The minimality of $\hat{S}_c$ is obvious from the way it is constructed in FMRF systems. If any $p \in J(C)_+$ is left out, then there will be some $r_k \in C \cap R_s$ with $\{\bullet r_k\} \not\subset \hat{S}_c \bullet$. Similarly if one of the resources in C is not considered, $r_i \in C$ is left out, then there will be at least one $r_j \in C$ with $\{\bullet r_j\} \not\subset \hat{S}_c \bullet$. Both the cases violate the definition of siphon.■

Lemma 4.2: Let C be a simple CW and $\hat{S}_c$ contain a decision place whose associated decision resource is not a shared resource, then $\hat{S}_c$ contains a p-invariant and can never be empty.

Proof: Let C be a simple CW and $\hat{S}_c$ be defined by (4.2). Suppose $\hat{S}_c$ contains a decision place p with an associated resource place r_p that is not shared, i.e. $J(r_p) = p$. Then $C \cap \bullet \bullet r_p = 1$ and $|p \bullet| > 1$, so that $p \in J(C)_0$ and $p \in J(C)_+$. Therefore, $p \in \hat{S}_c$. Moreover, $r_p \in \hat{S}_c$ so that the p-invariant $r_p \cup J(r_p)$ is in $\hat{S}_c$. ■

4.3 Critical Siphons & Deadlock Avoidance for FMRF Systems

Lemma 4.1 shows that $\hat{S}_c$ given by (4.2) is indeed a minimal siphon, even for FMRF, but Lemma 4.2 shows that in some situations for FMRF, $\hat{S}_c$ can never be

empty. Since deadlock can still occur in such situations, it is necessary to provide an alternative formula for critical siphons for FMRF.

4.3.1 FMRF Critical Siphons

Let C_s be the set of all simple CWs in a FMRF system. Let p be a decision place with resource place r_p=$R(p)$. Define the set of all the simple CWs in C_s containing the decision resource r_p as

$$C_p = \{C \in C_s \mid r_p \in C\} \tag{4.3}$$

Given a CW C, let the set of its decision places be

$$J_{dec}(C) = \{p \in J(C) \mid |p\bullet| > 1\} \tag{4.4}$$

Define,

$$C_p(C) = \{C_p \mid p \in J_{dec}(C)\} \tag{4.5}$$

which is the key to deadlock analysis in FMRF systems.

We need to recursively compute $C_p(C)$, as each individual simple CW in $C_p(C)$ defined in (4.5) may contain more decision resources, and so on as shown in figure 4.3. Hence we need an algorithm to calculate $C_p(C)$. Algorithm 1 shows a modified version of the greedy activity selector algorithm [9]. Define η_c as the number of CW in the system. A method for determining all the CWs in a system is given in Section 4.4.

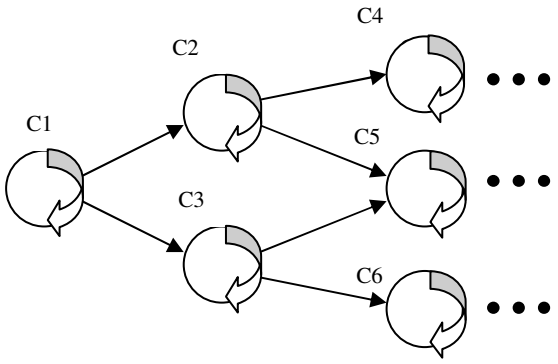


Figure 4.3 Representation of simple CW dependence through decision places

Algorithm 4.1: Compute $C_p(C)$ for a CW C

Given a CW C,

1. Calculate $C_p(C)$ using Equation 8

2. $x = 1$

3. $y = \eta_c$

4. For x ← 1: y

If any $r_p \in (C_x \subset C_p(C)) \notin C_p(C) \setminus C_x$ (for each decision resource in the system)

$C_p(C) = [C_p(C) \cup C_p(C_x)]$

5. Return $C_p(C)$ ■

Note that this algorithm computes the set $C_p(C)$ for a single CW. The time complexity of this algorithm is known to be linear [9] as the running time is directly proportional to η_c. Hence the complexity from step 4 to 5 is $O(\eta_c)$. The complexity for computing $C_p(C)$ for all the CWs in the system is $O(\eta_c{}^2)$

Also, when this algorithm terminates, $C_p(C)=C_p(C_p(C))$. This algorithm terminates in one of the two ways:

a) $J(C_p(C))_+ \cap J(C_p(C))_0 = \phi$ or

b) $J(C_p(C))_+ \cap J(C_p(C))_0 \neq \phi$.

In the discussions ahead, we will prove that in case (b), $C_p(C)$ can never be in CB and hence no CW in $C_p(C)$ can be in CB.

Define a relation $C \sim \tilde{C}$, if there exists a resource $r \in C \cap \tilde{C}$. Note that "~"satisfies the following properties:

1. $\tilde{C} \sim \tilde{C}$ (Reflexivity)

2. If $C \sim \tilde{C}$ then $\tilde{C} \sim C$ (Symmetry)

3. If $C \sim \tilde{C}$ and $\tilde{C} \sim \bar{C}$ then it is not guaranteed that

   $C \sim \bar{C}$ (Not Transitive)

Therefore, "~" is not an equivalence relation. Define a second relation $C \approx \tilde{C}$ if there exists a set of CW C_i such that $C \sim C_1 \sim C_2 ... \sim C_n \sim \tilde{C}$. Note that,

1. $C \sim C$ (Reflexivity)

2. If $C \sim \bar{C}$ then $\bar{C} \sim C$ (Symmetry)

3. If $C \approx \tilde{C}$ then $\tilde{C} \approx C$ (Transitive)

Therefore the relation "≈" is an equivalence relation [33] and partitions the set of all the CW into disjoint equivalence classes $K(C) = \{ \bar{C} : \bar{C} \approx C \}$, i.e. $K(C)=K(\bar{C})$.

Corollary 4.1: For any simple CW $\bar{C} \in C_p(C)$, $C_p(C) = C_p(\bar{C})$.

Proof: The algorithm merely computes the set $C_p(C)=K(C)$. Therefore, $C_p(C) = C_p(\bar{C})$. ■

Lemma 4.3: $C_p(C)$ is a CB if and only if:

1. $J(C_p(C))_+ \cap J(C_p(C))_0 = \phi$
2. $m(C_p(C)) = 0$ and
3. for each $r_i \in C_p(C), \forall p \in J(r_i)$ with $m(p) \neq 0$, $p \in J(C_p(C))_0$

Proof: Necessity: Let $C_p(C)$ be in CB i.e it is empty. This means that $m(C_p(C))=0$. Let $p \in J(C_p(C))$ and suppose $p \in J(C_p(C))_+$. Then either $p\bullet \cap r\bullet = \phi$ or there exists a resource $r \notin C_p(C)$ such that $p\bullet \cap r\bullet \neq \phi$. This means that a transition $t \in p\bullet$ may fire and put a token in $R(p)$ so that $C_p(C)$ does not remain empty. Therefore, all the marked jobs are in $J(C_p(C))_0$ and not in $J(C_p(C))_+$ and conditions 1 and 3 hold.

Sufficiency: Conditions 1 and 3 of Lemma 4.3 imply that any $r_i \in C_p(C)$ can get a token if and only if some $r_j \in C_p(C)$ with $j \neq i$, can get a token. However by condition 2 of Lemma 4.3, all the R-places in $C_p(C)$ are empty, and hence none of them can ever get any token. In other words, $C_p(C)$ is in CB. ■

Lemma 4.4: For MRF, C is in CB if and only if:

1. $m(C_p(C)) = 0$ and

2. for each $r_i \in C_p(C), \forall p \in J(r_i)$ with $m(p) \neq 0$, $p \in J(C_p(C))_0$

Proof: In MRF, $C_p(C) = C$ and condition 1 of Lemma 3 always holds. ■

Given a CW C define the object,

$$S_c = C_p(C) \cup J(C_p(C))_+ \tag{4.6}$$

The next results show that S_c is a critical siphon for C for FMRF systems under a certain condition.

Lemma 4.5: For any CW C, S_c defined in (4.6) is a minimal siphon for $C_p(C)$.

Proof: One needs to show that $\bullet S_c \subseteq S_c \bullet$, i.e., every transition having an output place in S_c has an input place in S_c. By construction, $\bullet S_c \subset \bullet C_p(C)$, i.e., the output places for these transitions are resources in $C_p(C)$. It is known that, for every $r_i \in C_p(C)$, there exists $r_j \in C_p(C), i \neq j$, such that $\bullet r_i \cap r_j \bullet \neq \phi$. So, if $r_i \in C_p(C) \cap R_{ns}$, there exists some $r_j \in C_p(C), i \neq j$, such that $\bullet r_i \in \{r_j \bullet\}$. On the other hand, $r_i \in C_p(C) \cap R_s$, then for every $x \in \{\bullet r_i\}$, either $x \in \{r_j \bullet\}$, for some $r_j \in C_p(C), i \neq j$, or $x \in p \bullet$, for some $p \in J(C_p(C)) +$. Moreover, $\forall p \in J(C_p(C))_+, \bullet p \in \{r \bullet\}$ for some $r \in C_p(C)$ by Property 3. This proves that $\bullet S_c \subseteq S_c \bullet$, therefore by construction, S_c is a siphon. Minimality follows by the way S_c is constructed. ■

Note, for MRF, $C_p(C) = C$, $S_c = \hat{S}_c$ and Lemma 4.1 is recovered.

Lemma 4.6: If $J(C_p(C))_+ \cap J(C_p(C))_0 \neq \phi$, then S_c contains a p-invariant and can never be empty.

Proof: If $J(C_p(C))_+ \cap J(C_p(C))_0 \neq \phi$, it means that there exists a decision place p such that $p \in J(C_p(C))_+$ and $p \in J(C_p(C))_0$. Also, by definition of S_c, $p \in S_c$. If r_p is the corresponding resource of p, $r_p \in S_c$, so the p-invariant $r_p \cup J(r_p)$ is also in S_c. Hence it can never be empty. ■

The next result shows that S_c is a critical siphon for $C_p(C)$.

Theorem 4.1: Given $C_p(C)$ and let $J(C_p(C))_+ \cap J(C_p(C))_0 = \phi$. Then $C_p(C)$ is in CB if and only if S_c is empty.

Proof. Necessity: If $C_p(C)$ is in CB, Lemma 4.3 shows that $m(C_p(C))=0$ and $m(p) \neq 0$ only for $p \in J(C_p(C))_0$. Suppose S_c is not empty. Then there is a place p' such that $p' \in J(C_p(C))_+$ and $m(p') \neq 0$. This is a contradiction.

Sufficiency: S_c is a siphon and by definition, once it is empty, it will remain so. Thus, when $m(C_p(C))=0$, with all the tokens lost to some $p \in J(r_i)$, with $m(p) \neq 0$ means that $p \notin J(C_p(C))_+$ and hence, $p \in J(C_p(C))_0$. Since $J(C_p(C))_+ \cap J(C_p(C))_0 = \phi$, by Lemma 4.3 $C_p(C)$ is in CB. ■

Theorem 4.2: Let $J(C_p(C))_+ \cap J(C_p(C))_0 \neq \phi$. Then $C_p(C)$ can never be in CB.

Proof: For $C_p(C)$ to be in CB, $m(C_p(C))=0$ and $m(p) \neq 0$ only with $p \in J(C_p(C))_0$. Suppose also $p \in J(C_p(C))_+$. This implies $\bullet(p\bullet) \cap R \notin C_p(C)$. Thus, this token can be fired. Hence, deadlock does not occur. ■

There exists a relationship between Lemma 4.6 and Theorem 4.2. When $J(C_p(C))_+ \cap J(C_p(C))_0 \neq \phi$, it means S_c has a p-invariant and hence, $C_p(C)$ can never be in CB.

Lemma 4.7: Given a CW $C=\{r_i\}$. Then

(i) $J(C)_0 \subset J(C_p(C))_0$

(ii)If $J(C_p(C))_+ \cap J(C_p(C))_0 = \phi$, then $J(C)_+ \setminus J(C)_0 \subset J(C_p(C))_+$

Proof: (i) Given $C=\{r_i\}$. Then by definition $C \subset C_p(C)$. Now, let there be place p' such that $p' \in J(C)_0$ and $p' \notin J(C_p(C))_0$. In this case, $\bullet(p'\bullet) \cap C \neq \phi$ and $\bullet(p'\bullet) \cap C_p(C) = \phi$. This means that $C \not\subset C_p(C)$, which is a contradiction. Hence,

$J(C)_0 = \{p \in J(C) \mid p\bullet \in \bullet r_{io}, r_i \in C\} \subset \{p \in J(C_p(C) \mid p \in r_{j0}, r_j \in C_p(C)\}$
$\subset J(C_p(C))_0$

(ii) In FMRF, if C is a CW with decision place p then, $|p\bullet| > 1$, $p \in J(C)_0$ and also possibly $p \in J(C)_+$ as stated earlier. Hence the set $\{J(C)_+ \setminus J(C)_0\}$ strictly consists of job places belonging to $J(C)_+$ without decision places. Let $\{J(C)_+ \setminus J(C)_0\}$ be denoted by $J(C)_{s+}$. Now by definition, $J(C) = J(C)_+ \cup J(C)_0$ and $J(C_p(C))=J(C_p(C))_+ \cup J(C_p(C))_0$. By (i), $J(C)_0 \subset J(C_p(C))_0$. Since $C \subset C_p(C)$, $J(C) \subset J(C_p(C))$. Therefore, $J(C)\setminus J(C)_0 \subset J(C_p(C))\setminus J(C_p(C))_0$. The term $J(C)\setminus J(C)_0$ is equal to $J(C)_+\setminus J(C)_0$ i.e. $J(C)_{s+}$. Also, the term $J(C_p(C))\setminus J(C_p(C))_0$ is $J(C_p(C))_+$ as $J(C_p(C))_+ \cap J(C_p(C))_0 = \phi$. Thus, $J(C)_+ \setminus J(C)_0 \subset J(C_p(C))_+$. ■

The next results show that S_c is a critical siphon for C.

Theorem 4.3: If $J(C_p(C))_+ \cap J(C_p(C))_0 = \phi$ then CW C is in CB if and only if $C_p(C)$ is in CB.

Proof: Necessity: Suppose C be in CB and $C_p(C)$ is not in CB.

Case 1: If C does not contain decision places then $C=C_p(C)$ and the result follows.

Case 2: If C is in CB, then $m(C)=0$. Consider a resource $r \in C$ and its job place p such that r is dead i.e. $m(r)=0$ for all future firings. Since r is dead, $p\bullet \in \bullet r$ and hence, $\bullet(p\bullet) \cap R \in \bullet\bullet r \cap R$ must all be dead for C to be in CB. $C_p(C)$ and C are related to each other by their decision resources. So, iterating this process in a PN from r backwards to a path of transitions and R-places which include all resources of $C_p(C)$, must be dead. For C to remain in CB, no token should be added to C and hence no token should be added to $C_p(C)$. Thus, all the jobs in $C_p(C)$ must be in $J(C_p(C))_0$. The condition $J(C_p(C))_+ \cap J(C_p(C))_0 = \phi$ ensures that when $C_p(C)$ is in CB, all the tokens are in $J(C_p(C))_0$ and not $J(Cp(C))_+$. Hence C is in CB.

Sufficiency: Let $C_p(C)$ be in CB. Then $m(C_p(C))=0$ and hence $m(C)=0$. Also by Lemma 4.3, all marked jobs are in $J(C_p(C))_0$. and not in $J(C_p(C))_+$ and $J(C_p(C))_0 \cap J(C_p(C))_+ = \phi$. By Lemma 4.7 $J(C)_0 \subset J(C_p(C))_0$ and $J(C)_{s+} \subset J(C_p(C))_+$ where $J(C)_{s+} = \{J(C)_+ \setminus J(C)_0\}$. Now, $J(C)_{s+} \cap J(C_p(C))_0 = \phi$. Hence for all resources r_i in C and $p \in J(r_i)$ having $m(p) \neq 0$, $p \in J(C)_0$. Therefore, $r_i \in C$ can get a token if and only if some other $r_j \in C_p(C)$ has a token. Hence C is in CB. ■

Corollary 4.2: Given a CW C, and suppose $J(C_p(C))_0 \cap J(C_p(C))_+ = \phi$, then C is in CB if and only if S_c is empty.

Proof: By Theorem 4.3, C is in CB if and only if $C_p(C)$ is in CB. Then Theorem 1 completes the proof. ■

Corollary 4.3: Given $J(C_p(C))_0 \cap J(C_p(C))_+ = \phi$, $C_p(C)$ is in CB if and only if all the simple CWs in $C_p(C)$ are in CB.

Proof: Necessity: Suppose $C_p(C)$ is in CB. Let there be a simple CW $C_1 \subset C_p(C)$ such that it is not in CB. It means either $m(C_1) \neq 0$ or token can be added to C_1. Since $C_1 \subset C_p(C)$, $m(C_p(C)) \neq 0$ or tokens can be added to $C_p(C)$. Hence, $C_p(C)$ is no longer in CB.

Sufficiency: If all the simple CWs in $C_p(C)$ are in CB, it means $m(C_i)=0$ and no tokens will ever be added to C_i. Since $C_p(C)=\bigcup C_i$, $m(C_p(C))=0$ and $m(J(C_p(C)_0)) \neq \phi$. Hence by definition of CB, $C_p(C)$ is in CB, provided $J(C_p(C))_0 \cap J(C_p(C))_+ = \phi$. ■

Corollary 4.4: Given a CW C, and suppose $J(C_p(C))_0 \cap J(C_p(C))_+ = \phi$, then any CW $\bar{C} \in C_p(C)$ is in CB if C is in CB.

Proof: By Theorem 4.3, C is in CB if and only if $C_p(C)$ is in CB. Then Corollary 4.3 completes the proof. ■

4.3.2 FMRF Critical Subsystems

For better perspective on achieving dispatching policies with deadlock avoidance, the concept of critical subsystems [32] is introduced. To avoid deadlock,

work in progress (WIP) must be limited within certain critical subsystems that are constructed by using critical siphons, called MAXWIP.

A critical subsystem for a CW C in MRF is defined as the set of J-places $J(C)_0$. In case of FMRF, Critical Subsystem for a CW C can be defined if $J(C_p(C))_0 \cap J(C_p(C))_+ = \phi$ and in this case it is defined as the set of J-places $J(C_p(C))_0$. Define the set $S_c^* = S_c \cup J(C_p(C))_0$ called the support of the binary p-invariant that minimally covers S_c.

Define $T_{pre}(C) = {\bullet J(C)_0} \setminus J(C)_0 {\bullet}$ as the precedent transitions of CW C and $T_{pos}(C) = J(C)_0 {\bullet} \setminus {\bullet J(C)_0}$ as the posterior transitions of CW C.

Definition 4.1: *Key Resources:* Let $\{C_i, C_j\}$ be two simple CWs such that $|C_i \cap C_j| = 1$ and define $C_i \cap C_j = r_k$. If $T_{pos}(C_i) \cap T_{pre}(C_j) \subset r_k \bullet$ and $T_{pos}(C_j) \cap T_{pre}(C_i) \subset r_k \bullet$, then $\{C_i, C_j\}$ is a cyclic CW (*CCW*). If in addition, $m(r_k)=1$, then r_k is called a key resource. A system with key resources is called an irregular system.

Vishwanadham *et al.* [89] showed that predictive simulation in PN can be used to avoid deadlocks. We formalize this by defining critical subsystems, which are the objects that must be examined in predictive simulation to avoid deadlock. Critical subsystems must be examined for WIP content in the look-ahead deadlock avoidance methods. To avoid deadlock in systems without key resources, 1-step look ahead is good enough [47]. If key resources are present, one must look ahead 2 steps to avoid deadlock [47]. Key resources are also related to the siphon depth variable ξ in [44]. For

regular FMRF systems, a depth variable ξ=1 is good enough. If key resources are present, one requires ξ>1.

The marking condition $m(S_c^*)$ can be given in terms of the initial marking of $C_p(C)$ *i.e* $m(C_p(C))$, since for FMRF, the initial marking assigns tokens only to R and PI places.

Lemma 4.8: In any FMRF system, the critical siphons S_c is empty if the critical subsystems $J(C_p(C))_0$ are full.

Proof: Since $S_c^* = S_c \cup J(C_p(C))_0$ is the support of the binary p-invariant, the total number of its tokens is conserved. Thus, S_c will be empty if and only if all the tokens are in $J(C_p(C))_0$ i.e. the critical subsystem. ■

Lemma 4.9: In a regular FMRF system, the critical siphons S_c are never empty if and only if the critical subsystems $J(C_p(C))_0$ are not full (i.e. $m(J(C_p(C)_0) \leq m(C_p(C))$-1) at each step.

Proof: Necessity: Suppose S_c is not empty. If $J(C_p(C)_0)$ is full, Lemma 4.8 proves that S_c has to be empty. This is a contradiction. Hence, $J(C_p(C)_0)$ is never full.

Sufficiency: Suppose $J(C_p(C)_0)$ is not full in a regular FMRF system and let the $m(J(C_p(C)_0)=m(C_p(C))$-1. This restriction ensures that at least one resource in $C_p(C)$ will have a token which will not be used for $J(C_p(C))_0$ but will be used for $J(C_p(C))_+$ such that S_c will never be empty. If S_c still remains empty, it means that there exists two CWs C_i, $C_j \in C_p(C)$ such that $C_i \cap C_j = r_k$ and $m(r_k)$=1. Let there be two transitions t_1, t_2 such that $t_1 \in T_{pre}(C_i) \cap T_{pos}(C_j)$ and $t_2 \in T_{pre}(C_j) \cap T_{pos}(C_i)$. $t_1 \bullet$ and $t_2 \bullet$ use the same

resource r_k. By construction, $t_1 \bullet \in J(C_i)_0 \cap J(C_j)_+$ and $t_2 \bullet \in J(C_i)_+ \cap J(C_j)_0$. Hence, at the next time step, if a token is present in $t_1 \bullet$, then $t_2 \bullet$ will always remain empty and vice versa even if the condition $m(J(C_p(C)_0)=m(C_p(C))$-1 holds true, since $m(r_k)$ becomes zero. Hence, S_c will remain empty. This is a definition of an irregular system and r_k is a key resource. For regular systems, S_c will never be empty if the restriction $m(J(C_p(C)_0) \leq m(C_p(C))$-1 is satisfied. ■

Note that one can now avoid deadlock of a simple CW C by ensuring that the marking of its critical subsystem S_c^* is less than the total marking of the initial resources in $C_p(C)$. To be able to analyze FMRF systems (and for MRF systems) and all its possible deadlock structures, we need to identify the set of all CWs i.e simple CWs and CWs composed of union of non-disjoint simple CWs (unions through shared resources among simple CWs) [13, 25]. Let us denote this set as CW_r.

Lemma 4.10: There is no CB in the FMRF system if for every $C \in CW_r$, the set $C_p(C)$ is not in CB.

Proof: By Theorem 4.1, when $C_p(C)$ is not in CB, it implies that S_c is not empty. Then Theorem 4.1 and Lemma 4.10 complete the proof. ■

Lemma 4.11: In a regular FMRF, there is no CB in the system if and only if for every $C \in CW_r$, one has $m(J(C_p(C))_0$ one less than the initial marking of $C_p(C)$. ■

If key resources are present, one can limit the WIP in the critical subsystems to $m(C_p(C))$-2 instead of looking ahead 2 steps. This is overly restrictive, but not by much.

These results show that we can avoid deadlock by MAXWIP, i.e. limiting WIP in all the critical subsystems.

4.4 Matrix Computation of Petri Net Objects in FMRF Systems

PN provides great pictorial insight and mathematical techniques for analysis, but they have sometimes had the deficiency of not providing an efficient computational framework for simple computer-based analysis. A matrix framework for computing structural objects of a PN can correct these deficiencies [47].

4.4.1 Circular Waits in Matrix Form

A wait relation digraph for MRF [36, 39] is defined as

$$W_r = (S_r \otimes F_r)^T \tag{4.7}$$

Each 'one' in the elements w_{ij} of W_r, represents that the digraph has an arc from resource i to resource j. CWs appear as loops in this digraph. Simple CW appear as simple loops e.g. not containing any smaller loops. W_r is used to compute all the simple CWs in MRF systems using a binary string algebra approach used by [50] which gives an output matrix CW_r^*. Each CW is represented as a row in the matrix CW_r^*. In this matrix CW_r^* each entry of 'one' in position (i,j) means that each resource j is included in the i^{th} simple CW.

However, due to the complexity of the Free-Choice extension of the MRF systems, and due to the diversity of loop paths that a set of resources contained in a simple CW might have, we need to identify not only the resources that compose each simple CW, but also the transitions that link them. This will give us specific information needed to locate siphons needed for constructions of our deadlock policy for FMRF systems. We define W_t (by duality of W_r) as

$$W_t = (F_r \otimes S_r)^T \tag{4.8}$$

This is a digraph of transitions. That is, W_t is a digraph having arcs from transition t_i to transition t_j (by 'bypassing' a resource $\bullet t_j \cap t_i \bullet$.). Each 'one' in the elements w_{ij} of W_t, represents that the digraph has an arc from transition *i* to transition *j*. Then, one can identify loops among transitions by using string algebra as above which gives the output matrix CW_t^*. We loosely say *t* is in CW if $t \in CW_t^*$.

But, even if we calculate two outcomes from the string algebra, transition loops and resource loops, we will not be able to identify which set of transition loops correspond to which set of resource loops (due to the behavior of the algorithm.). To do this, define,

$$W = \begin{bmatrix} O_r & S_r \\ F_r & O_t \end{bmatrix} \tag{4.9}$$

where O_r is a zero-matrix having *n*x*n* elements, O_t is a zero-matrix having *m*x*m* elements, *n* be the number of resources or rows (column) of S_r (F_r), and *m* be the number of transitions or rows (columns) of F_r (S_r). This is a digraph of transitions *T* and resources *R*.

For example, this digraph can be obtained by erasing all the jobs places from Figure 4.2, and keeping the resources, the transitions, and all the links between them, as shown in figure 4.4.

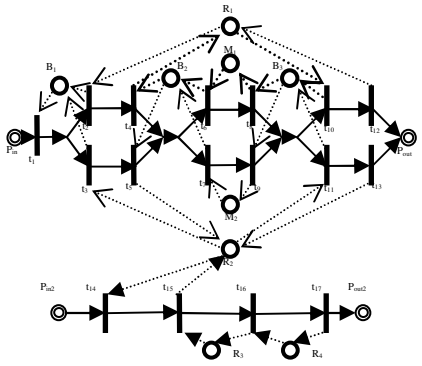


Figure 4.4 Graphical representation of digraph matrix *W*

Note that, using this digraph matrix *W* with the binary string algebra algorithm given in [18, 28], we get the set $C_w = [CW_r^* \; CW_t^*]$, where CW_r^* is the set of simple CW of resources and CW_t^* is a set of simple CW of transitions.

In order to find the complete set of simple CWs and the union of non-disjoint simple CWs, we use the Gurel algorithm [32, 50] which gives a matrix *G*. *G* provides the set of composed CWs (rows) from unions of simple CWs (columns) i.e. an entry of '1' in every (*i, j*) position implies j^{th} simple CW is included in the i^{th} composed CW. Then, we can calculate the set of loop resources CW_r and loop transitions CW_t using the following constructions,

$$CW_r = G^T \otimes CW_r^* \tag{4.10}$$

$$CW_t = G^T \otimes CW_t^* \tag{4.11}$$

Note that * denotes a simple CW, while CW_r and CW_t refer to all the CWs (i.e. simple and their unions).

4.4.2 Matrix Algorithm for Computing $C_p(C)$

Property 4 of FMRF, i.e $p \in J$, $|p\bullet| \geq 1$, is what differentiates FMRF from MRF systems. We apply this property of FMRF to compute $C_p(C)$ using matrices. To do so, we must first compute C_p and $J_{dec}(C)$ for each CW as required to run Algorithm 4.1. The next algorithm shows a routine which can easily be implemented using tools such as MATLAB®, to compute C_p and $J_{dec}(C)$.

Algorithm 4.2:

Calculating C_p using matrices

for i=1:|(CW_r)|

if $max(CW_r(i,:)*S_r*F_v)>1$ [(i,:) implies entire i^{th} row]

$C_p(i,:)=CW_r(i,:)$

end

end

Calculating $J_{dec}(C)$

for i=1:|(CWr)|

if $max(CW_r(i,:)*S_r*F_v)>1$

$J_{dec}(i,:)=CW_r(i,:)*S_r*F_v$

end

end

$J_{dec}(J_{dec}==1)=0$

$J_{dec}(J_{dec}>1) \ =1$

Finding initial $C_p(C)$

$D=C_p*S_r*F_v$

m=1

for i=1:|(CW_r)|

if $max(and(D(i,:),J_{dec}(i,:)))$==1

$C_p(C)((m,:))=CW_r(i,:)$

$m=m$+1

end

end ■

After computing C_p, $J_{dec}(C)$ and the initial $C_p(C)$ using Algorithm 4.2, we use Algorithm 4.1 to compute $C_p(C)$ recursively. Note that Algorithm 1 only computes the set $C_p(C)$ for a CW C under consideration.

Then, this set is converted to a single row by using union of all the rows of $C_p(C)$. The same procedure is followed for all the other CWs in the system using a simple 'for loop' to get the complete set $C_p(C)$ for all the CWs in the system.

The complexity of Algorithm 4.2 is $O(|(CW_r)|)$ as the running time is directly proportional to the number of CWs in the system. For calculating the same for all the CWs in the system, the complexity is $O(|(CW_r)|^2)$.

4.4.3 Critical Siphons & Subsystems in Matrix Form

In this section we use matrices to compute the PN objects i.e. Critical Siphons and Critical Subsystems required in Section III for deadlock analysis in FMRF systems. We also show how key resources can be found using matrices.

$\bullet C$ *and* $C \bullet$ are the set of input and output transitions from a CW C. Once CW_r is constructed, we use Algorithm 2 and then Algorithm 1 to construct $C_p(C)$ for each $C \in CW_r$. Let $\bullet C_p(C)$ and $C_p(C) \bullet$ be the set of input and output transitions from $C_p(C)$. In matrix formulation, it is denoted as ${}_dC_p(C)$ and $C_p(C)_d$ respectively. It is computed as:

$${}_dC_p(C) = C_p(C) \otimes S_r \tag{4.12}$$

$$C_p(C)_d = C_p(C) \otimes F_r^T \tag{4.13}$$

To find the set of transitions $CW_t(C_p(C))$ between resources in the set $C_p(C)$, we use,

$$CW_t(C_p(C)) = {}_d\, C_p(C) \otimes C_p(C)_d \tag{4.14}$$

The critical subsystems $J(C_p(C))_0$ for a given CW_r are given by,

$$J(C_p(C))_0 = CW_t(C_p(C)) \otimes F_v \tag{4.15}$$

The siphon job sets $J(C_p(C))_+$ are given by,

$$J(C_p(C))_+ = {}_dC_p(C) \otimes (\overline{{}_dC_p(C) \otimes C_p(C)_d}) \otimes F_v \tag{4.16}$$

Equations (4.15) and (4.16) are used for checking the condition $J(C_p(C))_+ \cap J(C_p(C))_0 = \phi$ (in Theorem 4.1-4.3). Equation (4.15) is used for deadlock avoidance in FMRF systems using the MAXWIP policy.

It is necessary to find key resources in the FMRF system to verify its regularity. To find key resources, following formulations are used.

The job sets $J(C)_0$ for each CW are given as,

$$J(C)_0 = CWt \otimes F_v \tag{4.17}$$

The precedent and posterior transitions of CWr is given as,

$$T_{pre} = J(CW_r)_0 S_v \wedge J(\overline{CW_r})_0 F_v^T \tag{4.18}$$

$$T_{pos} = J(CW_r)_0 F_v^T \wedge J(\overline{CW_r})_0 S_v \tag{4.19}$$

The set of cyclic circular waits (CCW) are given by,

$$CCW = T_{pre} T_{pos} \wedge (T_{pre} T_{pos}^T)^T \tag{4.20}$$

The cyclic precedent and posterior transitions are given as,

$$\hat{T}_{pre} = CCWT_{pos} \wedge T_{pre} \tag{4.21}$$

$$\hat{T}_{pos} = CCWT_{pre} \wedge T_{pos} \tag{4.22}$$

The set of key resources in the system is given by,

$$\mathrm{Re}\, sCW = \hat{T}_{pos}\, F_r \wedge \hat{T}_{pre}\, F_r \tag{4.23}$$

If Res*CW* is empty, it means that the system does not contain key resources and the system is regular. . In fact it is easy to see that Equation (4.23) is a matrix form of the condition in Definition 4.1 for key resources. Equation (4.23) is the test for regularity in FMRF systems. If the system is not regular, it should be redesigned so that the deadlock avoidance policy in the next section can be used.

Now we have all the machinery to compute using matrices the objects required for deadlock avoidance using Theorem 4.3 and Lemma 4.11. The important equations are (4.15), (4.16) and (4.23).

4.4.4 MAXWIP Dispatching Policy for DA

Lemmas 4.8, 4.9, and 4.10 formulate the MAXWIP dispatching policy for deadlock avoidance in FMRF systems. According to this policy, CB in FMRF is avoided by limiting WIP in the critical subsystems of each CW *C*. This policy can be implemented as a real-time deadlock avoidance control scheme by controlling the firing of the precedence transitions of the critical subsystems.

4.5 Examples

We now illustrate the new notions developed in this chapter, as well as the power of the matrix formulation, by computing them for two example FMRF systems. In the first example, deadlock can occur, while a small change in the structure yields Example 4.2, where deadlock can never occur. The third example is taken from [44] and is an example of an irregular system.

Example 4.1: Consider Figure 4.3. This particular system contains three decision resources B1, B2 and B3. According to Section 4.4, the relevant matrices for calculating the siphon jobs and critical subsystems are F_r, S_r, F_v and S_v . F_v is a matrix of jobs required to fire transitions. S_v is a matrix of jobs started on firing of transitions. F_r is a matrix of resources required to fire transitions and S_r is a matrix of resources released when transitions are fired. These matrices for the system in Figure 4.3 are given by,

$$F_r = \begin{array}{c} \begin{matrix} R1 & R2 & R3 & R4 & M1 & M2 & B1 & B2 & B3 \end{matrix} \\ \begin{bmatrix} 0&0&0&0&0&0&1&0&0 \\ 1&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&1 \\ 1&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \end{bmatrix} \end{array} \begin{array}{c} t1 \\ t2 \\ t3 \\ t4 \\ t5 \\ t6 \\ t7 \\ t8 \\ t9 \\ t10 \\ t11 \\ t12 \\ t13 \\ t14 \\ t15 \\ t16 \\ t17 \end{array}$$

$$S_r^{\,T} = \begin{array}{c} \begin{matrix} R1 & R2 & R3 & R4 & M1 & M2 & B1 & B2 & B3 \end{matrix} \\ \begin{bmatrix} 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0&0 \\ 1&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&1 \\ 1&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \end{bmatrix} \end{array} \begin{array}{c} t1 \\ t2 \\ t3 \\ t4 \\ t5 \\ t6 \\ t7 \\ t8 \\ t9 \\ t10 \\ t11 \\ t12 \\ t13 \\ t14 \\ t15 \\ t16 \\ t17 \end{array}$$

$$F_v = \begin{array}{c} \begin{matrix} b1 & r1a & r2a & b2 & m1 & m2 & b3 & r1b & r2b & r2c & r3a & r4a \end{matrix} \\ \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1 \end{bmatrix} \end{array} \begin{array}{c} t1 \\ t2 \\ t3 \\ t4 \\ t5 \\ t6 \\ t7 \\ t8 \\ t9 \\ t10 \\ t11 \\ t12 \\ t13 \\ t14 \\ t15 \\ t16 \\ t17 \end{array}$$

$$S_v^{\,T} = \begin{array}{c} \begin{matrix} b1 & r1a & r2a & b2 & m1 & m2 & b3 & r1b & r2b & r2c & r3a & r4a \end{matrix} \\ \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix} \end{array} \begin{array}{c} t1 \\ t2 \\ t3 \\ t4 \\ t5 \\ t6 \\ t7 \\ t8 \\ t9 \\ t10 \\ t11 \\ t12 \\ t13 \\ t14 \\ t15 \\ t16 \\ t17 \end{array}$$

The set of CWs in this system using the binary string algebra [32, 50] is given by:

$$CW_r = \begin{array}{c} \begin{matrix} R1 & R2 & R3 & R4 & M1 & M2 & B1 & B2 & B3 \end{matrix} \\ \begin{bmatrix} 1&0&0&0&1&0&0&1&1 \\ 1&0&0&0&0&1&0&1&1 \\ 0&1&0&0&1&0&0&1&1 \\ 0&1&0&0&0&1&0&1&1 \\ 1&0&0&0&1&1&0&1&1 \\ 1&1&0&0&1&0&0&1&1 \\ 1&1&0&0&1&1&0&1&1 \\ 1&1&0&0&0&1&0&1&1 \\ 0&1&0&0&1&1&0&1&1 \end{bmatrix} \end{array}$$

Each row in this set represents a CW e.g. the first row of this set represents the CW R1-M1-B1-B3. Using Algorithm 4.2 and 4.1, the set $C_p(C)$ for this system is computed as:

$$C_p(C) = \begin{array}{c} \begin{matrix} R1 & R2 & R3 & R4 & M1 & M2 & B1 & B2 & B3 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

We observe that in this case, each row is the same, which indicates that each CW in CW_r is related by the decision resources B2 and B3. This is why we end up with the same resources in each row of $C_p(C)$. In fact, all the CWs in CW_r are in the same equivalence class $C_p(C)$.

The siphon jobs and critical subsystems using Equations (4.15) and (4.16) are given by

$$J(C_p(C))_+ = \begin{array}{c} \begin{matrix} b1 & r1a & r2a & b2 & m1 & m2 & b3 & r1b & r2b & r2c & r3a & r4a \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

$$J(C_p(C))_0 = \begin{array}{c} \begin{matrix} b1 & r1a & r2a & b2 & m1 & m2 & b3 & r1b & r2b & r2c & r3a & r4a \end{matrix} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

In this case, $J(C_p(C))_0 \cap J(C_p(C))_+ = \phi$. Hence, if all the jobs in $J(C_p(C))_0$ i.e. *r1a*, *r2a*, *b2*, *m1*, *m2*, and *b3* are marked, then according to Theorem 4.3, $C_p(C)$ is in CB and deadlock will occur (See Figure 4.3). One can also verify that this system is regular by applying Equation (4.23). When the MAXWIP policy in Lemma 11 is applied, deadlock is avoided. ■

Example 4.2: Figure 4.5 shows a small change in the PN structure of Figure 4.2. A new resource R5 is added to the system which performs the task *r2b*, i.e. R2 is not used for *r2b*.

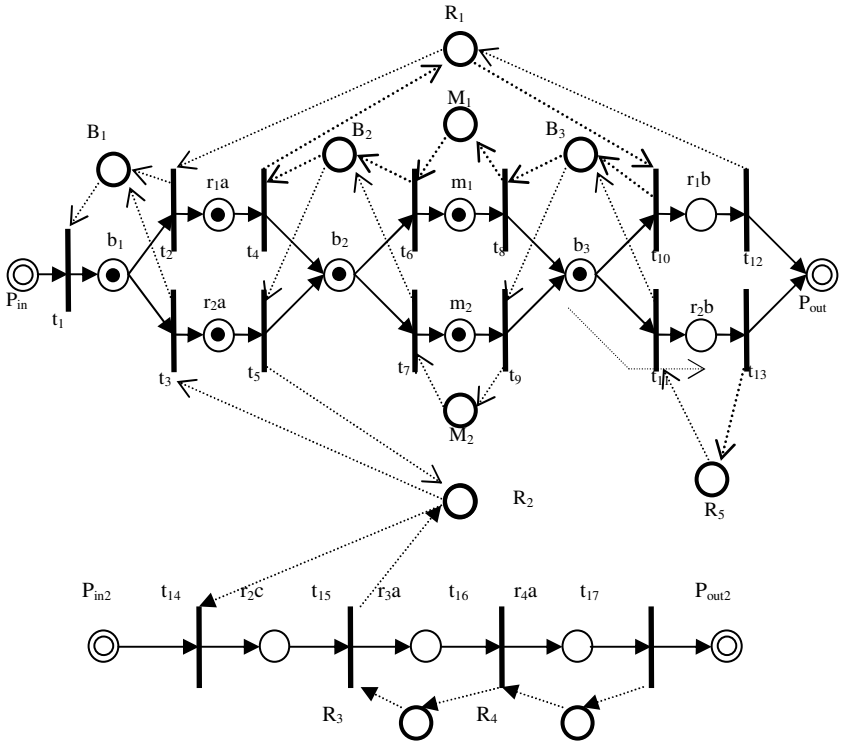


Figure 4.5 Case where $J(C_p(C))_0 \cap J(C_p(C))_+ \neq \phi$

Here, only the F_r and S_r matrices change. The matrices F_v and S_v remain the same. This is because a new resource is added to the system, but the tasks and their transitions remain the same. The addition of a new resource results in an additional column in the F_r and S_r^{T} matrices.

$$F_r = \begin{array}{c} \begin{matrix} R1 & R2 & R3 & R4 & R5 & M1 & M2 & B1 & B2 & B3 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{l} t1 \\ t2 \\ t3 \\ t4 \\ t5 \\ t6 \\ t7 \\ t8 \\ t9 \\ t10 \\ t11 \\ t12 \\ t13 \\ t14 \\ t15 \\ t16 \\ t17 \end{array} \quad S_r^T = \begin{array}{c} \begin{matrix} R1 & R2 & R3 & R4 & R5 & M1 & M2 & B1 & B2 & B3 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{l} t1 \\ t2 \\ t3 \\ t4 \\ t5 \\ t6 \\ t7 \\ t8 \\ t9 \\ t10 \\ t11 \\ t12 \\ t13 \\ t14 \\ t15 \\ t16 \\ t17 \end{array}$$

The corresponding CW_r and $C_p(C)$ matrices are given by:

$$CW_r = \begin{array}{c} \begin{matrix} R1 & R2 & R3 & R4 & R5 & M1 & M2 & B1 & B2 & B3 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

$$C_p(C) = \begin{array}{c} \begin{matrix} R1 & R2 & R3 & R4 & R5 & M1 & M2 & B1 & B2 & B3 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

The siphon jobs and critical subsystem using Equations (4.15) and (4.16) are given by:

$$J(C_p(C))_+ = \begin{array}{c} \begin{matrix} b1 & r1a & r2a & b2 & m1 & m2 & b3 & r1b & r2b & r2c & r3a & r4a \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$J(C_p(C))_0 = \begin{matrix} \text{b1} & \text{r1a} & \text{r2a} & \text{b2} & \text{m1} & \text{m2} & \text{b3} & \text{r1b} & \text{r2b} & \text{r2c} & \text{r3a} & \text{r4a} \end{matrix} \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

We see that $J(C_p(C))_0 \cap J(C_p(C))_+ \neq \phi$. Here, $R5 \notin C_p(C)$ and therefore, the task $b3 \in J(C_p(C))_0$ and $b3 \in J(C_p(C))_+$. Hence, according to Theorem 4.2, deadlock can never occur. ■

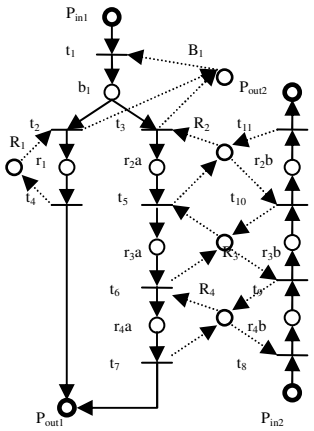


Figure 4.6 Example of an irregular system

Example 4.3: Figure 4.6 is a sample FMRF/S3PR PN from [44]. This system is an example of an irregular system. Using Algorithm 4.2 and 4.1, one can see that $C_p(C) = CW_r$.

Following are the relevant matrices for this system.

$$F_r = \begin{array}{c} \begin{matrix} B1 & R1 & R2 & R3 & R4 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} t1 \\ t2 \\ t3 \\ t4 \\ t5 \\ t6 \\ t7 \\ t8 \\ t9 \\ t10 \\ t11 \end{matrix} \quad S_r{}^T = \begin{array}{c} \begin{matrix} B1 & R1 & R2 & R3 & R4 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} t1 \\ t2 \\ t3 \\ t4 \\ t5 \\ t6 \\ t7 \\ t8 \\ t9 \\ t10 \\ t11 \end{matrix}$$

$$F_v = \begin{array}{c} \begin{matrix} b1 & r1 & r2a & r3a & r4a & r4b & r3b & r2b \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} t1 \\ t2 \\ t3 \\ t4 \\ t5 \\ t6 \\ t7 \\ t8 \\ t9 \\ t10 \\ t11 \end{matrix} \quad S_v{}^T = \begin{array}{c} \begin{matrix} b1 & r1 & r2a & r3a & r4a & r4b & r3b & r2b \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} t1 \\ t2 \\ t3 \\ t4 \\ t5 \\ t6 \\ t7 \\ t8 \\ t9 \\ t10 \\ t11 \end{matrix}$$

$$CW_r = \begin{array}{c} \begin{matrix} B1 & R1 & R2 & R3 & R4 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

Equation (4.23) gives ResCW as,

$$\mathrm{Re}\,sCW = \begin{array}{c} \begin{matrix} B1 & R1 & R2 & R3 & R4 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

This means R3 is a critical resource and when m(R3)=1, R3 becomes a key resource and system becomes irregular. If the system is not regular, it should be redesigned so that the deadlock avoidance policy in Lemma 4.11 can be used. ■

Chapter 5 shows how real-time dynamic decisions for resource dispatching and task assignment can be taken in a FMRF system using matrix-based Dempster Shafer Theory.

CHAPTER 5

DECISION-MAKING IN FMRF

5.1 Introduction

High Quality of service and good performance of a system are goals which engineers always strive to achieve. Reliability engineering integrates quality and performance of a system from the beginning to the end of the system life. Failure avoidance is the main approach for reliability assurance. There are two main types of failure avoidance in terms of maintenance, namely preventive and corrective. In preventive maintenance, all actions are intended to keep equipments in good operating condition. It should be able to indicate when a failure may occur so that actions can be taken to avoid failures. In corrective maintenance, a repair is performed after a failure has occurred. Condition-based maintenance (CBM) is an approach of preventive maintenance. Figure 5.1 shows an overview of maintenance strategies used in industries [4, 83, 89, 90, 95, 96]. The process of CBM involves monitoring the system, predicting failures and making repairs before these failures occur. This process can be scheduled or continuous. A system can contain many fault modes and a decision has to be taken on the type of repair necessary for eliminating any future faults. The task flow of detection and repair can be modeled using Petri Nets.

Petri Nets (PN) [55] have been used for job scheduling in manufacturing systems and wireless sensor networks [12, 32, 47, 48, 50, 51, 95]. One particular type of

PN called Free Choice Petri Nets can be used for CBM for fault diagnosis and fault avoidance in machinery. In Free-choice Petri Nets, also called as Free-choice Multi Reentrant Flow lines (FMRF) [2], there is a choice for tasks to be performed in the system. Hence decisions have to be made. There are multiple resources (Machines or sensors) present in the system that can perform tasks.

Though the PN framework offers rigorous grounds for theoretical analysis, it is sometimes inconvenient for actual computational analysis of the practical DES, causing problems of computational complexity in scheduling. Matrix techniques that exploit the PN structure can be used to alleviate this. Matrix-based discrete event controller in [47] has proved to be very efficient in computing PN objects and in sequencing tasks in manufacturing environments as well as sensor networks. This matrix form for DES provides a means for efficient implementation of DE controllers in actual systems. We use the matrix formulations of [47] in the CBM framework.

Since decisions have to be made in FMRF systems, various methods can be used such as belief PN, fuzzy PN and possibilistic PN [58]. In this chapter, we use evidence theory (also called Dempster Shafer theory [14, 15, 66-81]) for making decisions. Since routing choices must be made in FMRF systems, we use Dempster Shafer decision-making to select the correct job path in the PN structure. The reason for using Dempster Shafer theory is that it provides an excellent framework for conditions involving uncertainty.

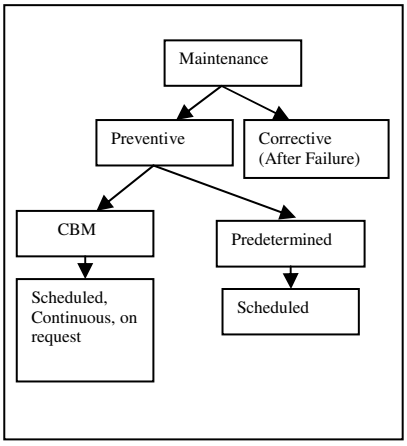


Figure 5.1 Overview of Maintenance

Dempster Shafer (DS) theory has been used extensively for data fusion in manufacturing systems and sensor networks. Smets [68-81] provides a framework for Transferable Belief Model for representing the quantified beliefs held by an agent at a given time on a given frame of discernment. It concerns the same concepts as considered by the Bayesian model, except it does not rely on probabilistic quantification, but on a more general system based on belief functions.

The mathematics of computation involving DS belief functions is difficult to fathom because of the many summations over set inclusions and intersections. The equations are often difficult to comprehend and discourage readers due to their complexity, and are often difficult to implement using software.

However, most of the operations in belief function theory happen to be linear operations and can be represented using the matrix notation [68-81]. Therefore, in this chapter we provide a new matrix formulation for updating evidence and computing beliefs and plausibilities in DS theory. This makes DS computations very easy to

perform using computer software such as MATLAB. Matrices also help greatly for the readability of the equations and the ease in their manipulations.

The chapter is organized as follows. In section 5.2, we give a brief description of Dempster Shafer theory and the rule for combination of evidence. Section 5.3 provides a new matrix formulation required for computing objects in the framework of evidence theory. Our motivation is to use PN for detecting machine failures and assigning actions for CBM. We use the class of PN called FMRF where multiple tasks can be assigned in a flow line. In Section 5.4, we show how evidence theory can be used in FMRF systems for CBM. Examples are provided in Sections 5.3 and 5.4 for better understanding of the framework.

5.2 Dempster Shafer Theory

In this section we provide a brief background on Dempster Shafer theory and the rule for combination of evidence. If θ is a frame of discernment then a function m: $2^{\theta} \rightarrow [0, 1]$ is called a basic probability assignment (bpa) if $m(\phi) = 0$

and

$$\sum_{A \subset \theta} m(A) = 1 \tag{5.1}$$

The term $m(A)$ is called A's basic probability number and $m(A)$ is the measure of the belief that is committed exactly to A. The subsets A of the frame of discernment θ for which $m(A)$ are strictly positive are called the focal elements of the basic probability assignment. The union of all the focal elements of a belief function is called its core.

Properties of bpa:

1. It is not required that $m(\theta)=1$.
2. It is not required that $m(A) \leq m(B)$ when $A \subset B$.
3. No relationship between $m(A)$ and $m(\overline{A})$ is required.
4. Also $m(A)+m(\overline{A})$ does not always have to be 1.

A Belief function Bel: $2^{\theta} \rightarrow [0, 1]$ is defined as,

$$Bel(A)=\sum_{B \subset A} m(B) \tag{5.2}$$

For any $A \subset \theta$. $Bel(A)$ measures the total belief of all possible subsets of A. The belief function measures how much the information given by a source supports the belief in a specified element as the right answer.

Properties of the Belief Function:

1. $Bel(\Phi)=0$.
2. $Bel(\theta)=1$.
3. $Bel(A_1 \cup ..A_n) \geq \sum_i Bel(A_i) - \sum_{i<j} Bel(A_i \cap A_j) + .. + (-1)^{n+1} Bel(A_1 \cap ... \cap A_n)$.

A Plausibility function Pl: $2^{\theta} \rightarrow [0, 1]$ is defined as,

$$Pl(A)=\sum_{B \cap A \neq 0} m(B) \tag{5.3}$$

Plausibility measures how much the information given by a source does not contradict a specified element as the right answer, i.e. how much we should believe in an element if all unknown beliefs are assigned to it.

Properties of the Plausibility Function:

1. $Pl(\Phi)=0$.

2. $Pl(\theta)=1$.

3. $Pl(A_1 \cap .. A_n) \leq \sum_i Pl(A_i) - \sum_{i<j} Pl(A_i \cup A_j) + .. + (-1)^{n+1} Pl(A_1 \cup ... \cup A_n)$.

Let Bel_1 and Bel_2 be two belief functions induced by two distinct pieces of evidence. Let m_1 and m_2 denote their bpas, respectively. The Dempster rule of combination aims at building the bpa that represents the impact of the combined evidence. It is defined as,

$$m_{1,2}(A) = \frac{\sum m_1(B_i) m_2(C_j)}{1-\kappa} \tag{5.4}$$

where the conflict $\kappa = \sum_{B \cap C = \phi} m_1(B_i) m_2(C_j)$. The conflict κ is the normalizing factor. The Dempster's rule is called the conjunctive rule of combination as it builds the bpa when both pieces of evidence are accepted.

Smets [68-81] provides a method for decision-making in the context of Transferable Belief Model (TBM). The TBM works at two modes:

1. The Credal level where the beliefs are entertained and are represented by belief functions.
2. The Pignistic level where the beliefs are used to make decisions and are represented by probability functions called pignistic probabilities.

The pignistic probability *Bet* is defined as,

$$Bet(B) = \sum_{A \subseteq \theta} \frac{|B \cap A|}{|A|} m(A), \text{ for all } B \subset \theta \tag{5.5}$$

This function will allow us to take decisions under uncertainties in a FMRF system which will be explained in Sections 5.3 and 5.4.

5.3 Matrix Formulation for Dempster Shafer Theory

The mathematics of DS theory can become quite difficult due to various summations over set inclusions and intersections. This has hampered the use of DS theory in real-time discrete event control systems. The DS formulation can be represented using matrices as most of the operations in DS theory are linear. In this section we define matrix techniques that can ease the computation involved in DS theory and make it practically more useful for real-time supervisory control.

Consider for illustration, the sets in the frame of discernment as shown in the Figure 5.2.

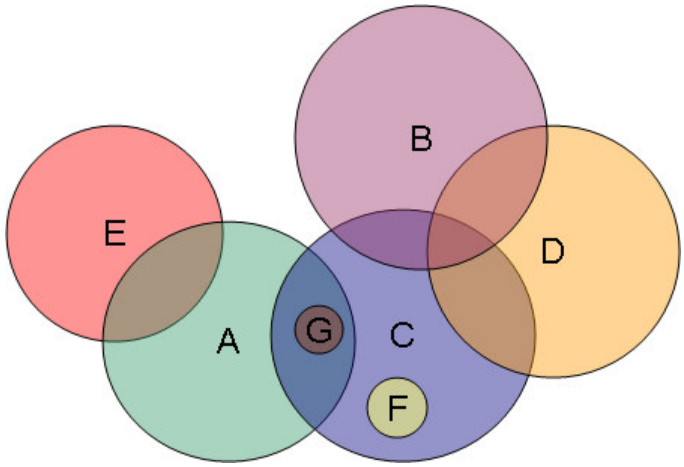


Figure 5.2 Sets- Unions and Intersections

Given a frame of discernment θ with focal elements $\{S_1, S_2, \cdots, S_q\}$, define a $q \times q$ containment matrix C as a matrix with element $(i, j)=1$ if $S_i \supset S_j$, and $(i, j)=0$ otherwise. For example, C for Figure 5.2 is given as,

$$C = \begin{array}{c} \begin{matrix} A & B & C & D & E & F & G \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \\ G \end{matrix} \end{array}$$

Define a $q \times q$ intersection matrix E having element $(i, j)=1$ if $S_i \cap S_j \neq \phi$, and $(i,j)=0$ otherwise. For example, E for Figure 5.2 is given as,

$$E = \begin{array}{c} \begin{matrix} A & B & C & D & E & F & G \end{matrix} \\ \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \\ G \end{matrix} \end{array}$$

For a given E, the conflict matrix is defined as $K = \bar{E}$, with the over-bar denoting the logical negation.

Define a q-vector of bpas indexed by the focal elements of θ as $m = [m(S_1)\ m(S_2)...m(S_q)]^T$. Define a commensurate belief q-vector $Bel = [Bel(S_1)...Bel(S_q)]^T$ and similarly a plausibility q-vector Pl. Then, the following Lemmas can be used to find the Belief and Plausibility using matrices.

Lemma 5.1: Given the containment matrix C and the bpa vector m, the Belief can be calculated as,

$$Bel = C \bullet m \tag{5.6}$$

Proof: $C(i, j)$=1 only if $S_i \supset S_j$, and $C(i, j)=0$ otherwise. Since, m is a vector of bpas of all the focal elements, Cm denotes the vector $Bel(S_i) = \sum_{S_j \subset S_i} m(S_j)$ *for* $i, j = 1$ *to* $|m|, S_i, S_j \in \theta$. ■

Lemma 5.2: Given the intersection matrix E and the bpa vector m, the Plausibility can be calculated as,

$$Pl = E \bullet m \tag{5.7}$$

Proof: In the same lines as Lemma 1, $E(i, j)$= 1 only if $S_i \cap S_j \neq \phi$ and zero otherwise. Since, m is a vector of bpas of all the focal elements, Em denotes the vector $Pl(S_i) = \sum_{S_j \cap S_i \neq \phi} m(S_j)$ *for* $i, j = 1$ *to* $|m|, S_i, S_j \in \theta$. ■

The Dempster rule of combination is computed efficiently as follows. Given two bodies of evidence, the two bpa vectors m_1 and m_2 will not generally be commensurate since the two cores are different, e.g. the two vectors are indexed by different focal element sets. Therefore, merge the two cores to obtain the joint core, consisting of all the focal elements of the two bodies of evidence, each occurring once. Expand the two bpa vectors m_1 and m_2, indexing each by the same joint core set and adding zeros as necessary to the vectors. Define the combination matrix,

$$S = m_2 m_1^{T} \tag{5.8}$$

and the conflict as,

$$K_{1,2} = m_2^{T} K m_1 \tag{5.9}$$

Lemma 5.3: For the given bpa vectors m_1 and m_2, the Dempster's rule of combination to find $m_{1,2}$ using matrices is given by,

$$m_{1,2} = (diag(S \times C \ + C^T \times S) - diag(S)) / (1 - K_{1,2}) \tag{5.10}$$

Proof: $C(i, j)=1$ only if $C_i \supset C_j$ and zero, otherwise. S is the combination matrix. Now, let $S_1=SC$. $S_1(i, j) = \sum_{r=1}^{n} S_{ir} C_{rj}$. The diagonal $S_1(i, i)$ contains the sum of all the elements such that $m_j \subset m_i$, $\forall i, j$. Similarly, let $S_2=C^TS$. Again, the diagonal $S_2(i, i)$ contains the sum of all the elements such that, $m_i \subset m_j \ \forall i, j$. Let $m_{1,2}= diag(S_1+S_2)$. Since $diag(S)=diag(S^T)$, the additional term in the diagonal diag(S_1+S_2) must be eliminated from the sum $m_{1,2}$ such that the condition $\sum_{B \cap C = A \neq 0} m_1(B_i) m_2(C_j)$ holds. Hence, $m_{1,2}=$ $diag(S1+S2-S)/(1-K)$, satisfies the condition, $\dfrac{\sum_{B \cap C = A \neq 0} m_1(B_i) m_2(C_j)}{1 - \kappa}$. ■

Example 5.1 (Diagnostics): Let t_1 denote a mechanical failure, t_2 denote an electrical failure and (t_1, t_2) denote a mechanical or an electrical failure. Then the bpa vector consists of $[t_1, t_2, (t_1, t_2)\ \theta]^T$ where θ contains all the sets. Sensor 1 provides the initial bpa vector for these sets as $m_1=[0.4\ 0.3\ 0.3\ 0]^T$ and sensor 2 provides the initial bpa vector as $m_2=[0.5\ 0.2\ 0.3\ 0]^T$. Then, the relevant matrices for this problem are:

$$C = \begin{array}{c} \begin{matrix} t_1 & t_2 & t_1, t_2 & \theta \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array} \begin{matrix} t_1 \\ t_2 \\ t_1, t_2 \\ \theta \end{matrix} \quad \text{and} \quad E = \begin{array}{c} \begin{matrix} t_1 & t_2 & t_1, t_2 & \theta \end{matrix} \\ \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array} \begin{matrix} t_1 \\ t_2 \\ t_1, t_2 \\ \theta \end{matrix}$$

Using equations (5.8), (5.9) and (5.10), we get the combination of evidence as $m_{1,2}$=[0.6104 0.2727 0.1169 0] ^{T}. This means that there is 61.04% chance of a mechanical failure, 27.27% chance of an electrical failure and 11.69% chance of electrical or mechanical failure. The combined belief and plausibility using equations 6 and 7 are [0.6104 0.2727 1 1]^{T} and [0.7273 0.3896 1 1]^{T} respectively. ■

Finding Belief and Plausibility is not sufficient for taking concrete decisions. For instance, providing the *Bel* and *Pl* of the set $\{t_1, t_2\}$ does not show how to choose whether t_1 or t_2 occurred. The pignistic transformation provides concrete probabilities for the elements in a set so that a decision can be taken. Smets [68-81] provides a formulation for finding the pignistic probabilities using matrices. Let m be the bpa vector. Then the pignistic transformation is given as,

$$Bet = BetM \times m \tag{5.11}$$

where, *BetM* is a user-selected matrix that transfers belief from a given set to its subsets for decision-making. In the example just given it is not enough to assign *Bel* and *Pl* to the set $\{t_1, t_2\}$ for taking concrete decisions on t_1 or t_2. For the set $\{t_1, t_2, (t_1,t_2)\ \theta\ \}$, the matrix *BetM* can be given as,

$$\begin{array}{c} \quad t_1 \quad t_2 \quad t_1,t_2 \quad \theta \\ BetM = \begin{bmatrix} 1 & 0 & 1/2 & 0 \\ 0 & 1 & 1/2 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{array}{c} t_1 \\ t_2 \\ t_1,t_2 \\ \theta \end{array} \end{array}$$

Each element in *BetM* matrix is chosen in such way that the sum of all the elements in *Bet* equals one. The *BetM* matrix is selected by the designer to apportion probabilities as

prescribed by the sets. Hence, decision can now be taken on t_1 and t_2 in the presence of uncertainty because we have pignistic probabilities for each primitive event, i.e. t_1 and t_2 instead of basic probability assignments on sets of events.

For Example 5.1, the pignistic probabilities are given as,

$$Bet_{1,2}[t_1\ t_2\ (t_1,t_2)\ \theta]^T = BetM \times m_{1,2} = [0.6688\ \ 0.3312\ \ 0\ 0]^T$$

This means the probability of mechanical failure ($Bet(t_1)$) is 0.6688 and probability of an electrical failure ($Bet(t_2)$) is 0.3312 i.e. the machine is much likely to have a mechanical failure than an electrical failure.

5.4 Embedding DS into FMRF Systems

In CBM, we are required to monitor a system, predict failures and repair the system before the failures occur. The flow of detection and repair can be modeled as a FMRF PN, where a decision place denotes the decision-making component involved in predicting failures. For example, in the PN shown in Figure 4.2 (Chapter 4), P_{in} might represent fault detection in a system. The resources might represent sensors that detect the faults or machines that repair the faults. The decision places represent the tasks that decide which future job path to take to repair the faults based on the type of fault in the system.

In chapter 4, we mentioned that a decision must be made to decide which transitions in a FMRF system must be enabled. Consider figure 5.3. The system in figure 5.3 consists of two decision transitions namely t_1 and t_2, while A is a decision place.

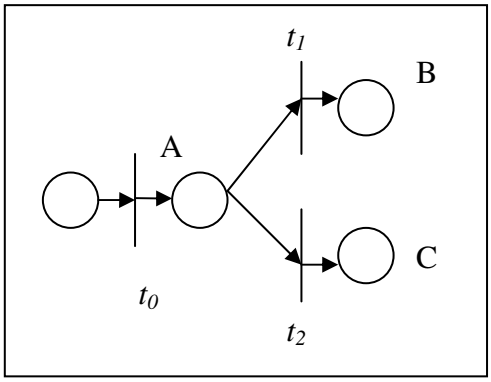


Figure 5.3 Decision in FMRF system

The problem in FMRF system is to find which transition out of t_1 and t_2 should be enabled. The possibilities are t_1 is enabled, or t_2 is enabled.

From Equation 2.5, the post transitions are given by $F_v \otimes v$. In the case of FMRF systems, $F_v \otimes v$ enables multiple transitions if v is a decision place. In the case of Equation 2.5, x is the vector of transitions and multiple transitions can be enabled if a given task is a decision task and a given resource is a decision resource. A Matlab routine can be used to find the number of decision transitions in a FMRF system as shown in Routine 5.1.

```
% Initialize matrices Fv, Sv, Fr and Sr
sizeFv=size(Fv)
n=zeros(sizeFv(2),1) % Initialize job vector
for i=1:sizeFv(2)
   n(i)=1;
   Y=Fv*n
   if sum(Y)>1   % finding decision transitions in FMRF
      A= find(Y==1) %finds the indices of decision transitions
   end
   n(i)=0;
end
```

Routine 5.1. Finding Decision Transitions

The vector A contains the indices of the decision transitions. This can be used for updating vector x in the DEC sequence after a decision is made. Let m_1, $m_2 \ldots m_n$ be the bpas for the decision transitions and their unions in ascending order. Then the probability of each transition can be found using equations (5.5-5.8). The decision transition vector ζ which is actually a '*Bet*' on transitions is then given by,

$$\zeta = BetM \times m \tag{5.12}$$

Here, ζ is the pignistic probability of the transitions and their unions discussed in Section 5.3-Equation 5.11. Each decision place has associated bpas for its post transitions in the vector ζ.

We define a threshold *th*, such that,

$$\begin{aligned} \zeta(i) &= 1 \quad if \ \ \zeta(i) > th \\ &= 0 \quad Otherwise \end{aligned}$$

The state transition vector can then be updated using a Matlab routine as shown below:

```
% Given A, BetM, x, th and ζ represented as z

z(z>=th)=1
z(z<th)=0

for i=1:length(A)
    x(A(i))=z(i)
end
```

Routine 5.2. Updating state transition vector x

The process of decision-making is shown in figure 5.4.

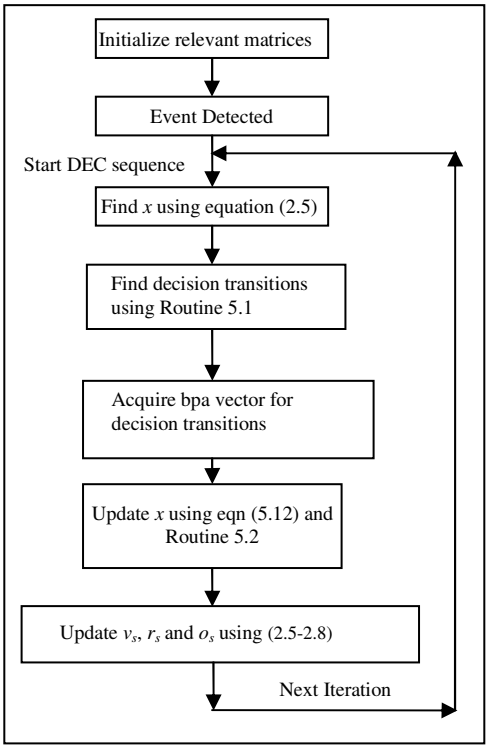


Figure 5.4 Updating the state transition matrix x

Example 5.2 (Machine Diagnostics): Consider figure 5.5. In this example, when a fault in a machine is detected, the DEC sequence is triggered. The sequence is as follows:

1. Sensor A has to find whether machine has an electrical fault or a mechanical fault or both.
2. Machine M has to repair a mechanical fault.
3. Machine E has to repair an electrical fault.
4. Output is “Machine is repaired”.

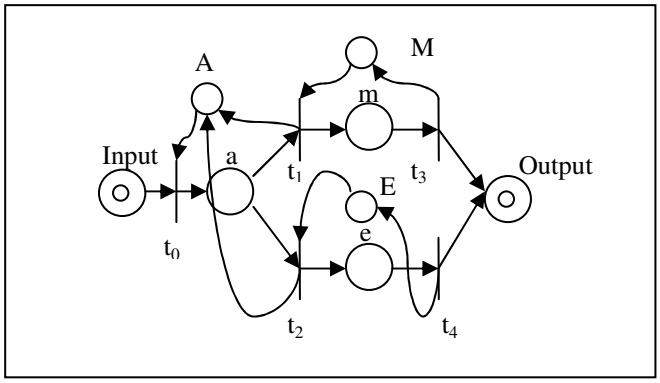


Figure 5.5 Example of FMRF for CBM

Sensor A can be a combination of multiple sensors detecting mechanical or electrical faults. In the case of uncertainties, Sensor A has to decide what action has to be taken. The relevant matrices for this example are:

$$\begin{matrix} & a & m & e \text{ (tasks)} \end{matrix} \qquad \begin{matrix} A & M & E \text{ (resources)} \end{matrix}$$

$$F_v = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{matrix} t_0 \\ t_1 \\ t_2 \\ t_3 \\ t_4 \end{matrix} \qquad F_r = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{matrix} t_0 \\ t_1 \\ t_2 \\ t_3 \\ t_4 \end{matrix}$$

$$S_v^{\,T} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{matrix} t_0 \\ t_1 \\ t_2 \\ t_3 \\ t_4 \end{matrix} \qquad S_r^{\,T} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{matrix} t_0 \\ t_1 \\ t_2 \\ t_3 \\ t_4 \end{matrix}$$

When the task ‘a’ is competed, equation 13 gives the value of x as, $x = [0\,1\,1\,0\,0]^T$. This means that transitions t_1 and t_2 are both triggered. Now we use Dempster Shafer formulation to decide the path. Using Routine 5.1, two decision transitions are found, namely t_1 and t_2. Suppose Sensor A is composed of two unreliable sensors giving bpas for t_1 and t_2 as in Example 5.1, and the threshold *th* is given as 0.5 then the output of

Routine 5.2 gives $x = [0\,1\,0\,0\,0]^T$, which means transition t_1 is enabled and task 'm' will be performed and resource M will be used. This is because; the probability of the mechanical fault is higher than the electrical fault.

In figure 5.6, the system is triggered two times with different bpas for transitions t_1 and t_2. In the first case, let the bpa vector $[t_1\ t_2\ (t_1,t_2)\ \theta]$'=[0.5 0.3 0.2 0] ^{T}, and second case, let it be [0.2 0.6 0.2 0] ^{T}. The transition vector for these cases when $x = [0\,1\,1\,0\,0]^T$ are $x = [0\,1\,0\,0\,0]^T$, and $x = [0\,0\,1\,0\,0]^T$ respectively, since ζ =[0.6 0.4 0 0]^T, and [0.3 0.7 0 0] ^{T} for the two cases. ■

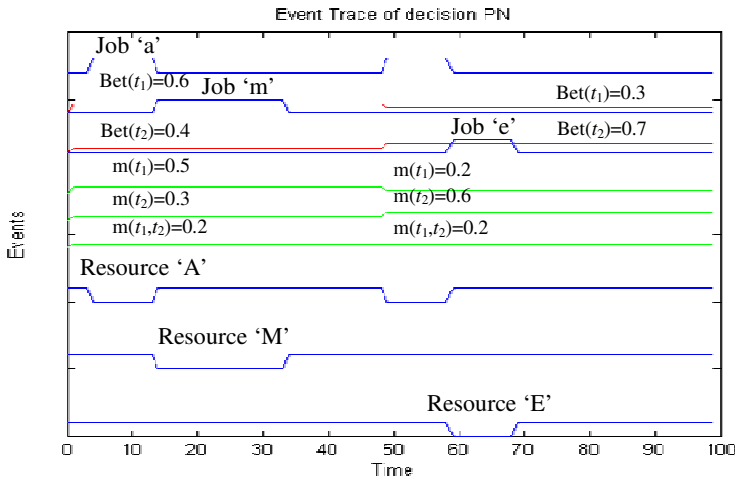


Figure 5.6 Event trace of Example 5.2

This example depicts a very simple system, but this architecture can be extended to complicated systems with more number of decision transitions. Some systems may have shared resources which might cause problems of deadlock. This can be resolved using the MAXWIP deadlock avoidance policy mentioned in [47].

Chapter 6 shows how trust can be established between nodes in a distributed system so that dynamic decisions can be taken for a consensus.

CHAPTER 6

TRUST CONSENSUS IN DIRECTED GRAPHS

6.1 Introduction

Battlefield or disaster area teams may be heterogeneous networks consisting of interacting humans, ground sensors, and unmanned airborne or ground vehicles (UAV, UGV). Developed team scenarios include the War-fighter Information Network-Tactical (WIN-T), DARPA Agile Information Control Environment (AICE), C4ISR Architectures for the War-fighter (CAW), Joint Force Air Component Commander (JFACC) Project, etc. Such scenarios should provide intelligent shared services of sensors and mobile nodes to augment the capabilities of the remote-site mission commander and on-site war-fighter in terms of: (1) extended sensing ranges, (2) sensing of modalities such as IR and ultrasound not normally open to humans, and (3) cooperative control of UAV/UGV to extend the war fighter strike range. Also (4) Automated decision assistance (via, e.g., handheld PDAs) should be provided to the war fighter based on algorithms that only depend on local information from nearest neighbor sensor nodes or humans, yet yield network-wide guaranteed performance.

Given the presence of enemy components and the possibility of node compromise, a *trust consensus* must be reached by the team that determines which nodes to trust, which to disregard, and which to avoid. Trust algorithms for unmanned nodes must be autonomous computationally efficient numerical schemes. However,

existing schemes for control of dynamical systems on communications graphs (in the style of work by [3, 37, 52]) do not take into account trust propagation and maintenance (such as work by [38, 86]). Yet it is a fact that biological groups such as flocks, swarms, herds, do have built-in trust mechanisms to identify team members, team leaders, and enemies to be treated as obstacles or avoided. Cooperative mission planning should involve decisions made in the context of the trust opinions of all nodes, and be based on performance criteria set by human war fighter nodes or team leaders. These performance criteria may change with time depending on varying mission objectives in the field.

Recently, many researchers have worked on problems that are essentially different forms of agreement problems with differences in the types of agent dynamics, properties of graphs and the names of the tasks of interest. In [23], graph Laplacians were used for the task of formation stabilization for groups of agents with linear dynamics. In [37], directed graphs were be used to represent the information exchange between the agents. In [16], a linear update scheme was introduced for directed graphs. In [8] a Lyapunov-based approach was used to consider stability of consensus synchronization for balanced and weakly connected networks. The work by [52] solved the average consensus problem with directed graphs which required the graph to be strongly connected and balanced. In [59], it was shown that under certain assumptions consensus can be reached asymptotically under dynamically changing interaction topologies if the union of the collection of interaction graphs across some time intervals has a spanning tree frequently enough. The spanning tree requirement is a milder

condition than connectedness and is therefore suitable for practical applications. They also allowed the link weighing factors to be time-varying which provides additional flexibility. In contrast to the aforementioned schemes, the work in this chapter uses a bilinear scheme for trust consensus in directed graphs.

In this chapter, we develop a framework for trust propagation and maintenance in team networks of nodes that yields global consensus of trust under rich enough communication structure graphs. Most of the work in literature considers the graph Laplacian to be static or have time-varying weights. In this chapter we consider the case where the graph Laplacian is a time-varying function of the trusts based on the graph connectivity. This makes the trust consensus scheme bilinear.

Also, most of the consensus schemes in the literature consider the case where the trust consensus lies in [0, 1]. Attacks on the security of networked systems can often be characterized in terms of malicious nodes [38]. To characterize malicious nodes, negative trust values are needed. For the case that trust is in [-1, 1] the situation is more realistic, interesting, and complex. We can now talk about identifying and pruning out malicious nodes. Now, 1 means complete trust, 0 means no opinion, and -1 means complete distrust. For this, we make use of 'One-step Distrust model' [38] or 'Graph Pruning and Reconnection' to achieve trust consensus in the case of distrusted nodes.

There has been a tremendous amount of interest in flocking and swarming that has primarily originated from the pioneering work of Reynolds [63]. The trust consensus schemes developed in this chapter is incorporated into cooperative control

laws that depend on local information from neighboring nodes, yet yield team-wide desired behavior such as flocking.

This chapter is organized as follows. In Section 6.2, we describe the notions involved in trust graphs and formally devise a bilinear trust consensus scheme in continuous time and discrete time. Section 6.3 contains our main results with the convergence performance for the two consensus schemes. Section 6.4 gives examples of emerging team behavior using these schemes with a case studies on flocking and formations.

6.2 Background on Trust Graphs

Given a network of N agents or nodes $V=\{v_1,\ldots,v_N\}$ who are to engage in cooperative trust evaluation. Define a trust graph $G = (V, E)$, where edge $(v_i, v_j)\in E$ if node v_j obtains a direct trust evaluation about node v_i. Note this is backwards from [10, 18]. Define the direct trust neighborhood of node v_i as $N_i = \{v_j : (v_j, v_i)\in E\}$, i.e. the set of nodes with edges incoming to v_i. The graph is directed since if node j can obtain a direct evaluation of trust about node i, the reverse may not be true. Given the trust graph, define the graph adjacency matrix $A = [a_{ij}]$ where $a_{ij} = 1$ if e_{ji} is an edge, and $a_{ij} = 0$ otherwise. A is a constant matrix defined by the direct trust relations between nodes. In fact, adjacency matrix A captures the information flow in the trust graph. Define the in-degree matrix as $D = diag\{n_i\}$, $n_i = \sum_{j\in N_i} a_{ij}$, and the trust graph Laplacian L as $L=D-W$. If there is a directed path, e.g. a sequence of nodes $v_0, v_1, \cdots, v_r$ such that $(v_i, v_{i+1})\in E, i\in\{0,1,\cdots,r-1\}$, then, node v_r should be able to form an indirect trust

opinion about node v_0 based on the opinions of the agents along the path. Likewise, if two paths converge at an agent v_r, each of which contains agent v_0, then v_r has a basis to form a more confident opinion about the trustworthiness of agent v_0 than if there were only a single path.

6.2.1 Trust Consensus Protocols

We encode the trust opinions an agent *i* has about other agents in the network as a trust vector $\xi_i \in R^n$ associated with each *node*, with elements indexed by all the nodes about which node *i* has an opinion. That is $\xi_i = [\xi_{ii_1}\ \xi_{ii_2} ...]^T$ where ξ_{ij} is the trust node *i* has for node *j*. Here, the trust is assumed to be in [0, 1]. We can also have the case where trust is in [-1, 1], where 1 means 'fully trusted', 0 means 'no opinion' and -1 means 'fully distrusted'. For the latter, we make use of 'One-step Distrust model' or 'Graph Pruning and Reconnection' to take care of the distrusted nodes (explained in Section 6.3.4).

Consider the following trust consensus scheme in continuous time.

$$\dot{\xi}_i = u_i \tag{6.1}$$

$$u_i = \sum_{j \in N_i} w_{ij}(\xi_j - \xi_i) \tag{6.2}$$

In [59], w_{ij} was taken as $a_{ij}\sigma_{ij}$ where σ_{ij} is a time-varying weighting factor chosen from any finite set. In [38], w_{ij} was taken as $a_{ij}c_{ij}$, where c_{ij} is the confidence node *i* has in its trust opinion of node *j*. Hence each node has an associated [ξ,*c*], i.e.

trust and confidence which form a semi-group [86]. In [38], the weights c_{ij} were kept constant throughout.

In this paper, we propose the following local voting continuous-time trust protocol,

$$u_i = \sum_{j \in N_i} a_{ij} \xi_{ij} (\xi_j - \xi_i) \tag{6.3}$$

This protocol is bilinear in the trust values. Note that this defines a graph topology that stays constant, yet the edge weights are equal to ξ_{ij}, the trust that node *i* has for its neighbor node *j*. The weighted adjacency matrix is defined by $W = [w_{ij}] = [a_{ij} \xi_{ij}]$. This defines a graph which has a constant topology given by the adjacency matrix *A*, yet whose edge weights vary as node *i* changes its trust opinion about its neighbor nodes. If ξ_i's are scalars, (6.3) can be rewritten as,

$$\begin{aligned} u_i &= \sum_{j \in N_i} a_{ij} \xi_{ij} (\xi_j - \xi_i) \\ &= \sum_{j \in N_i} a_{ij} \xi_{ij} \xi_j - \sum_{j \in N_i} a_{ij} \xi_{ij} \xi_i \\ &= -(D(t) - W(t)) \xi_i \\ \dot{\xi}_i &= -(D(t) - W(t)) \xi_i = -L(t) \xi_i \end{aligned} \tag{6.4}$$

Here, $D(t)$ and $W(t)$ are time-varying in-degree and weighted adjacency matrices respectively, that are functions of node trusts ξ. Also, $L(t)$ is a time-varying matrix which is a function of node trusts. Note that the node trust vectors $\xi_i(t)$ have nonzero entries ξ_{ij} corresponding to the weights of incoming edges e_{ji}, which have $a_{ij} = 1$, but there may also be nonzero entries $\xi_{ij}(t)$ that do not correspond to edges in the graph.

Thus, though a node i forms a trust opinion about more and more nodes as trust propagates through the graph, its direct trust neighbors (the graph edges coming into node i) never change, and are defined by the adjacency matrix A.

Since $\xi_i \in R^N$, we must use Kronecker Product [31, 34] to write,

$$\dot{\xi} = -(L(t) \otimes I_N)\xi \tag{6.5}$$

where I_N is an identity matrix of $N \times N$.

Here, $\xi = [\xi_1^T \cdots \xi_N^T]^T \in R^{N^2}$ is the overall network trust vector.

The Laplacian $L(t)$ corresponds to a time-varying trust graph $G(t)$. The initial Laplacian $L(0)$ corresponds to the initial trust graph $G(0)$. Note that the row sum of $L(t)$ is zero for $\forall t$. Hence, $L(t)$ has a zero eigenvalue corresponding to the right eigenvector of $\mathbf{1}$, where $\mathbf{1}$ is a column vector with all entries equal to one.

We also propose the following nonlinear local voting discrete-time trust consensus protocol based on the Vicsek model [88],

$$\xi_i(k+1) = \xi_i(k) + \frac{1}{1+n_i} \sum_{j \in N_i} \xi_{ij}(\xi_j - \xi_i) \tag{6.6}$$

Equation (6.6) can be rewritten in the scalar case as,

$$\xi_i(k+1) = (I - (I + D(k))^{-1} L(k))\xi_i(k)$$

$$\xi_i(k+1) = F(k)\xi_i(k) \tag{6.7}$$

where

$$F(k) = I - (I + D(k))^{-1} L(k) = (I + D(k))^{-1}(I + W(k)).$$

Since $\xi_i \in R^N$, we must use Kronecker product to write,

$$\xi(k+1) = (F(k) \otimes I_N)\xi(k) \tag{6.8}$$

Here, $\xi = [\xi_1^T \cdots \xi_N^T]^T \in R^{N^2}$. Note that $F(k)$ is a time-varying stochastic matrix that depends on the trust values ξ_{ij}. The matrix $F(k)$ corresponds to a time-varying trust graph $G(k)$ with Laplacian $L(k)$ $F(0)$ corresponds to the initial trust graph $G(0)$ with initial Laplacian $L(0)$. For each k, $F(k)$ has a eigenvalue of one corresponding to the right eigenvector of **1**, where **1** is a column vector with all entries equal to one. Even if $F(k)$, $F(k\text{-}1)$, $F(k\text{-}2)$,…, $F(0)$ are time-varying, the graph topology remains the same, only the weights in F change, which we prove in Section 6.3.

6.3 Convergence of Trust

We say that a protocol achieves (asymptotic) consensus if for every i, j one has $\xi_i(t) \to \xi_j(t) \to \xi_*$ in continuous-time, $\xi_i(k) \to \xi_j(k) \to \xi_*$ in discrete-time, where ξ_* is called the consensus trust vector value. If this occurs, then in the limit one has $\xi_{ip} = \xi_{jp}$ for all i, j so that all nodes arrive at the same trust value for each other at node p.

The main result of this paper is that the bilinear trust protocol (6.5) for continuous-time and (6.8) for discrete-time achieve asymptotic consensus for a trust graph G if and only if the initial trust graph $G(0)$ has a spanning tree . We are of course inspired by [59], which covers the case of linear integrator dynamics.

Two nonnegative matrices are said to be of the same type if their zero elements are in the same locations [59]. We will use the notation $P \sim Q$ to denote that P and Q

are of the same type. Two graphs on the same nodes are of the same type if their edge sets are the same.

6.3.1 Consensus in the Discrete-time Scheme

In this section, we prove that the trust consensus scheme in Equation (6.8) achieves asymptotic consensus for a trust graph G if the graph has a spanning tree. For each $F(k)$ associate a set of graphs $\{G(k)\}$. Now, F is a time-varying function of the trusts with the initial trusts $\xi_i(0)$ in [0, 1]. Consider the local voting discrete time trust consensus scheme based on the Vicsek model in Equation (6.7). Let $F(0)$ represent the initial directed graph $G(0)$. If $\xi_{ij}(0)$ is an edge in $G(0)$ then $\xi_{ij}(k)$ is an edge for all $G(k)$, for $k \geq 0$. This is formalized in the next result.

Lemma 6.1: Consider a network with initial graph $G(0)$ running the discrete-time consensus scheme in (6.8) with initial condition $\xi(0)$. Let $\xi_{ij}(k) > 0$ for some time instant $k \geq 0$. Then $\xi_{ij}(k+1) > 0$. As a result, $G(k)$ for $k \geq 0$ are all of the same type.

Proof: From Equation (6.8), each updated node trust is a weighted average of its neighboring trust values such that the weights are nonnegative and less than 1, because the row sum of $F(k)$ and $F(k) \otimes I_N$ is 1, i.e. they are stochastic. Equation (6.8) can be rewritten for each state as,

$$\xi_{ij}(k+1) = \sum_l f_{il}(k)\xi_{lj}(k)$$

$$= f_{ii}(k)\xi_{ij}(k) + \sum_{l \neq i} f_{il}(k)\xi_{lj}(k)$$

where $f_{ij}(k)$ is the $(i,j)^{th}$ element of $F(k)$. Then by definition of $F(k)$, we know that, $0 \le f_{ij}(k) < 1$, for $i \ne j$ and $0 < f_{ii}(k) \le 1$. Also, $f_{ii} = \frac{1}{1+n_i} > 0$. Hence, if $\xi_{ij}(k) > 0$, the first term is always positive. The second term is a weighted average which once again is always nonnegative for non-zero initial trusts. Therefore, for $k \ge 0$, if $\xi_{ij}(k) > 0$, $\xi_{ij}(k+1) > 0$.

Thus, if $\xi_{ij}(0) > 0$ is an edge weight for $G(0)$, then $\xi_{ij}(k) > 0, \forall k > 0$ is an edge weight for $G(k)$. Therefore, $G(k)$, $\forall k \ge 0$ are all of the same type. ■

Theorem 6.1: The discrete time trust consensus scheme in Equation (6.8) achieves a trust consensus for $\xi_{ij}(k)$ if and only if the initial graph $G(0)$ has a spanning tree.

Proof: Now $G(0)$ has a spanning tree if and only if $G(k)$, $\forall k > 0$, has a spanning tree by Lemma 6.1. This is a necessary and sufficient condition for the union of graphs over any finite time interval to have a joint spanning tree. Therefore, the result (Theorem 3.8) in [59] and convergence of SIA from [91] proves the result. ■

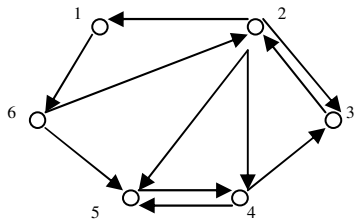


Figure 6.1 A Six Node Directed Graph

Example 6.1: Consider a six node network as shown in Figure 6.1. Let the initial $\xi(0) \in R^6$ selected randomly in [0, 1].

Figure 6.2 shows convergence of trust in a six node network with 6 states using the discrete time scheme given by Equation (6.8). ■

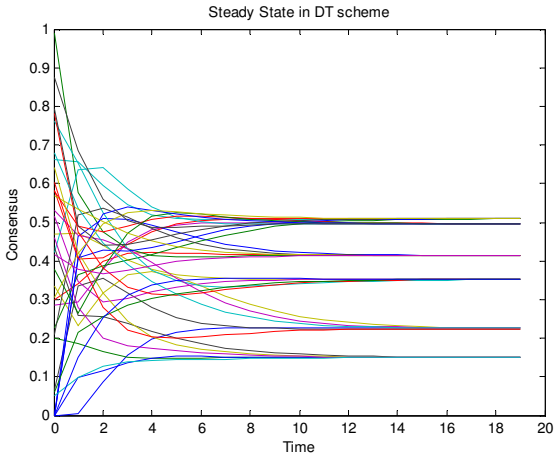


Figure 6.2 Trust Consensus in the Discrete Time Scheme

6.3.2 Consensus in the Continuous Time Scheme

In this section, we prove that the trust consensus scheme in Equation (6.5) achieves asymptotic consensus for a trust graph G if the graph has a spanning tree. For each $L(t)$ associate a set of graphs $\{G(t)\}$. For the continuous time scheme, let $L(t)=[l_{ij}(t)]$, $l_{ij} \geq 0$, $i \neq j$ and $\sum_j l_{ij} = 0$. Let $\varphi(t,t_0)$ be the corresponding transition matrix of $L(t)$ and is defined as $\phi(t,t_0) = I + \int_{t_0}^{t} L(\sigma 1)d\sigma 1 + \int_{t_0}^{t} L(\sigma 1)\int_{t_0}^{\sigma 1} L(\sigma 2)d\sigma 2 d\sigma 1 + \ldots$ From [59], we know that the transition matrix $\phi_L(t,t_0)$ of $L(t)$ is a nonnegative stochastic matrix with positive diagonal elements. Also, the corresponding transition matrix of $L(t) \otimes I_n$ is $\phi_L(t,t_0) \otimes I_N$ which is once again a nonnegative stochastic matrix with

positive diagonal entries. In the same lines as in Lemma 6.1, we can prove the following Lemma.

Lemma 6.2: Consider a network with initial graph $G(0)$ running the continuous time consensus scheme in (6.5) with initial condition $\xi(0)$. Let $\xi_{ij}(0) > 0$. Then for $\forall\ t > 0$, $\xi_{ij}(t) > 0$. As a result, $G(t)$ for $t \geq 0$ are all of the same type.

Proof: Solution of Equation (6.5) can be written as $\xi(t) = (\phi_L(t,0) \otimes I_N)\xi(0)$. This can be rewritten for each state as,

$$\xi_{ij}(t) = \phi_{Lii}(t,0)\xi_{ij}(0) + \sum_{l \neq i} \phi_{Lil}(t,0)\xi_{lj}(0) \tag{6.9}$$

Here, the diagonal elements of $\phi_L(t,0) \otimes I_N$ are always positive and therefore the first term in the RHS of Equation (6.9) will always be positive for $\xi_{ij}(0) > 0$. The second term in the RHS of Equation (6.9) is always nonnegative since $\phi_L(t,0) \otimes I_N$ is a nonnegative stochastic matrix with positive diagonal entries. Thus, if $\xi_{ij}(0) > 0$ is an edge weight for $G(0)$, then $\xi_{ij}(t) > 0, \forall\ t > 0$ is an edge weight for $G(t)$. Therefore, $G(t)$, $\forall\ t \geq 0$ are all of the same type. ■

Theorem 6.2: The continuous time trust consensus scheme in Equation (6.5) achieves trust consensus for $\xi_{ij}\ (t)$ if and only if the initial graph $G(0)$ has a spanning tree.

Proof: Now $G(0)$ has a spanning tree if and only if $G(t)$, $\forall\, t$ has a spanning tree by Lemma 6.2. Also $\phi_L(t,0)$ is a continuous function of $L(t)$ for the interval [0, t]. This is a necessary and sufficient condition for the union of graphs over any finite time interval

to have a joint spanning tree. Therefore, the result (Theorem 3.2) in [60] proves the result. ■

Example 6.2: Consider the same six node network as shown in Figure 6.1. Let the initial $\xi(0) \in R^6$ be the same as in Example 6.1. Figure 6.3 shows convergence of trust in a six node network with six states using the continuous time scheme given by Equation (6.5). It can be observed that the discrete time and the continuous time schemes give different consensus values for the same initial conditions. ■

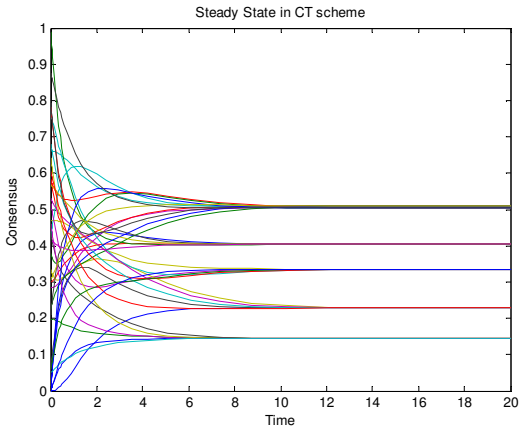


Figure 6.3 Trust Consensus in the Continuous Time Scheme

6.3.3 Relation of the Continuous & the Discrete-time Protocols

The Laplacian L in the continuous time scheme is related to the stochastic matrix F in the discrete time scheme at each time instance. As shown in Figures 6.2 and 6.3, the trust consensus using (6.8) and (6.5) do not converge to the same consensus. This is because the graph represented by $F(k)$ is not the same as the graph represented by $L(t)$. In fact,

$$F = I-(I+D)^{-1}L \qquad (6.10)$$

It can be seen that the discrete time consensus scheme is the first order Euler approximation of the continuous time scheme given by,

$$\dot{\xi}_i = -(I + D)^{-1} L \xi_i \tag{6.11}$$

If this scheme is used, both the continuous time in (6.11) and the discrete time scheme in (6.8) would approximately converge to the same consensus. See Figures 6.4 and 6.5. Here for the same network in Figure 6.2, initial trusts $\xi(0) \in R^6$ are selected randomly in [0, 1].

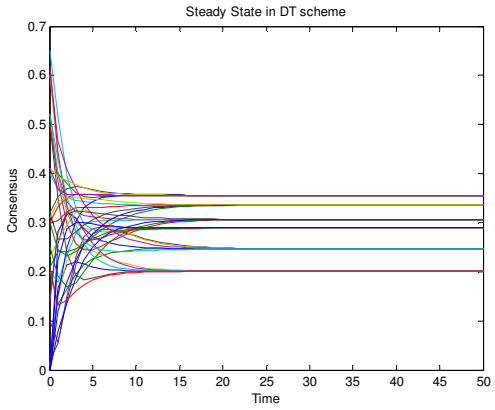


Figure 6.4 Trust Consensus in the Discrete Time Scheme

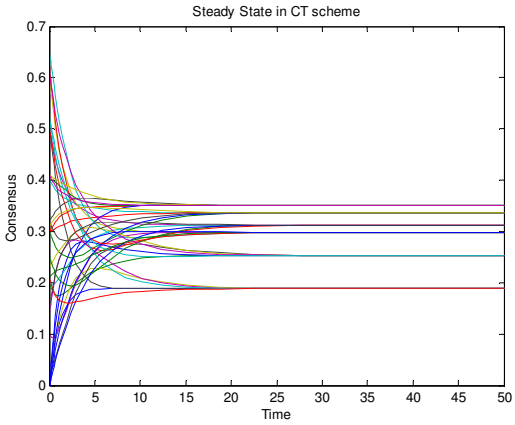


Figure 6.5 Trust Consensus in the Continuous Time Scheme using scheme (6.11)

6.3.4 Network Containing Distrusted Nodes

Attacks on the security of networked systems can often be characterized in terms of malicious nodes [38, 86]. To characterize malicious nodes, negative trust values are needed. For the case that trust is in [-1, 1] the situation is more realistic, interesting, and complex. One can now talk about identifying and pruning out malicious nodes. Here, 1 means complete trust, 0 means no opinion, and -1 means complete distrust. When trust can take on negative values, many philosophical issues arise about how to propagate distrust. i.e. if there are two negative trusts along a trust path, do negative values multiply? This would mean that the enemy of one's enemy is one's friend. One could, for instance, take here the 'one step distrust' paradigm from [38]. This means that if one has a distrusted neighbor, then one discounts all his opinions about any other nodes. Since we take the trust values as the graph edge weights, the edge weights can become negative as trust consensus propagates. When this happens, we cut the edges to isolate the distrusted neighbor. Figure 6.6(a) shows the 'one step distrust model' for a tree network.

Another method to account for negative trusts is to use the 'Graph Pruning and Reconnection' method. A simple algorithm to strip out distrusted nodes and reconnect the trusted nodes around it is shown in Figure 6.6(b). In this algorithm, after a malicious node is pruned, its neighbors are fully reconnected along all the cut paths. This means that if the pruned node has in-degree of m and out-degree of n, then mn new edges are added. This method has in fact been well studied in graph theory.

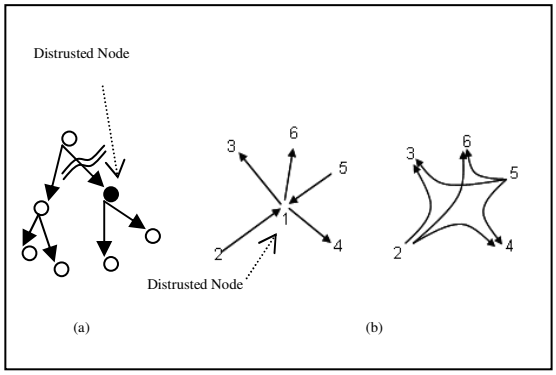


Figure 6.6 One Step Distrust Model for negative trusts (a), Graph Pruning to remove the distrusted node (b)

6.4. Team Behaviors Based on Trust

Different team behaviors will emerge automatically depending on the trust each node has for its neighbors, e.g. flock [43, 84, 85], or swarm [24, 25] with trusted neighbors, follow trusted leader, avoid enemy node. In this section we explore flocking behavior in a distributed network of agents.

6.4.1 Flocking in a Network of Trusted Nodes

The flocking model consists of three steering behaviors which describe how an individual agent maneuvers based on the positions and velocities of the neighboring flock-mates (Reynolds' rules [63]):

1. Separation: steer to avoid closely located flock-mates.
2. Alignment: steer towards the average heading of local flock-mates.
3. Cohesion: steer to move toward the average position of local flock-mates.

The superposition of these three rules results in all agents moving in a formation [8, 16], with a common heading while avoiding collisions. Generalizations of this model include a leader follower strategy, in which one agent acts as the group leader and the other agents would just follow the aforementioned rules, resulting in leader following.

Consider the node dynamics having local rule,

$$\dot{x}_i = \sum_{j \in N_i^c} w_{ij} \xi_{ij} (x_j - x_i) \tag{6.12}$$

with w_{ij} some control graph edge weights (control gains) and N_i^c the control neighborhood of node *i.* Suppose the trust of node *i* for node *j* satisfies the bilinear trust local voting dynamics,

$$\dot{\xi}_i = \sum_{j \in N_i^t} a_{ij} \xi_{ij} (\xi_j - \xi_i) \tag{6.13}$$

with N_i^t the trust neighborhood of node *i.* Note that this is a coupled system.

Example 6.3: Let x_i represent the heading of node *i* in a formation. Consider the formation graph shown in Figure 6.7. First we run the trust update protocol above on the case of fully trusted nodes. That is, the initial trust vectors $\xi_i(0)$ of the nodes have all entries positive or zero. Then, as the trusts change, the edge weights change but stay positive, so the graph structure is preserved. Then, all nodes converge to the initial heading value $x_1(0)$ of the leader.

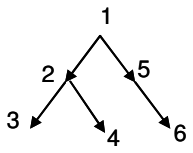
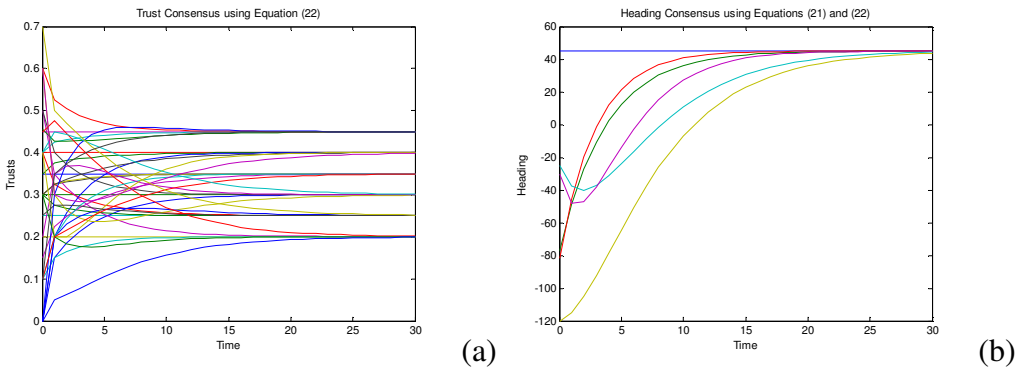


Figure 6.7 Tree network with one leader and five followers

Let the initial $\xi(0) \in R^6$ selected randomly in [0, 1]. Figure 6.8(a) shows that the trusts of the followers converge to the initial trusts of the leader node. Let the heading of each node be $\theta \in R^1$. Figure 6.8(b) shows the heading consensus in this network. Here, the heading of the followers converge to the heading of the leader. Note that even if the node headings are negative, the network heading will converge since $\xi(0)$ is in [0, 1].



(a) (b)

Figure 6.8 Convergence of trusts of all the nodes (a), Convergence of headings of all the nodes in a tree network (b)

Figure 6.9 shows the motion of each node with the follower node headings converging to the heading of the leader node. Here the velocity of each node is considered to be the same. ■

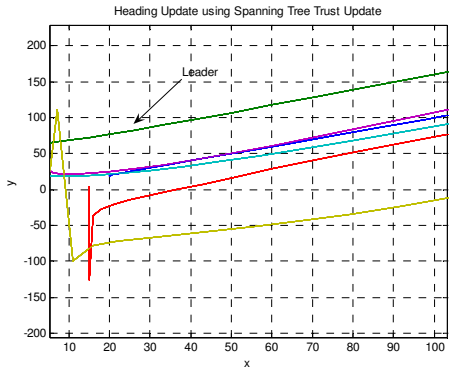


Figure 6.9 Convergence of headings of all the nodes in a tree network

6.4.2 Flocking in a Network Containing Distrusted Nodes

When the trust consensus scheme is used for trust values in [-1, 1], we make use of methods described in Section 6.3.4 to either disconnect the distrusted node or use pruning and reconnection.

Example 6.4: Pruning Distrusted Node in Flocking

Here we consider one distrusted node i.e. node 5. Specifically, the initial trust vectors $\xi_i(0)$ of the nodes have most entries positive or zero, but several of nodes have negative initial trust values $\xi_{i5}(0)$ for node 5. If the bilinear trust update algorithm is run on this graph, the trust consensus for node 5 eventually becomes negative, and its edge is cut from the group. Its follower (node 6) follows it. Neither reaches the leader's consensus heading. This is shown in Figure 6.10. ■

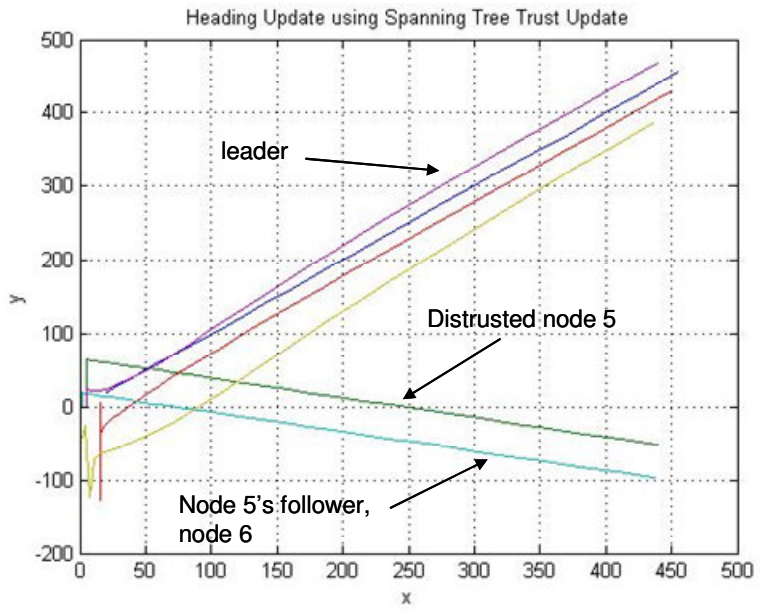
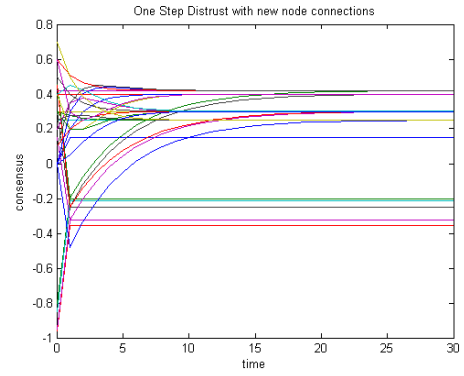


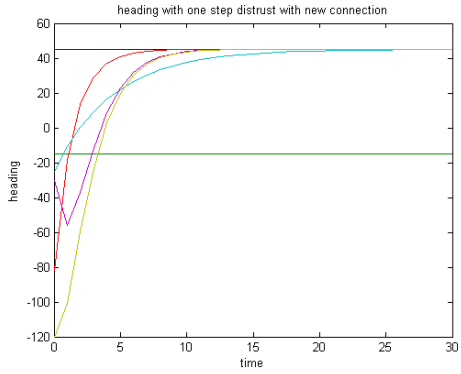
Figure 6.10 Pruning malicious node 5. Its follower 6 is also pruned.

Example 6.5: Pruning and Reconnection

In this example we consider that, though node 5 eventually becomes distrusted, all nodes in fact continue to trust node 6. The bilinear trust consensus scheme is run again, but when node 5 is pruned out, its follower node 6 is reconnected to the leader node 1. Now, all nodes except node 5 achieve the consensus heading. Figure 6.11(a) shows the convergence of trusts and Figure 6.11(b) shows the convergence of the headings. It can be seen that the heading of node 6 converges to the heading of the leader when network reconnection is used. Figure 6.12 shows the motion of each node with the follower node headings converging to the heading of the leader node while node 5 remains completely disconnected and never converges to the heading of the leader. ■



(a)



(b)

Figure 6.11 Convergence of trusts of all the nodes (a), Convergence of headings of all the nodes in a tree network after pruning and reconnection (b)

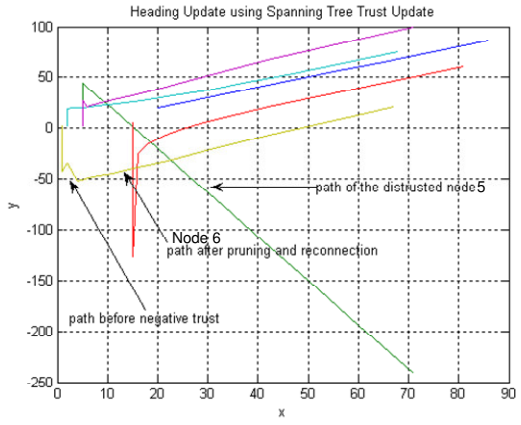


Figure 6.12 Pruning malicious node 5 with reconnection of its trusted follower 6.

6.4.3 Formations in a Distributed Network

Formation of autonomous vehicles refers to a set of spatially distributed vehicles whose dynamic states are coupled through a common control law. Following shows an easier way to maintain formations in a desired configuration. Moreover, as the desired configuration changes, the formation can quickly be moved into the new desired structure.

Consider the following dynamics,

$$x\ (k+1) = (F\ (k) \otimes I_3)\, x\ (k) \tag{6.14}$$

$$\xi\ (k+1) = (F\ (k) \otimes I_N)\, \xi\ (k) \tag{6.15}$$

Here $x = [(x_1^d)^T\ (x_2^d)^T\ ...(x_N^d)^T]$ with $x_i^d \in R^3$ the desired (x,y,z) position of node i in the formation with respect to the leader. All other nodes take their initial states as their own actual initial positions. Also note that F is a time-varying function of ξ. For a tree structure with leader as the root node, the consensus value assuming constant trust values is given by $x_{ss} = x_1(0)$.

Example 6.6: For the same tree network in Example 6.5, we want the desired positions of the nodes in the hexagonal formation structure. Let the initial state leader contain the desired formation positions. If we run the above coupled node dynamics and bilinear trust update, all nodes converge to the initial state of the leader, i.e. to their desired formation positions as shown in Figure 6.13. If the desired relative positions of all or some of the nodes change, then the leader simply resets $x_{ss} = x_1(0)$, and all nodes will

automatically converge to the new consensus trust and positions, as specified by the leader in its initial state vector. ■

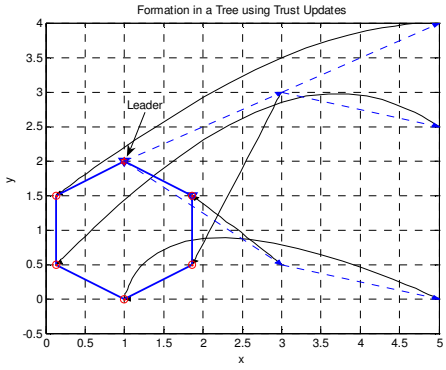


Figure 6.13 Convergence of positions of all the nodes in a tree network to a hexagon formation

Chapter 7 concludes the work in this dissertation with some ideas on future work.

CHAPTER 7

CONCLUSION

7.1 Conclusion

This dissertation presented novel matrix-based methods for decision and control in distributed cooperative systems. A novel matrix-based Discrete Event Controller has been implemented for task planning and resource dispatching in a distributed network consisting of stationary ground sensors and mobile agents.

A new matrix-based algorithm has been developed and implemented for deadlock avoidance in systems consisting of shared resources and utilizing dynamic resource assignment. The analysis of deadlock is based on objects called critical siphons and critical subsystems in such systems.

The analysis of deadlock avoidance becomes even more difficult when routing of tasks and resources are involved. The critical siphons and critical subsystems have to be redefined. This dissertation presented a new matrix-based approach for deadlock avoidance in such systems. This work proved that the new approach is a generalized approach that can be used for systems with or without routing. This work also presented a method for tackling a certain pathological conditions called second order deadlocks using a regularity test.

In the presence of numerous agents in a distributed network, a collective decision can be made based on the individual decisions of agents which is called data

fusion. Dempster Shafer (DS) theory has been extensively used in the past for data fusion since it provides an excellent framework for conditions involving uncertainty. This dissertation provided a new matrix formulation for updating evidence and computing beliefs and plausibilities in DS theory. The work also showed how evidence theory can be used in routing systems with a case study on Condition Based Maintenance.

Finally, this work presented a framework for trust propagation and maintenance in a network of nodes or mobile agents that yields global consensus of trust under rich enough communication structure graphs. This work considered the case where the graph structure is a time-varying function of the trusts based on the graph connectivity which makes the trust consensus scheme bilinear. This trust consensus is incorporated into cooperative control laws that depend on local information from neighboring nodes, yet yield team-wide desired behavior such as flocking and formations.

7.2 Future Research

This research considered the problem of deadlocks in MRF and FMRF systems. There are other types of distributed cooperative systems having different structural properties. FMRF is a general form of MRF systems. The FMRF structure differs from PN structures such as ES3PR (Extended S3PR) or S3PGR2 (System of Simple Sequential Processes with General Resource Requirements). Such structures make use of more than one resource for a given task. Deadlock prevention in such systems is well studied. We need to investigate deadlock avoidance algorithms in such systems.

In the case of trust propagation and maintenance in a distributed network, the final consensus value of all the nodes is well defined for graphs that are strongly connected and balanced with constant edge weights. For a weaker case such as a spanning tree, the final consensus value is not known even if the convergence of trusts has been proven.

The bilinear trust propagation scheme mentioned in this dissertation is more flexible than the constant edge weight case, which assumes that each node continues to put the same weight on the opinions of its neighbors even though its trust about those neighbors changes. However, analysis is a bit more tricky and interesting. In fact, note that the steady-state or consensus values of trust obey the quadratic equation $0 = -(L(t) \otimes I_N)\xi$. It would be interesting to investigate the final consensus value of trusts for different graph structures using the bilinear trust consensus scheme.

The convergence rate of trust consensus protocols depends on the Fiedler eigenvalues [52]. Figure 7.1 shows the Fiedler eigenvalues for continuous and discrete-time cases. Note that the continuous-time trust update scheme converges faster than the discrete-time scheme. This has to do with the way the Fiedler eigenvalue $\lambda_2(L)$ maps to the Fiedler eigenvalue $\lambda_2(F)$. Note further that the analysis is complicated by the fact that *L* and *F* are both time-varying.

Figure 7.1 shows the Fielder eigenvalue of *F*(*k*) for the discrete-time trust update (figure 7.1 (a)) and *L*(*t*) for the continuous time update (figure 7.1 (b)) for the examples 6.1 and 6.2. Also shown in figure 7.1 (a) is the Fiedler eigenvalue of *L*(*k*) in discrete-

time for example 6.1. Interestingly, the Fiedler eigenvalues of $L(k)$ and $L(t)$ converge to the same value.

It is interesting that the small-world model [38] has a Fiedler eigenvalue that decays quickly, so that information propagates quickly in this model and consensus is quickly achieved.

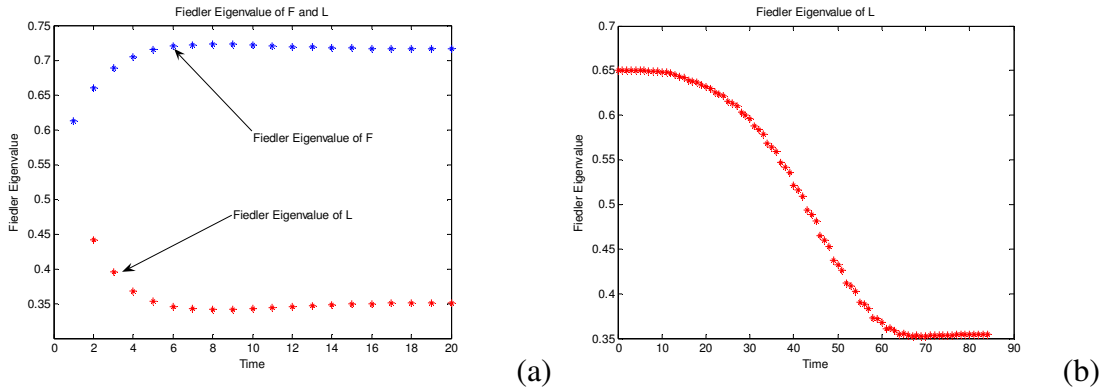


Figure 7.1 Fiedler Eigenvalues of F and L, Discrete time case (a), Continuous time case (b)

Baras et al., [86] study convergence and convergence rates in trust propagation using the Fiedler eigenvalue, and show means to speed up the convergence rate. It is interesting to study the design of good communication graph structures for trust in networked teams such that the Fiedler eigenvalue has the fastest decay rate.

REFERENCES

[1] Ballal P., Giordano V., Lewis F., "Deadlock Free Dynamic Resource Assignment in Multi-Robot Systems with Multiple Missions: A Matrix-Based Approach," Proceedings of Mediterranean Conference on Control & Automation, Ancona, Italy, June 2006.

[2] Ballal P., Lewis F., Mireles J., Sreenath K., "Deadlock avoidance for free choice multi-reentrant flow lines: Critical siphons and critical subsystems", Proc. Mediterranean Conf. Control & Automation, Athens, Greece, June 2007.

[3] Beard R., and Stepanyan V., " Synchronization of information in distributed multiple vehicle coordinated control," in *Proc. IEEE Conf. Decision and Control*, Maui, HI, Dec. 2003, pp. 2029-2034.

[4] Baccelli F., Foss S., and Gaujal B., "Free-choice Petri nets—An algebraic approach," IEEE Trans. Automat. Contr., vol. 41, pp. 1751–1778,Dec. 1996.

[5] Bengtsson M., "Condition Based Maintenance System Technology – Where is Development Heading?" In proceedings of the 17th European Maintenance Congress, May 11-13, 2004, AMS (Spanish Maintenance Society), Barcelona, Spain, B-19.580-2004.

[6] Butler Z., Rus D., "Event-based motion control for mobile-sensor network", IEEE Transactions on Pervasive Computing, vol.2 issue 4, October-December 2003

[7] Chong C., Kumar S., “Sensor Networks: Evolution, Opportunities and Challenges”, Proceedings of the IEEE, col. 91, no.8, August 2003

[8] Chopra N., Spong M., “Passivity-based control of multi-agent systems,” in “Advances in Robot Control: From Everyday Physics to Human-Like Movements,” ed. S. Kawamura and M. Svinin, pp. 107-134, Springer-Verlag, Berlin, 2006.

[9] Corman, T., Leisenson, C., and Rivest R., “Introduction to Algorithms,” Prentice Hall of India, 2001.

[10] Cortes J., Martinez S., Karatas T., Bullo F., “Coverage control for mobile sensing network”, IEEE Transactions on Robotics and Automation, Vol. 20, Issue 2, pp. 243–255, April 2004

[11] Dantu K., Rahimi M., Shah H., Babel S., Dhariwal A., Sukhatme G., “Robomote: enabling mobility in sensor networks”, Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on 15 April 2005 Page(s):404 – 409

[12] Davey A., Grosvenor R., Morgan P., Prickett P., "Petri-net based machine tool failure diagnostics", in Rao, R.B.K.N (Eds),Condition Monitoring and Diagnostic Engineering Management, Sheffield Academic Press, pp.722-3, 1996.

[13] Desel J., Esparza J., “Free choice petri nets,” Cambridge Tracts in Theoretical Computer Science, vol. 40, Cambridge University Press, Cambridge, UK, 1995.

[14] Dempster A., “Upper and lower probabilities induced by a multiple valued mapping, Ann. Math. Statist. 38 (1967) 325–339.

[15]Dempster A., A generalization of Bayesian inference, J. Roy. Statist. Soc. B 30 (1968) 205–247.

[16]Dunbar W., and Murray R., "Distributed receding horizon control for multi-vehicle formation stabilization," Automatica, vol. 42, pp. 549-558, 2006.

[17]Ezpeleta, J., Colom J.M., Martinez J., "A Petri Net Based Deadlock Prevention Policy for Flexible Manufacturing Systems," IEEE Transaction on Robotics and Automation, Volume 11, Issue 2, April 1995 Page(s):173 – 184.

[18]Ezpeleta J., Recalde L., "A deadlock avoidance approach for nonsequential resource allocation systems," in Proc. IEEE Int Conf. Syst., Man, Cybern., Oct. 2002.

[19]Fanti M. P., Maione G., and Turchiano B., "Distributed Event-Control for Deadlock Avoidance in Automated Manufacturing Systems," International Journal of Production Research 39(9), 2001, Page(s): 1993-2021.

[20]Fanti M.P., Maione B., Mascolo S., Turchiano B., "Event-based feedback control for deadlock avoidance in flexible production systems," IEEE Transactions on Robotics and Automation, Volume 13, Issue 3, June 1997 Page(s):347 - 363

[21]Fanti, M.P., Maione B., Turchiano B., "Comparing digraph and Petri net approaches to deadlock avoidance in FMS," IEEE Transactions on Systems, Man and Cybernetics, Part B, Volume 30, Issue 5, Oct. 2000 Page(s):783 - 798

[22]Fanti M., Zhou M., "Deadlock control methods in automated manufacturing systems," IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 34(1):5-22, 2004.

[23]Fax J., and Murray R., “Information flow and cooperative control of vehicle formations,” IEEE Trans. Automatic Control, vol. 49, no. 9, pp. 1465-1476, Sept. 2004.

[24]Gazi V., and Passino K., “Stability analysis of swarms,” IEEE Trans. Automatic Control, vol. 48, no. 4, pp. 692-697, April 2003.

[25]Gazi V., and Passino K., “A class of attractions/repulsion functions for stable swarm aggregations,” Int. J. Control, vol. 77, no. 18, pp. 1567-1579, 2004.

[26]Gerkey B., Mataric M., “Sold! Auction methods for multirobot coordination”, IEEE Transactions on Robotics and Autoation, vol. 18, no. 5, October 2002

[27]Gerkey B., Mataric M., “A formal analysis and taxonomy of task allocation in multi-robot systems”, Int. Journal of Robotics Research, vol.:18 , no.5 , September 2004

[28]Giordano V., Lewis F., Mireles J., Turchiano B., “Coordination Control Policy for Mobile Sensor Networks with Shared Heterogeneous Resources,” Proceedings of the IEEE International Conference on Control and Automation, Budapest, June 2005.

[29]Giordano V., Ballal P., Lewis F., Turchiano B., Zhang J. B., “Supervisory Control of Mobile Sensor Networks: Math Formulation, Simulation, Implementation,” IEEE Transactions on Systems, Man and Cybernetics, Part B, Volume 36, Issue 4, Aug. 2006 Page(s):806 – 819.

[30]Giordano V., Lewis F., Turchaino B., Ballal P., Yeshala V., “Matrix Computational Framework for Discrete Event Control of Wireless Sensor Networks with Some

Mobile Agents," Proceedings of the Mediterranean Conference on Control & Automation, Limassol, Cyprus, June 2005.

[31]Godsil C., and Royle G., "Algebraic Graph Theory," *Springer Graduate Texts in Mathematics*, no. 207, New York, 2001.

[32]Gurel, A., Bogdan S., Lewis F., " Matrix Approach to Deadlock-Free Dispatching in Multi-Class Finite Buffer Flowlines," IEEE Transactions on Automatic Control, Volume 45, Issue 11, Nov 2000 Page(s): 2086-2090.

[33]Herstein I. N., "Topics in Algebra," Blaisdell Publishing Co., 1964.

[34]Horn R., and Johnson C., "Matrix Analysis," Cambridge, UK., Cambridge Univ. Press, 1985.

[35]Huang Y., Jeng M., Xie X., Chung D., "Siphon-Based Deadlock Prevention Policy for Flexible Manufacturing Systems", IEEE Trans. On Systems, Man, Cybernetics, Part A, Vol 36, Issue 6, Nov, 2006.

[36]Huang Y., Lin H., Lin J., "A Siphon-Based Deadlock Prevention Policy for FMS", IEEE Conf. on Systems, Man and Cybernetics, 2005.

[37]Jadbabaie A., Lin J., and Morse A., "Coordination of groups of mobile autonomous agents using nearest neighbor rules," IEEE Trans. Automatic Control, vol. 48, no. 6, pp. 988-1001, June 2003.

[38]Jiang T., and Baras J., "Trust evaluation in anarchy: a case study on autonomous networks," Proc. Infocom, Barcelona, 2006.

[39]Judd R., Zhang W., Deering P., Lipset R., “A Scalable Deadlock Avoidance Algorithm for Flexible Manufacturing Systems with Free Choice in Part Routing,” Proceedings of IEEE American Control Conference, 2000.

[40]King J., Pretty R., Gosine R., “Coordinated execution of tasks in multiagent environment”, IEEE Transactions on Systems, Man and Cybernetics- Part A: Systems and Humans, Vol. 33, Issue 5, pp. 615–619, September 2003.

[41]Kusiak A., “Intelligent scheduling of automated machining systems,” in Intelligent design and Manufacturing. A. Kusiak (ed.) Wiley, New York (1992).

[42]Lawley M., “Integrating Routing Flexibility and Algebraic Deadlock Avoidance Policies in Automated Manufacturing Systems,” International Journal of Production Research, , 2000, Page(s): 2931-2950.

[43]Lee D., and Spong M., “Stable flocking of multiple inertial agents on balanced graphs,” preprint, 2007.

[44]Li Z., Zhou M., “Elementary Siphons of Petri Nets and their Application to Deadlock Prevention in Flexible Manufacturing Systems”, IEEE Trans. On Systems, Man, Cybernetics, Part A, Vol 34, Issue 1, Jan 2004.

[45]Li Z., Wei N., “An Improved Deadlock Control Policy using Elementary Siphons and MIP Approach”, IEEE conf. on Industrial Technology, ICIT 2005.

[46]Li Z., Wei N., Zhu R., “ Deadlock Prevention Policy for FMS using Petri Nets”, ICCA 2005.

[47]Lewis F., Gurel A., Bogdan S., Docanalp A. Pastravanu O., "Analysis of Deadlock and Circular Waits Using a Matrix Model for Flexible Manufacturing Systems," Automatica, vol.34, no. 9, September 1998.

[48]Lewis F., "Wireless sensor networks," Smart environments: Technologies, Protocols, and Applications, ed. D. J. Cook and S. K.Das, John Wiley, New York, 2004.

[49]Maione G., Naso D., "New control policies preventing deadlock in automated manufacturing systems," Proceedings of IEEE Conference on Emerging Technologies and Factory Automation, September 2003

[50]Mireles J., Lewis F., Gurel A., "Implementation of a Deadlock Avoidance Policy for Multipart Reentrant Flow Lines Using a Matrix-Based Discrete Event Controller," Proceedings of the International symposium on advances in robot dynamics and control, New Orleans, November 2002.

[51]Murata, T. "Petri Nets: Properties, Analysis and Applications," Proceedings of the IEEE, vol.77, no.4, April 1989, pp.541-80.

[52]Olfati-Saber R., and Murray R., "Consensus problems in networks of agents with switching topology and time-delays," IEEE Trans. Automatic Control, vol. 49, no. 9, pp. 1520-1533, Sept. 2004.

[53]Ozmutlu, S. And Harmonosky, C.M., "A Real Time Methodology for Minimizing Mean Flowtime in FMSs With Routing Flexibility: Threshold Based Alternate Routing," European Journal of Operational Research, 2005, Pages(s): 369-384.

[54]Park J., Reveliotis S., “Deadlock Avoidance in Sequential Resource Allocation Systems with Multiple Resource Acquisitions and Flexible Routings,” IEEE Transactions on Automatic Control, Vol. 46, 2001, pp 1572-1583.

[55]Peterson, J. L., “Petri Net Theory and the Modeling of Systems,” Prentice-Hall, Englewood Cliffs, NJ, 1981.

[56]Piroddi L., Ferrarini L., “ A Modular Approach for Deadlock Avoidance in FMS,” Proceedings of IEEE Conference on Decision and Control, 2005.

[57]Puccinelli D., Haenggi M., “Wireless Sensor Networks: Applications and Challenges of Ubiquitous Sensing”, IEEE Circuits and Systems Magazine, vol. 5, pp. 19-29, August 2005.

[58]Qiu D., “Supervisory control of fuzzy discrete event systems: A formal approach,” IEEE Trans. Syst., Man, Cybern., B, vol. 35, no. 1, pp. 72–88, Feb. 2005.

[59]Ren W., and Beard R., “Consensus seeking in multiagent systems under dynamically changing interaction topologies,” IEEE Trans. Automatic Control, vol. 50, no. 5, pp. 655-661, May 2005.

[60]Ren W., Beard R., Kingston D., “Multi-agent Kalman Consensus with Relative Uncertainty,” *Proceedings of ACC, June 2005.*.

[61]Reveliotis S., “On the Siphon-based Characterization of Liveness in Sequential Resource Allocation Systems,” Tech. Report, School of Industrial & Systems, Eng. Georgia Tech, 2001.

[62]Revliotis S., “Real-time Management of Resource Allocation Systems: A Discrete Event Systems Approach”, NY, Springer, 2005.

[63]Reynolds C., “Flocks, herds and schools: a distributed behavioral model.,” *Computer Graphics*, 1987, 2 1 (4):25-34.

[64]Saligrama V., Alanyali M., and Savas O., “Distributed detection in sensor networks with packet losses and finite capacity links,” IEEE Trans. Signal Proc., vol. 54, no. 11, pp. 4118-4132, Nov. 2006.

[65]Sinopoli B., Sharp C., Schenato L., Schaffert S., Sastry S., “Distributed Control Applications Within Sensor Networks”, Proceedings of the IEEE, vol. 91, no.8, August 2003

[66]Shafer G., “Allocation of Probability: a Theory of Partial Belief,” Ph.d. thesis, Princeton University, 1974.

[67]Shafer G., “A Mathematical Theory of Evidence,” Princeton University Press, Princeton, NJ, 1976.

[68]Smets P., “The transferable belief model for quantified belief representation,” in: D.M. Gabbay, P.Smets (Eds.), Handbook of Defeasible Reasoning and Uncertainty Management Systems, vol. 1,Kluwer, Doordrecht, The Netherlands, 1998, pp. 267–301.

[69]Smets P., “Decision making in a context where uncertainty is represented by belief functions,” in R.P. Srivastava, T. Mock (Eds.), Belief Functions in Business Decisions, Physica-Verlag, Heidelberg, Germany, 2002, pp. 17–61.

[70]Smets P., “Upper and lower probability functions versus belief functions”, Proc. International Symposium on Fuzzy Systems and Knowledge Engineering, Guangzhou, China, July 10-16, pg 17-21., 1987.

[71]Smets P., “Belief functions”, in Smets P, Mamdani A., Dubois D., and Prade H. ed. Non-standard logics for automated reasoning. Academic Press, London p 253-286. 1988.

[72]Smets P., “The combination of evidence in the transferable belief model”, IEEE-Pattern analysis and Machine Intelligence, 12:447-458. 1990

[73]Smets P., “The transferable belief model and possibility theory”, Proc. NAFIPS-90, pg. 215-218. 1990.

[74]Smets P., “Constructing the pignistic probability function in a context of uncertainty”, Uncertainty in Artificial Intelligence 5, Henrion M., Shachter R.D., Kanal L.N. and Lemmer J.F. eds, North Holland, Amsterdam, , 29-40. 1990.

[75]Smets P., “The Transferable Belief Model and Other Interpretations of Dempster-Shafer's Model”, Procs of the 6th Conf.on Uncertainty in AI, Cambridge, MA. 1990.

[76]Smets P., “Varieties of ignorance”, Information Sciences. 57-58:135-144. 1991.

[77]Smets P., “Belief functions: the disjunctive rule of combination and the generalized Bayesian theorem”, Int. J. Approximate Reasoning, 1991.

[78]Smets P., “The nature of the unnormalized beliefs encountered in the transferable belief model”, in Dubois D., Wellman M.P., d’Ambrosio B. and Smets P. Uncertainty in AI 92. Morgan Kaufmann, San Mateo, Ca, USA, 1992, pg.292-297. 1992.

[79]Smets P., “The transferable belief model and random sets”, Int. J. Intell. Systems 7:37-46. 1992.

[80]Smets P., “Decision making in the TBM: the necessity of the pignistic transformation,” Int. J. Approx. Reason. 38 (2005) 133–147.

[81]Smets P., “The application of the matrix calculus to belief functions, Int. J. Approx. Reason. 31 (2002) 1–30.

[82]Tacconi D., Lewis F., “A new matrix model for discrete event systems: application to simulation”, IEEE Control System Magazine, vol.17 October 1997

[83]Takata S., Kirnura F., Houten F., Westkamper E., Shpitalni M., Ceglarek D., Lee J., “Maintenance Changing role in Life Cycle Management,” Annals of the CIRP 53 (2004) (2), pp. 643–655.

[84]Tanner H., Jadbabaie A., and Pappas G., “Stable flocking of mobile agents, Part i: Fixed Topology,” in *Proc. IEEE Conf. Decision and Control*, Maui, HI, Dec 2003, pp. 2010-2015.

[85]Tanner H., Jadbabaie A., and Pappas G., “Stable flocking of mobile agents, Part ii: Dynamic Topology,” in *Proc. IEEE Conf. Decision and Control*, Maui, HI, Dec 2003, pp. 2016-2021.

[86]Theodorakopoulos G., and Baras J., “On trust models and trust evaluation metrics for ad hoc networks,” IEEE J. Selected Areas in Communications,” vol. 24, no. 2, pp. 318-328, Feb. 2006.

[87]Uzam M., Zhou M., “An Improved Iterative Synthesis Method for Liveness Enforcing Supervisors of FMS”, International Journal of Production Research, Vol 44, Issue 10, May 2006.

[88] Vicsek T., Czirok A., Jacob E., Cohen I., and Schochet O., “Novel type of phase transitions in a system of self—driven particles,” *Phys. Rev. Lett*., vol 75, pp.1226-1229, 1995.

[89] Viswanadham N., Narahari Y., Johnson T., " Deadlock prevention and deadlock avoidance in flexible manufacturing systems using Petri net models," IEEE Trans. on Robotics and Automation, vol. 6, pp. 713-723, Dec. 1990.

[90] Williams J. H., Davies A., Drake P.R., “Condition-Based Maintenance and Machine Diagnostics,” Chapman & Hall, 1992.

[91] Wolfowitz J., “ Products of indecomposable, aperiodic, stochastic matrices,” *Proc. Amer. Math. Soc*., vol. 15, pp. 733-736, 1963.

[92] Wysk, R.A.; Yang, N.S.; Joshi, S.; “Detection of Deadlocks in Flexible Manufacturing Cells,” IEEE Transactions on Robotics and Automation, vol.:7 , Issue: 6 , December 1991.

[93] Xing K.Y., Hu B.S., Chen H.X., “Deadlock Avoidance Policy for Petri-net Modeling of Flexible Manufacturing Systems With Shared Resources,” IEEE Transactions on Automatic Control, Volume 41, Feb 1991 Page(s): 289-295.

[94] Xing K., Jin X., Feng Y., “ Deadlock Avoidance Petri Net Controller for Manufacturing Systems with Multiple Resource Service,” Proceedings of IEEE Conference on Robotics and Automation, 2005.

[95] Yang B., Jeong S., Oh Y., Tan A., “Case-based reasoning system with Petri nets for induction motor fault diagnosis,” Expert Systems with Applications 27 (2004) 301–311.

[96] Yang, S., "A condition-based failure-prediction and processing-scheme for preventive maintenance," IEEE Transactions on Reliability, Vol. 52 No. 3, pp. 373-83, 2003.

[97] Zhou M., Fanti M., " Deadlock Resolution in Computer-Integrated Systems", Marcel Dekker, Inc, Singapore, 2004.

BIOGRAPHICAL INFORMATION

Prasanna Mohan Ballal received his Bachelor of Engineering degree in Electronics and Telecommunication from Mumbai University, India in 2002. He then worked for IndiaGames Ltd., as a software programmer. He received his Master of Science degree in Electrical Engineering in 2005 and PhD in Electrical Engineering in 2008 from University of Texas at Arlington, USA. He has been working as a Graduate Research Associate in Distributed Intelligence & Autonomy Laboratory at Automation and Robotics Research Institute, Fort Worth, USA and has authored or co-authored over 14 journal and conference publications. His research interests include Control System Design, Wireless Sensor Networks and Signal Processing.