A PETRI NET FORMULATION FOR THE GENERAL SCHEDULING PROBLEM

by

MICHAEL KENJI IMAI

B.Eng.Mgt., McMaster University, 1979

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES
(Department of Electrical Engineering)

We accept this thesis as conforming
to the required standard

UNIVERSITY OF BRITISH COLUMBIA

September, 1981

Michael K. Imai

Department of Electrical Engineering

The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date October 2, 1981

Abstract

This thesis introduces a new Petri net formulation for the general scheduling problem. The first part of this thesis concerns the development of the Petri net formulation. The Petri net formulation is a synthesis of concepts from three classes of Petri nets, marked graphs, timed Petri nets and coloured Petri nets. The general approach to the scheduling problem begins with the construction of a Petri net which models the structure of the general scheduling problem. The scheduling strategy is modeled by modifying the algorithm for the analysis of Petri nets. A schedule is generated by the execution of the Petri net model under the modified analysis.

In the second part of this thesis, The Petri net formulation is used for the analysis of a particular scheduling problem. The problem addressed is the scheduling of a task system on a set of processors of different speeds. The scheduling strategy to be analyzed is list scheduling. A new heuristic is proposed for the ordering of the tasks into a list. The proposed heuristic combines notions from the highest levels first heuristic and the longest processing time heuristic.

The performance of the proposed heuristic is evaluated by a comparision with other list ordering heuristics. The schedules which are generated by the proposed heuristic are compared to the schedules which are generated by the highest levels first heuristic, the Coffman and Graham Algorithm A and a random list for fifteen precedence constraints. The proposed heuristic generated a better schedule in 98 of 160 cases tested for the 15 precedence constraints. The proposed heuristic generated a schedule as good as the schedule which is generated by any other list in a further 39 cases.

Table of Contents

List of Figures

List of Tables

Acknowledgement

I would like to thank Dr. Mabo R. Ito for introducing both the topics involved in the reseach, Petri nets and scheduling theory. I also thank him for his guidance and encouragement thoughout the course of this work.

I gratefully acknowledge the financial support of both the National Sciences and Engineering Council and the University of British Columbia.

CHAPTER 1

INTRODUCTION

A Petri net is a formal model of information flow. Petri nets are particularly useful for the description and the analysis of systems which exhibit asynchronous concurrent behavior. Petri nets provide a natural and compact representation for asynchronous concurrent systems. The natural and compact representation is reflected in a simple graphical representation. Petri nets are easily extended so as to increase their ability to model systems. This extendibility allows compact and natural descriptions for a wide variety of systems.

An important aspect of Petri nets is the ability to model both the structure and the behavior of systems. The stucture of a system is the static properties in a system which impose restrictions on the behavior of the system. The behavior of a system is the set of actions which occur as a result of conditions in the system. The behavior is dynamic in nature since the occurrence of an action causes new conditions to become valid. The new conditions in turn allow more actions to occur. Thus, Petri nets are a powerful method of describing systems in which the structure and behavior are of equal importance.

The general scheduling problem is that of scheduling the execution of a set of tasks on a set of resources. For the purposes of this thesis, the set of resources is a computer system. The computer system is comprised of processors, primary and secondary storage, and possibly other devices. The task system then consists of blocks of independent code from a program. It is assumed the resources are sufficient for the execution of all of the tasks; that is, the minimum amount of any resource is the maximum, over all tasks, of the minimum amount required by any single task. It is assumed that a valid schedule exists. The scope of the problem is to find the best schedule given a task system, a set of resources and a scheduling strategy.

The general scheduling problem may be considered in terms of structure and behavior. The structure of the general scheduling problem is the operational precedence constraints of the task system. Further structure is imposed by the resources which are used for the execution of the task system. If we consider storage devices as a set of resources, the set of resources may be subdivided into primary and secondary storage with primary storage being the faster of the two. The secondary storage may be comprised of disk and tape drives. In either case, there are differences which impose constraints on the use of the resource of storage space. These relations between the members of a set of resources add to the structure of the general scheduling problem.

The total behavior of a system is characterised by a state space. The transitions between the states are the possible actions which may occur in a particular state. Suppose that a particular behavior is imposed on the actions of the system. As the system progresses through the state space, a certain sequence of states is followed. In terms of the general scheduling problem, the states are defined by the tasks which have completed execution, the tasks which are to be executed and the tasks which are currently executing. A schedule in this description is a sequence of states. The scheduling strategy determines which sequence of states is followed. Hence, the scheduling strategy is the behavior of the general scheduling problem.

The objective of this thesis is the development of a Petri net formulation to provide a means of studying general scheduling problems. Petri nets have the ability to model both the structure arising from the precedence relation of the task system and the structure arising from the relationships present in the resources. Petri nets also have the ability to describe the behavior of the system. The scheduling strategies to be evaluated are modeled by modifications to the Petri net analysis.

In an early paper by Shapiro and Saint [84], Petri nets are applied to a sequencing problem. The machine code for a FORTRAN DO-loop executing on a CDC 6600 is optimised using a Petri net model of the DO loop and the hardware constraints. The modeling which was presented is at a much lower level than is intended by the Petri net formulation to be presented here. The method of optimization is execution of the net until a solution which is at or near a theoretical optimal one is found.

A second use of Petri nets in a scheduling context uses a restricted class of Petri nets which cannot model conflicts [92, 93]. To resolve a conflict, a Petri net is generated for each possible resolution. Each net is then evaluated to determine the best manner in which to resolve the conflict. In both approaches the optimization is through exhaustive searches; this is not the aim of the formulation to be presented here. Both approaches are discussed in detail after the introduction of the formulation.

1.2 Overview of the Thesis

Chapter 2 is an introduction to Petri nets. The basic ideas and properties are discusssed at an informal level. The analysis of Petri nets is presented in two sections, the structural analysis and the behavioral anaylsis. Three classes of Petri nets, marked graphs, timed Petri nets and coloured Petri nets, are introduced with emphasis on the properties which are relevant to the development of the formulation in Chapter 3.

Chapter 3 deals with the definition of the Petri net formulation of the general scheduling problem. The general scheduling model upon which the formulation is based is presented. The definition of the Petri net formulation is separated into two logical sections. First, the construction of the Petri net model of the formulation is discussed. Secondly, the modeling of the behavior of the scheduling strategy by the Petri net analysis is presented. The Petri net formulation is compared in detail to previous uses of Petri nets for scheduling.

Chapter 4 presents the analysis of a specific scheduling problem to demonstrate the use of the Petri net formulation. The problem which is analyzed is the use of list scheduling on processors of different speeds. Results of the comparison of the heuristics and a discussion of the results is presented.

Chapter 5 is a summary of the significant results and conclusions. Suggestions for further research are also included.

Appendix A contains the formal definitions and the formal notation for the terms which are discussed at an informal level in this thesis.

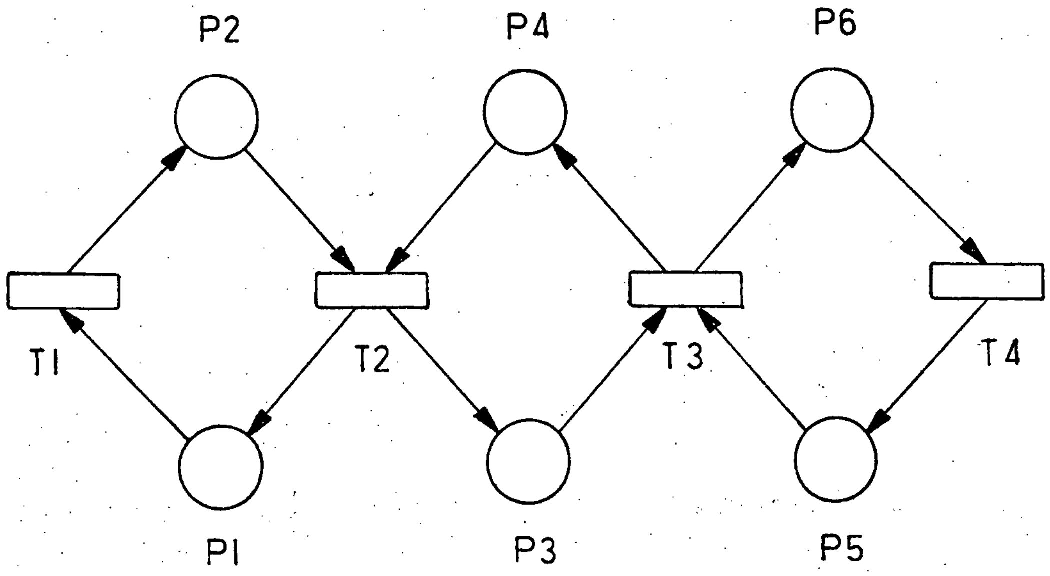
CHAPTER 2
PETRI NETS

A Petri net is a formal model for the representation of systems. Petri nets are useful for the modeling of systems in which the concurrency of actions is an important aspect of the system. In these situations, the Petri nets provide a natural representation for the system which is reflected in a simple graphical representation. The Petri net is easily extended to increase its ability to model different concurrent systems.

2.1 Basic Concepts and Ideas

A Petri net is a pair, <P,T>, where P is a set of places and T is a set of transitions [53]. A convenient means of conceptualizing a Petri net is in terms of its graphical representation, a bipartite directed graph. Figure 2.1(a) shows the graph of a simple net whose formal representation appears in Figure 2.1(b). The formal representation is defined in Appendix A. In the graphical representation, the two types of nodes, circles and rectangles, represent places and transitions, respectively.

The structural properties of the net are the relationships between the places and the transitions of the net. The relationships are the directed edges in the graphical representation of the net. A place which is connected by an



(a) Graphical Representation

P={P1,P2,P3,P4,P5,P6}

T={T1,T2,T3,T4}

T1: P1 $\rightarrow$ P2

T2: P2+P4 $\rightarrow$ P1+P3

T3: P3+P5 $\rightarrow$ P4+P6

T4: P6 $\rightarrow$ P5

(b) Formal Representation

Figure 2.1: A simple Petri net

edge from itself to a transition, is said to be an input place to that transition. A place which is connected by an edge from a transition to itself is said to be an output place of that transition. Similarily, transitions may be described in terms of inputs and outputs to places. In Figure 2.1(a), place P5 is an output place of transition T4. Places P2 and P4 are input places to transition T2.

Consider the analogy of the flowchart of a computer program. The structure of the flowchart represents the relationship between the different sections of the program. If a marker is used to trace on the flowchart the execution of the program, then, the marker traverses the flowchart as the different sections of the program are executed. The markers in a Petri net are called tokens which are held in the places of the net. The graphical representation of a token is a dot[1], as shown in Figure 2.2. A marked Petri net is a triple, <P,T,m>, where m is a marking. A marking is a distribution of tokens in the places of a net. Each marking represents a different state of the Petri net; just as in the flowchart analogy, the flowchart marker indicates the state of the computer program.

The behavioral or dynamic properties of the net are characterized by the movement of tokens in the net. Tokens move through the net by the firing of the transitions. A

[1]If a place holds several tokens it is convenient to indicate the number of tokens by the number rather than a number of dots.

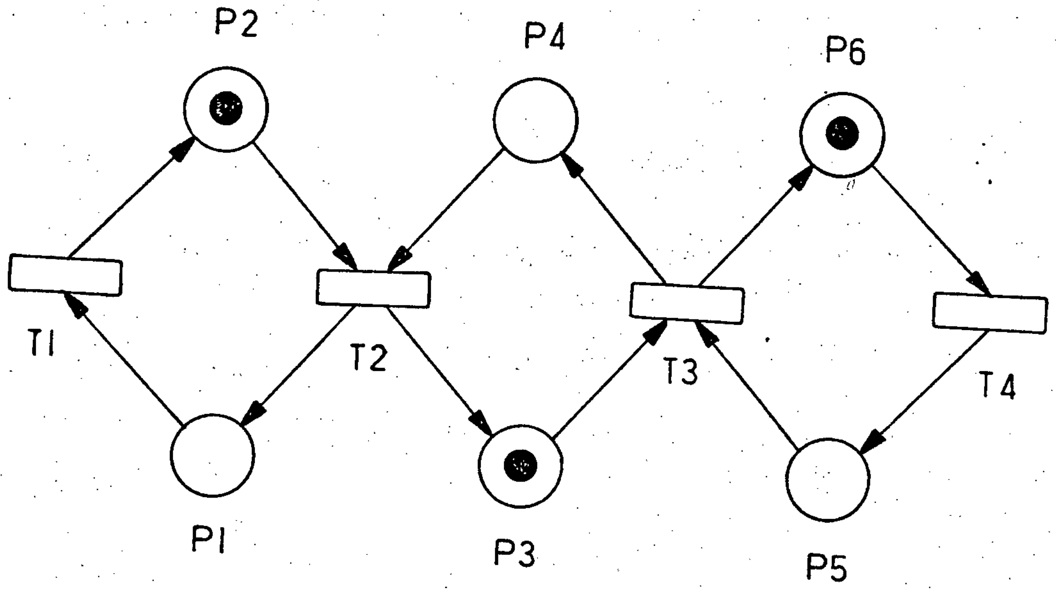


Figure 2.2: A marked Petri net

transition may fire if it is enabled in the current marking under the assumed firing rules. In the simplest firing rule, a transition is enabled when there is at least one token in each of its input places. In more complex firing rules, weights which are assigned to the edges and capacities which are assigned to places are taken into consideration when determining whether a transition is enabled.

In Figure 2.2, transition T4 is enabled under the marking shown. A transition fires by removing a token from each of its input places and depositing a token into each of its output places. The firing of a transition is assumed to occur instantaneously. The movement of tokens is shown in Figures 2.3 and 2.4 which illustrates the firing of transitions, T4 then T3, respectively, on the marked net of Figure 2.2. The execution of a net is the firing of a sequence of transitions from an initial marking of the net.

The Petri net models the structure and the behavior of systems which may be expressed in terms of conditions and events. The places represent conditions which may be present in the system and the transitions represent the events in the system which occur as a result of the conditions present. The edges indicate which conditions are necessary for the occurrence of an event and which conditions result from the occurrence of an event. Consider the following conditions and events for the net shown in Figure 2.1.

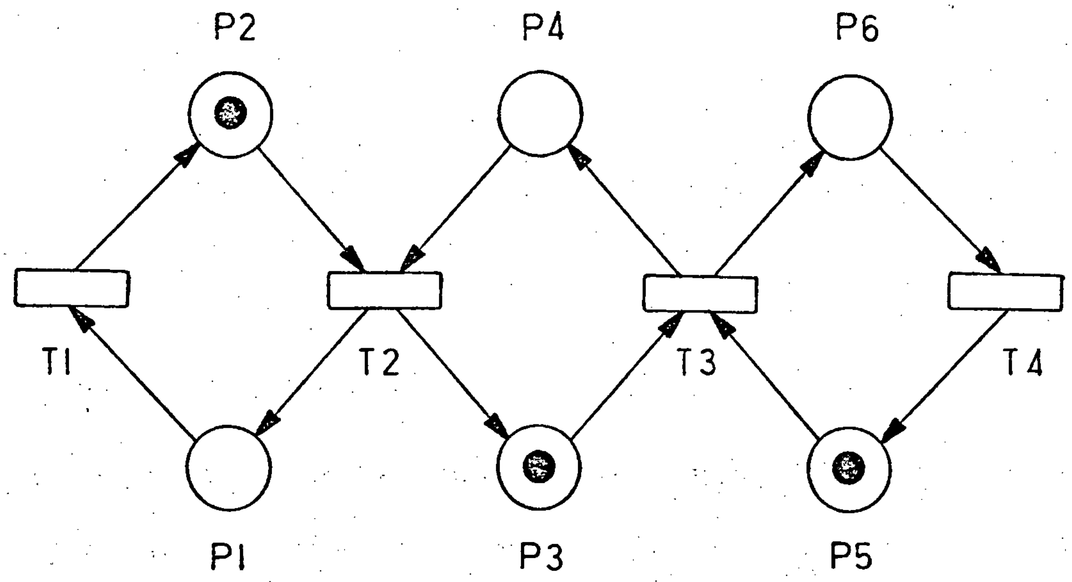


Figure 2.3: T4 fired

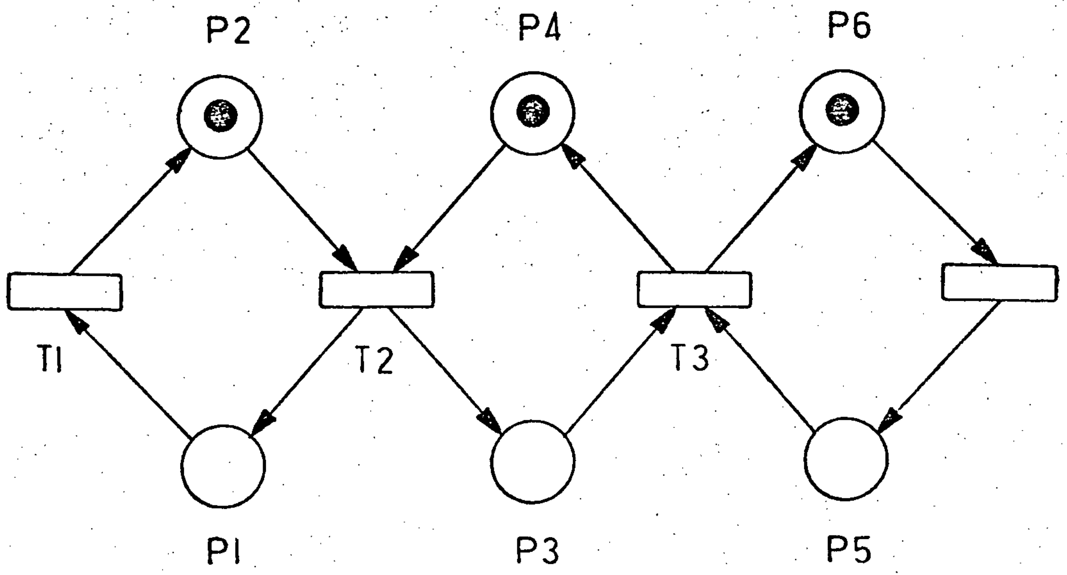


Figure 2.4: T3 fired

P1: consumer ready to consume a unit

P2: consumer ready to take a unit from buffer

P3: buffer empty

P4: buffer full

P5: producer ready to produce a unit

P6: producer ready to deposit a unit into buffer

T1: consumer consumes a unit

T2: consumer takes a unit from buffer

T3: producer deposits a unit into buffer

T4: producer produces a unit

The Petri net now models the interaction of a producer and a consumer through a bounded buffer. The marking indicate the conditions in the system which are currently valid.

The example illustrates two important ideas underlying the ability of Petri nets to model systems. First, the net correctly models the concurrency in the system. In Figure 2.4, transitions T2 and T4 are enabled under the marking. In terms of the producer-consumer system, the consumer can take a unit from the buffer or the producer can produce another unit. These actions may occur concurrently, which is modeled if both T2 and T4 fire together. The ability to model concurrency arises since the firing of a transition causes only a local change of state. Only places which are input or output to a transition are affected.

The second important idea is that the Petri net model imposes the correct sequencing of events in the system. In the example, transitions T4 and T3 are fired before T2 may fire. In terms of the producer-consumer system, the producer must produce a unit and deposit it into the buffer before the consumer can take a unit from the buffer. The transitions are sequenced since they are connected by a series of edges upon which the tokens travel. Petri nets thus have the ability to model correctly both the concurrency between events and the sequencing constraints of events of the system.

Consider the addition of a second consumer to the producer-consumer example shown in Figure 2.2. The Petri net must model the contention between the consumers when both are ready to take a unit from the buffer. The Petri net model is shown in Figure 2.5 where the additional places are the conditions and events of the second consumer. Transitions T2 and T6 share a common input place, P4, and are said to be in conflict. Under the marking shown, either T2 or T6 but not both may fire. Hence the Petri net correctly models the contention for the unit in the buffer. Conflict is a structural property which exists as a result of the relationship between related events.

The behavior of a system is characterized by the movement of tokens in the Petri net model of the system. Properties of the behavior of the system can be inferred from the behavioral properties of the Petri net. Three such properties of Petri

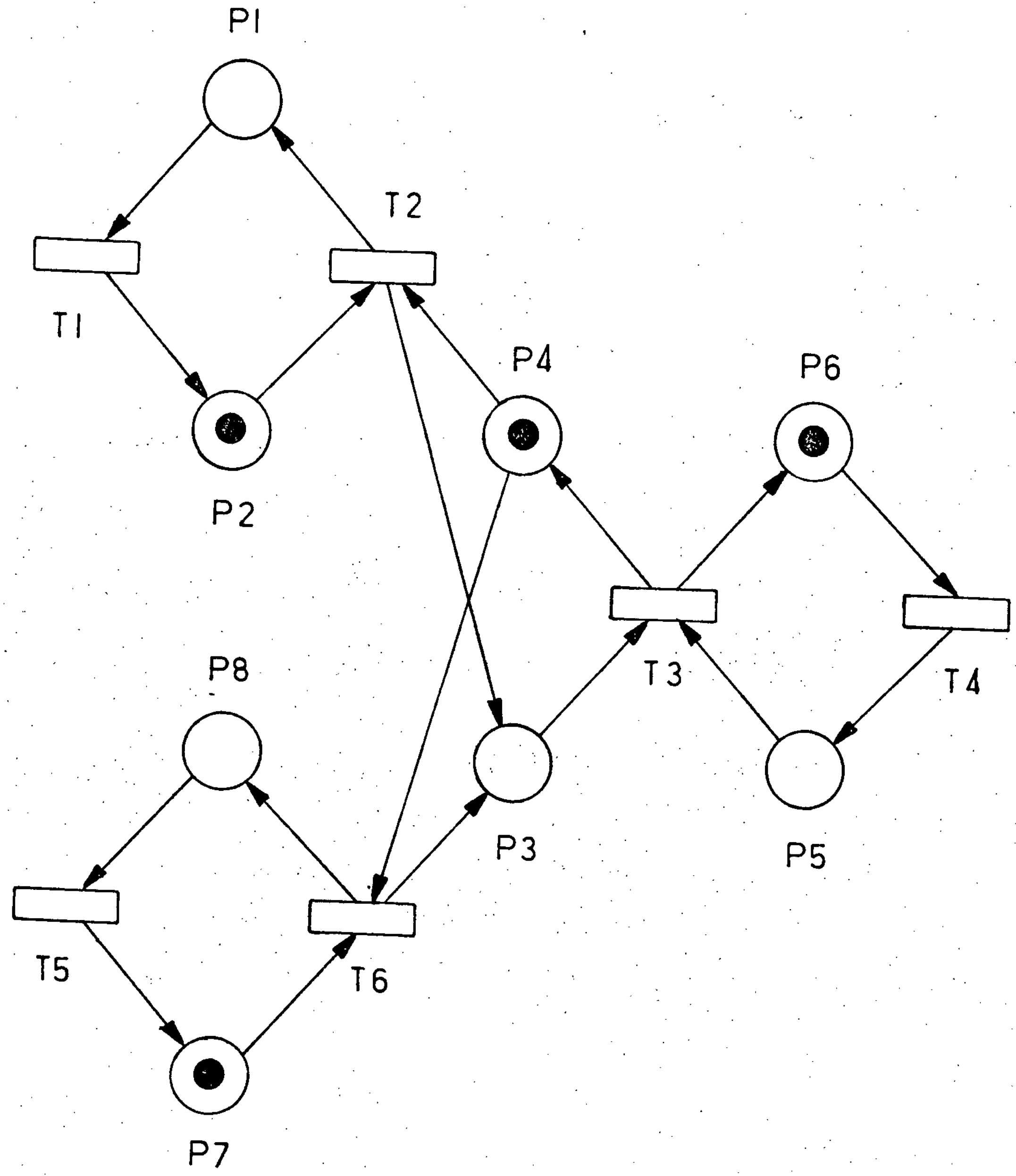


Figure 2.5: 2 consumer - single producer system

nets are liveness, persistence and boundedness.

The liveness property is important for the determination of the properties of a system. A transition is live if there exists a marking reachable from the initial marking in which the transition is enabled. A marking, m_2, is reachable from a marking, m_1, if there exists a sequence of transitions, σ, which takes m_1 through a sequence of states resulting in m_2. A stronger statement of liveness is that a transition is live if there exists a marking reachable from all other markings of the net in which the transition is enabled. A net is live if all of its transitions are live. If the Petri net model of a system is live, then one can infer (assuming a correct model), that the system has no actions which never occur and that the system cannot reach a state in which no actions may occur.

The liveness property of Petri nets has been the subject of much research [3, 5, 6, 11, 43, 52, 53, 55, 68, 69, 76, 88, 94]. The use of the liveness property for determining properties of parallel programs [36, 38, 44, 48] and for determining properties of concurrent systems [18, 31, 37, 41, 50, 51, 70, 71, 72, 76, 83, 91, 98, 99] is also well studied.

The behavioral property closely related to conflict is persistence. A marked net is persistent if the firing of a transition does not disable any other transition. In Figure 2.5, the firing of one of T2 or T6, under the marking shown, removes a token from the place, P4, thus disabling the other transition. Hence, the net is not persistent. If no conflicts exist in the net, then, the net is persistent. However, a net may have a conflict and exhibit persistence. In Figure 2.6, the transitions T1 and T2 are in conflict since they share a common input place. The firing of T1 under the marking shown does not disable T2. The firing of T2 under its enabling marking does not disable T1. The persistence property of Petri nets is useful in the modeling and design of hardware [8, 66]. A persistent Petri net model of a hardware circuit ensures that no race hazards exist in the circuit.

The final property of Petri nets to be discussed is boundedness. For some interpretations of a net, one may consider the tokens to represent resources and the places to represent storage for the resources. For most real systems of interest, the amount of storage for resources is finite. A place is k—bounded if the token count of that place never exceeds k. A net is bounded if all of its places are bounded. A net is safe if all of its places are bounded by a value of one. It can easily be seen that the examples discussed are not only bounded but also safe.

This completes the introduction to Petri nets and the properties relevant to the model introduced in Chapter 3. Petri nets are a rich area of research covering a wide range of applications and theoretical topics. A selection of papers is listed in the Bibliography and also in the two survey papers [2, 78].

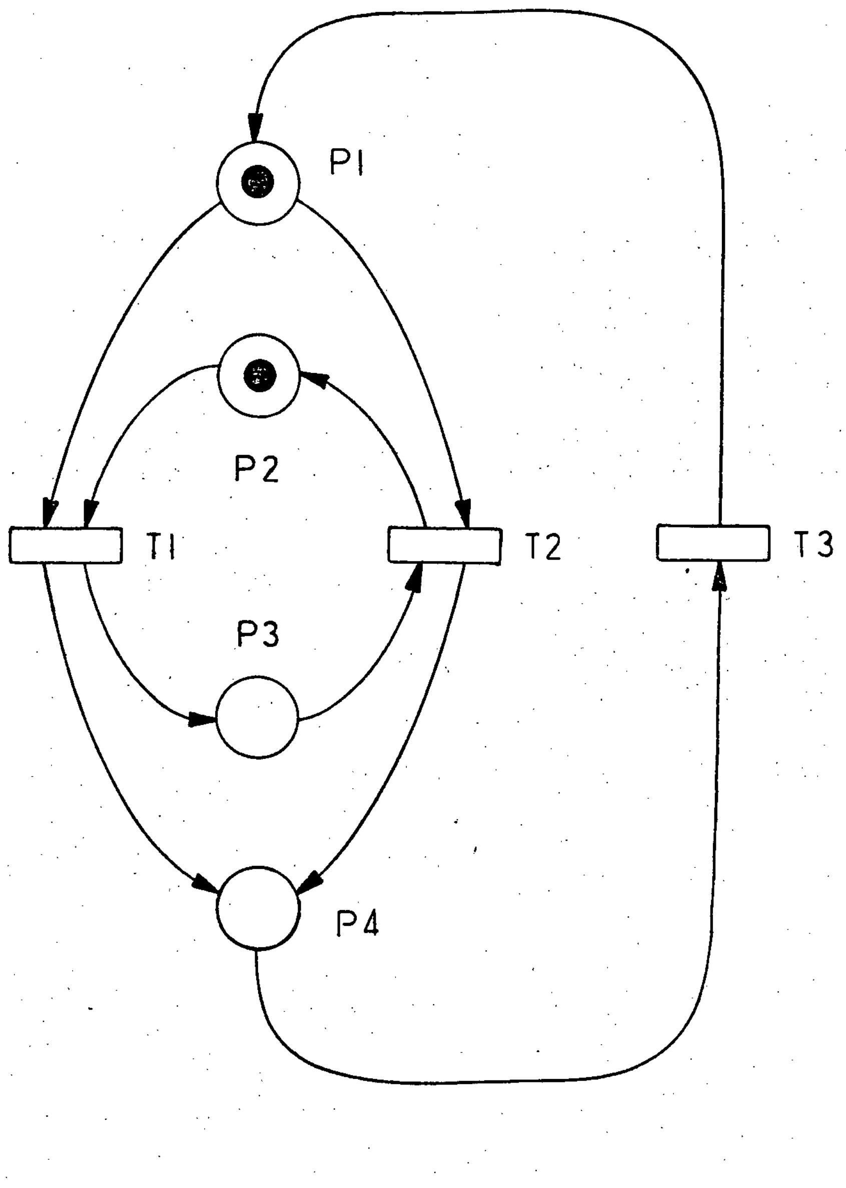


Figure 2.6: A persistent Petri net

2.2 Analysis of Petri Nets

The analysis of Petri nets is separated into two phases: structural and behavioral analysis [74]. The separation of the analysis into two phases is based upon the two stages of modeling, synthesis of the model and operation of the model. The structural analysis determines the static properties of the net and the behavioral analysis determines the dynamic properties of the net. The structural analysis is used to restrict the behavioral analysis to Petri nets which are live and bounded.

2.2.1 Structural Analysis

Structural analysis is based upon the notion that certain structures in the net affect the behavior of the net. The characteristic matrix is an $n \times m$ matrix representing the structure of the net. The elements of Γ, Γ_{ij}, are the net effect on the token count of place i by the firing of transition j. The structures are identified by the solutions to

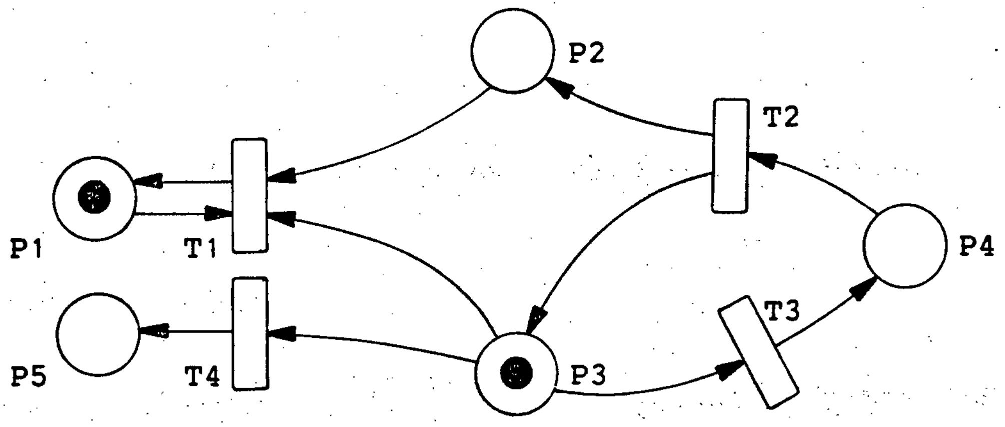
$$\Gamma \bullet g \leq 0,$$
$$\Gamma \bullet g \geq 0,$$
$$f \bullet \Gamma \leq 0,$$
$$\text{and} \quad f \bullet \Gamma \geq 0,$$

where f and g are vectors and "•" is a matix multiplication.

The solutions to $\Gamma \bullet g \geq 0$ and to $\Gamma \bullet g \leq 0$ identify sets of transitions which create or destroy tokens. Consider the example shown in Figure 2.7. g_1 is the transpose of a positive integer solution to $\Gamma \bullet g \geq 0$[1]. Each element of g_1 is greater than or equal to zero. T2 and T3 are identified by the non-zero elements of g_1. The structures identified by the positive integer solutions to $\Gamma \bullet g \geq 0$ are called generators. As T2 and T3 are repeatedly fired, an arbitrary number of tokens can accumulate in place P4. g_2 is the transpose of a non-trivial solution to $\Gamma \bullet g \leq 0$. T1 and T2 form an absorber. If T1 and T2 are repeatedly fired, an arbitrary number of tokens can be removed from the place P4. If a generator exists, the net cannot be bounded. If an absorber exists in a net which is bounded, then, the net is not live [88].

The solutions to $f \bullet \Gamma \leq 0$ and to $f \bullet \Gamma \geq 0$ identify sets of places which affect the movement of tokens through the net. f_1 is a positive integer solution to $f \bullet \Gamma \leq 0$. The places P3 and P4 which are identified by the non-zero elements of f_1 are called a deadlock. If transition T3 or T2 is fired a token remains in P4 or P3, respectively. However, if T1 or T4 is fired, then, P3 loses a token which it cannot regain by the firing of any other transition. f_2 is a solution to $f \bullet \Gamma \geq 0$ which identifies P5 as a trap. Once a token is placed in P5 it cannot be return to the rest of the net. In general a trap consists of several places; the tokens in the trap remain in the trap unable to

[1]The relations $\leq$ and $\geq$ are component-wise comparisons.



(a) A Petri net

(b) Characteristic matrix, Γ

$g_1=(0, 1, 1, 0)$

$g_2=(1, 1, 0, 0)$

$f_1=(0, 0, 1, 1, 0)$

$f_2=(0, 0, 0, 0, 1)$

(c) Solutions to Γ•g, f•Γ

Figure 2.7: Structural analysis

circulate in the rest of the net. If a deadlock exists in a net, then, the net is not live. If a trap exists in a net, then, the net is live if it is not bounded [88].

The dot product, $\Gamma \bullet g$, is the overall result on the token count of a place by the firing of a sequence of transitions, σ. In the firing sequence, σ, each transition is fired k_i times, where

$$g=(k_1,k_2,\ldots,k_i,\ldots,k_m).$$

If $\Gamma \bullet g=0$, then, the net effect of a firing sequence, σ, on a marking x is 0. If the net effect is 0 , then, a marking returns to itself, that is, $x=\sigma\Rightarrow x$. The firing sequence, σ, may be executed infinitely many times on the state x. If the firing sequence contains each transition, then, the state x is said to be strongly non-terminating. A Petri net is strongly non-terminating if there is such a state in the net [53]. If the solution to $f \bullet \Gamma=0$ and each of the elements of f is a positive integer, then, the net is both a deadlock and a trap. A net which is a single trap and a single deadlock is bounded [88].

A special case exists when the net is both strongly non-terminating and bounded. The net is said to be well-behaved. The necessary and sufficient conditions for a net to be well-behaved are given by

Theorem 1 [53]
A GPN is well-behaved if and only if $\Gamma \bullet g=0$ and $f \bullet \Gamma=0$ have positive integer solutions for the variables f and g. Γ is the characteristic matrix.

A well-behaved net precludes the modeling of systems with transient behavior. The well-behavedness is used to restrict the analysis to nets which are live and bounded.

2.2.2 Behavioral Analysis

The behavioral analysis of a Petri net is a systematic search of the possible states reachable from the initial marking. The systematic search is performed by the generation of a tree representing all of the markings reachable from the initial marking. The tree is called the reachability tree. The behavioral properties of liveness, boundedness and persistence are decided by examination of the reachability tree.

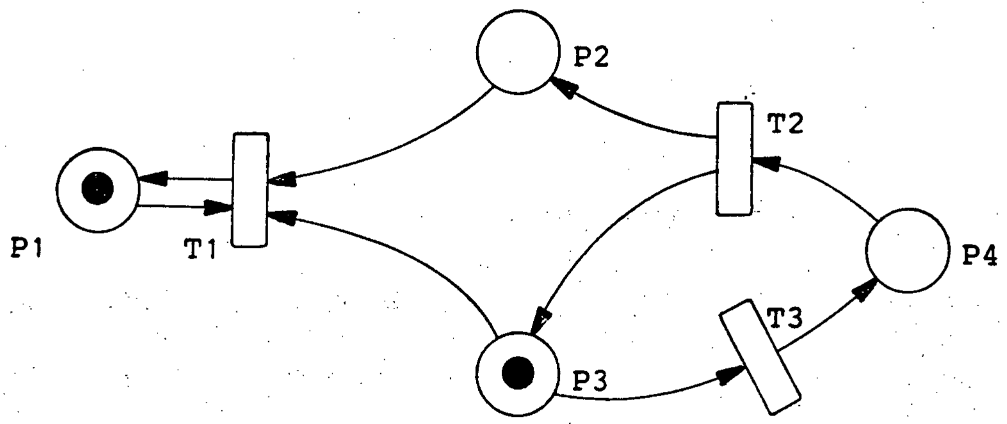
The reachability tree is a directed graph. The nodes are labeled with a marking and the edges are labeled with a transition. The root node is labeled with the initial marking. The generation of the reachability tree begins with the root node. For each unmarked leaf of the tree, generate all of the markings which are directly reachable from the marking labeling the leaf. For each new marking, create a new node and label it with that marking. Draw an edge from the leaf to each new node and label the edge with the transition which was fired to create the marking labeling the new node. If the marking is a dead marking, that is, there are no transitions enabled, then mark it as a dead marking. If the marking already labels another node in the tree, then, mark the node as an existing node. Repeat the above process until all of the leaves have been marked and no new nodes can be generated.

It can easily be seen that a bounded net generates a finite tree, since a finite number of states are possible. However, if a place is not bounded there exist infinitely many markings for the net. A symbol, ω, is introduced to indicate places with arbitrarily many tokens. The behavior of ω is defined to be

$$\omega+a=\omega$$
$$\omega-a=\omega$$
$$a<\omega,$$

where a is a natural number. ω may be though of as an infinity coefficient to a place in the marking, x (see Definitions 1 and 3, Appendix A). For a marking, x, in the reachability tree, if $x \geq y$, where y is any marking on a path from the root node and $\geq$ is a component-wise comparison, $x_i \geq y_i$, $1 \leq i \leq n$, then, ω replaces the coefficients for which $x_i > y_i$ is true.

Figure 2.8 shows a Petri net and its reachability tree. The initial marking of the net is P1+P3. T3 is the only transition enabled in the initial marking. The marking P1+P4 is created by the firing of T3. Transition T2 fires on the marking P1+P4 yielding the marking P1+P2+P3. Note that $P1+P2+P3 \geq P1+P3$ and $P2>0P2$, hence, the coefficient of P2 becomes ω. Transitions T1 and T3 are enabled under $P1+\omega P2+P3$. The firing of T1 creates the marking $P1+\omega P2$, since $\omega-1=\omega$. Since there are no transitions which are enabled under $P1+\omega P2$, the node is marked as a dead marking. The firing of T3 in the



(a) a Petri net

$$\begin{tikzcd}
 & (P1+P3) \arrow[d, "T3"] & \\
 & (P1+P4) \arrow[d, "T2"] & \\
 & (P1+\omega P2+P3) \arrow[dl, "T1"'] \arrow[dr, "T3"] & \\
(P1+\omega P2)^{**} & & (P1+\omega P2+P4) \arrow[d, "T2"] \\
 & & (P1+\omega P2+P3)^{*}
\end{tikzcd}$$

where * indicates an existing state and ** indicates a dead state

(b) Reachability Tree

Figure 2.8: Behavioral analysis

marking P1+ωP2+P3 creates the marking P1+ωP2+P4. Transition T2 is fired to create the marking P1+ωP2+P3. Since the marking labels a node, the leaf is marked as an existing node. The generation of the tree is complete since all of the leaves are either dead markings or existing markings and no new markings can be generated.

The properties of the net are decided by inspection of the reachability tree. The tree of Figure 2.8(b) has a leaf marked as a dead state. Thus, the net is not live. Since the tree contains nodes which are labeled with markings which use ω for a coefficient, the net is not bounded. The net is not persistence since the net is not live.

Figure 2.9 shows the reachability tree for the Petri net of Figure 2.6. The tree contains no leaves which are marked as dead markings, hence, the net is live. The coefficients for each place in each of the markings is 1, therefore the net is safe. The net is also persistent, since, each node has at most a single sucessor.

2.3 Classes of Petri Nets

The Petri net model presented in Chapter 3.3 is based upon three classes of Petri nets: marked graphs, timed Petri nets and coloured Petri nets. Marked graphs are a class of Petri nets with a restricted structure. The restriction permits general conditions for the liveness and the safeness

$$\begin{tikzcd}
(P1+P2) \arrow[d, "T1"] \\
(P3+P4) \arrow[d, "T3"] \\
(P3+P1) \arrow[d, "T2"] \\
(P2+P4) \arrow[d, "T3"] \\
(P1+P2)^{*}
\end{tikzcd}$$

where * indicates an existing state

Figure 2.9: Reachability tree for net of Figure 2.6

of nets belonging to the class. The timed nets model the notion of time in the net. Coloured nets allow the modeling of resource attributes and dynamic hierarchies. The timed Petri nets and coloured Petri nets are extensions to the Petri nets described in Chapter 2.1. The extensions increase ability of Petri nets to model systems in a natural and compact manner.

2.3.1 Marked Graphs

The class of Petri nets known as marked graphs[1], is created by restricting the structure of the net. Each place of the net is restricted to be an input to a single transition and to be output to a single transition. The example shown in Figure 2.1 is a marked graph. Since no conflicts are allowed, only deterministic systems can be modeled with a marked graph. However, the restrictions do allow a simplification of the analysis of the nets. Marked graphs are a particularily well-studied class of Petri nets [17, 44, 39, 52, 53, 67, 69, 70, 92, 93, 100] and many results are known for this class of Petri nets.

Due to the restricted nature of the structure of the marked graph, a directed graph, G=<V,E>, where V is a set of vertices and E is a set of edges, may be used as the graphical representation of a marked graph. The sets V and E are the transitions and places, respectively, of the marked graph. The

[1]A marked graph is not to be confused with a marked Petri net.

tokens in this representation are rectangles placed on the edges of the graph. The producer-consumer model of Figure 2.2 is shown in its directed graph representation in Figure 2.10.

As a marked graph is executed, the tokens move from edge to edge as the transitions fire. The tokens follow the directed paths in the graph, G. The liveness and safeness properties can be specified in terms of the directed paths. The following theorems define these properties.

Theorem 2 [17]

> A marking is live if and only if the token count of every directed circuit is positive.

Theorem 3 [17]

> A live marking is safe if and only if every edge is in a directed circuit with a token count of 1.

In the producer-consumer example shown in Figure 2.10, the liveness and safeness are verified easily. There are three directed circuits, T2–P1–T1–P2–T2, T2–P3–T3–P4–T2 and T4–P5–T3–P6–T4. By inspection, each circuit has a token count of 1. Hence, the marked graph is live and safe.

2.3.2 Timed Petri Nets

In a real system, actions occur over a duration of time. During this time neither the input conditions nor the output conditions are valid. As described above, the transitions have been assumed to occur instantaneously. The notion of time is

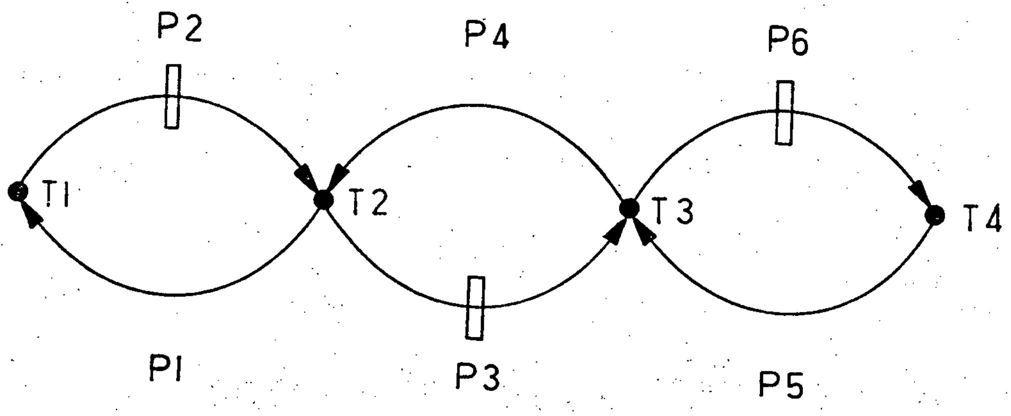


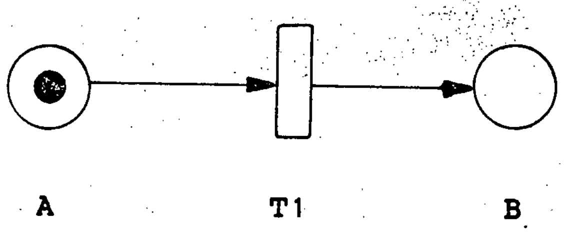
Figure 2.10: A simple marked graph

introduced in the form of a delay between the time the transition initiates firing by removing tokens from the input places and the time the transition completes firing by depositing tokens in the output places [63, 64, 82, 81, 92, 93, 102]. Figure 2.11 illutrates how a timed transition fires where transition T1 has a delay of τ_1, a positive real number.

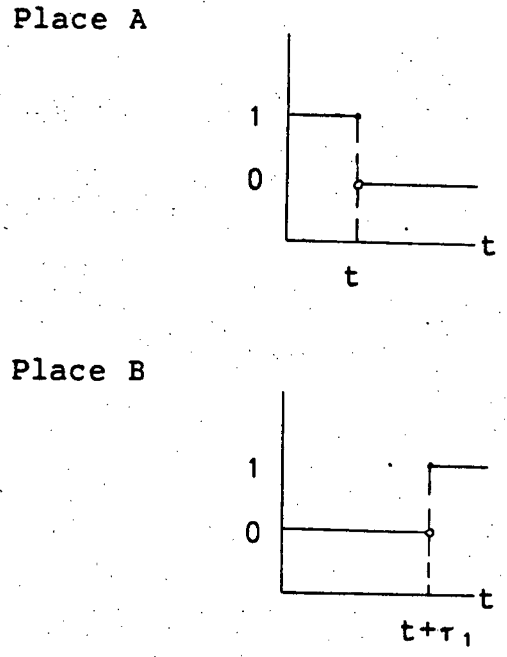
An alternative method of modeling the notion of time is a delay in the places of the net [10, 71, 73, 87]. Tokens are unable to be used to enable transitions for a delay time after it is deposited in a place. The modeling of the time delay in this manner requires a complex mechanism for the execution of a timed net.

Using the ideas of stepwise refinements [94], the two methods of modeling time can be shown to equivalent. Consider transition T1 of Figure 2.12(a) with time delay, τ_1. Transition T1 can be substituted by two transitions, T1' and T1", and a place PT1, as shown in Figure 2.12(b). In the net of Figure 2.12(b), T1' and T1" occur instantaneously and the token is held in place PT1 for a time τ_1. The time delay is now modeled as a delay in a place. Similarily, a place with a delay may be replaced by a transition with a delay and two places as shown in Figure 2.13.

A second concept of importance for timed Petri nets is synchrony. In real systems, several actions may begin close enough in time that they may be considered to begin



(a) Transition T1



(b) Token count of places A and B

Figure 2.11: Firing of transition T1

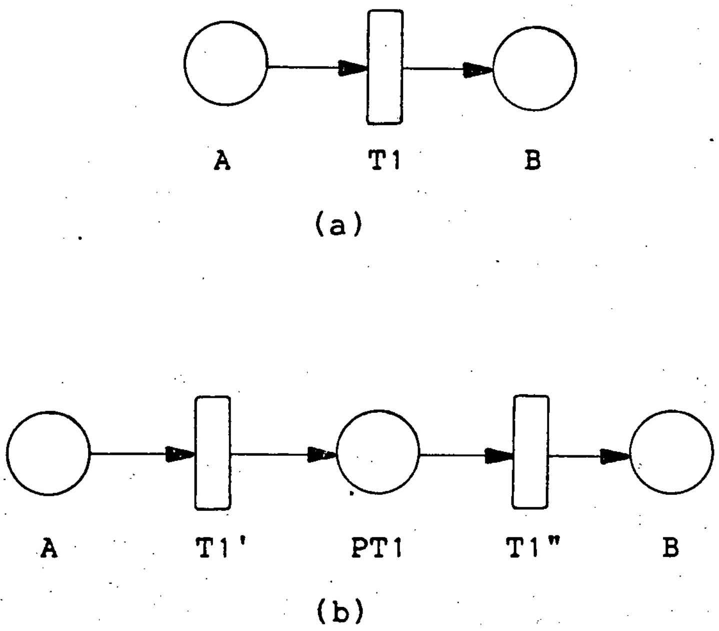


Figure 2.12: A timed transition

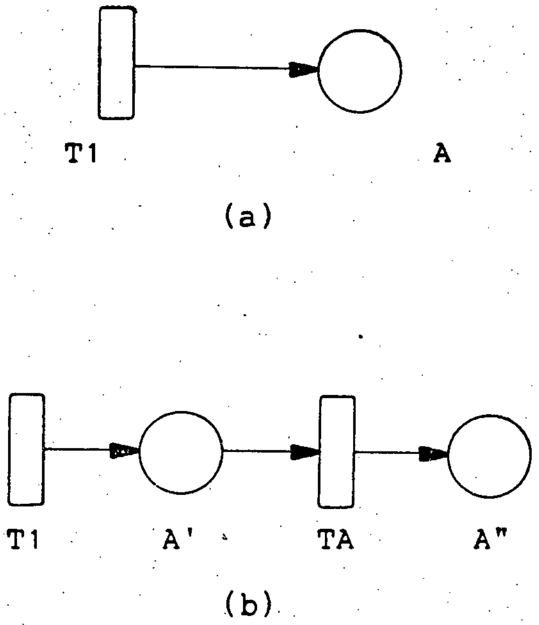


Figure 2.13: A timed transition

simultaneously. If each action is modeled as a separate transition, then in order to correctly simulate the behavior of the actions, the transitions must begin firing at the same time. The notion of synchronous firing of transitions also reduces the size of the state space of the net. Consider two states, one in which k transitions are enabled and the second in which the k transitions have been fired. If a single transition is fired at a time, there are k! different orders in which the transitions may be fired. For the firing sequences, there are $\sum_{i=1}^{k}\binom{k}{i}$ intermediate states in which some of the transitions have been fired. If the transitions are allowed to fire synchronously, then, all of the intermediate states are eliminated. Hence, a more compact representation of the state space is possible.

The introduction of the notion of time to Petri nets complicates the description of the state of a net. A marking, m, of a net is no longer sufficient to describe the state of a net. At any time, transitions may be firing and this information must be include in the state description. The time at which the descriptor is taken is an issue here. A convenient time to describe the net is the instant after a transition completes firing [102]. At this time, other transitions may become enabled and begin firing as a result of the transition completing its firing. The transitions which have not completed firing are described by the remaining time function, r.

$r=\{(T_i,R_i)\}\subseteq T\times R^+$

where R_i is a real positive number which is the time remaining in the firing of transition T_i. The state of a timed net is a pair, (m,r), when a transition has just completed firing.

Since the modeling of the notion of time is a behavioral extension, the analysis of timed Petri nets differs only in the behavioral analysis. A tree called the graph of instantaneous descriptors, GRID, represents the state space of the timed net. The graph of instantaneous descriptors is generated in the same manner as the reachability tree, described in Chapter 2.2.2. The nodes of the GRID are instantaneous descriptors, $d_i=(m_i,r_i)$, and the edges are labeled with a selector, where a selector, $s\epsilon T^*$. The selector, s_i, indicates which transitions were fired to create the state, d_i.

Since tokens become available to enable transitions at the completion of the firing of a transition, a state is not dead if transitions have yet to complete firing. A state is dead if there are no transitions which have not completed firing and there are no transitions enabled.

2.3.3 Coloured Petri Nets

In a Petri net model of a system, the tokens often represent resources in the system. The resources may have attributes which may not be easily represented in a Petri net. Many authors have addressed this problem by assigning the attributes to the tokens [21, 32, 38, 71, 73, 83, 98, 99, 101]. Coloured tokens are a convenient method of visualizing tokens with attributes [38, 83, 101]. Except in the case of an infinite number of colours, a Petri net with coloured tokens has an equivalent Petri net with uncoloured tokens. The construction of the equivalent net without coloured tokens involves a duplication of transitions and places for each colour [79]. The construction results in a large and complex net. The use of coloured tokens provides a natural and compact representation.

Beyond the simple use of attributes, the imposition of a partial order on the colours allows the modeling of priorty hierarchies. Consider the example shown in Figure 2.14 [38]. The producers, P_{i1} and P_{i2}, deposit units into the corresponding buffer, B_i. The consumers, C_1 and C_2 consume units from buffers, B_1 and B_2, respectively. The consumers interact with their corresponding buffer through a channel of capacity one. Consumer C_1 taking a unit produced by producer P_{11} has the highest priority on the channel over all others. Consumer C_2 taking a unit produced by producer P_{22} has the lowest priority on the channel. The priority is a dynamic

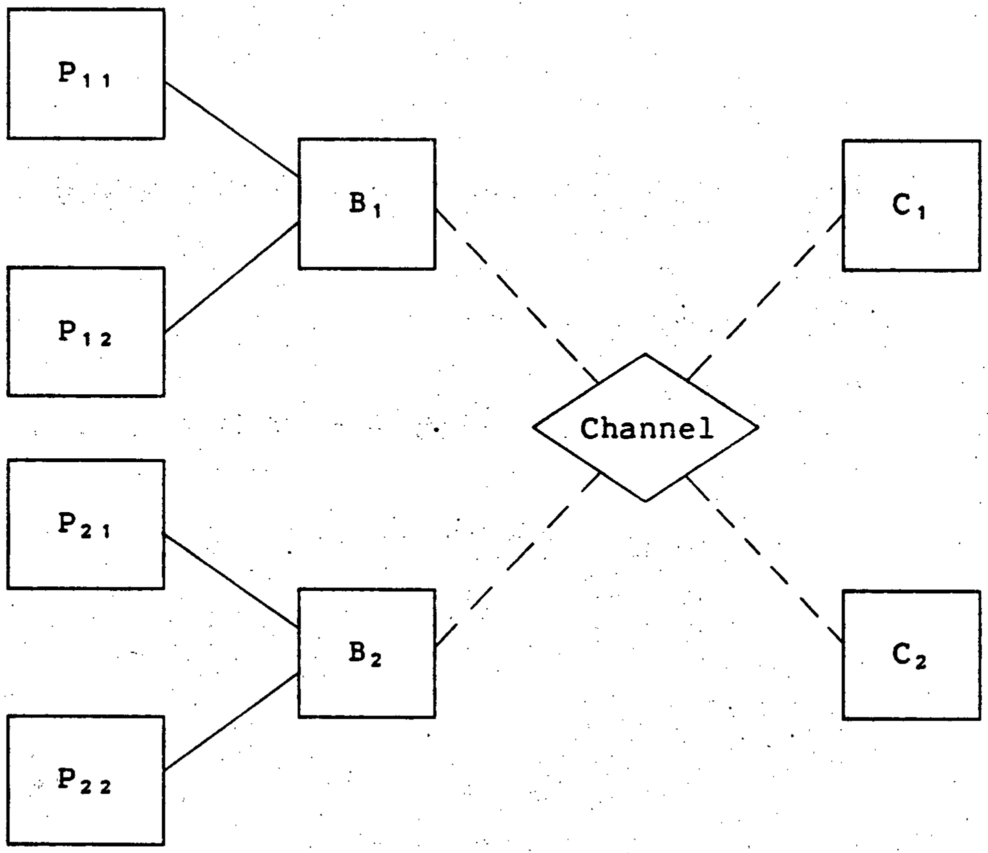


Figure 2.14: A producer-consumer system

priority since the priority of the consumers depends on the units present in the buffers.

The coloured Petri net model of the producer-consumer system of Figure 2.14 is shown in Figure 2.15. The partial order on the colours is shown at the bottom of Figure 2.15. The labels on the output edges of the transitions are the colour of the token deposited in the output place. The labels on the input edges of the transitions are the minimum colour which may be used to enable the transition. The tokens are compared under the partial order shown. Tokens not connected by a path are not comparable and cannot be used for an edge so labeled.

The priority of a transition is determined by the minimum colour of the tokens which are available to enable the transition at that point in the execution of the net. Suppose P7 contains a token of colour C_{12} and P8 contains a token of colour C_{22}. Both T5 and T6 are enabled but T5 has a higher priority since the minimum enabling colour of T5, C_{12}, is higher than the minimum enabling colour of T6, C_{22}. If P7 has a token of colour C_{12} and P8 has a token of colour C_{21}, either T5 or T6 may fire since C_{21} and C_{12} are incomparable under the partial order. One can easily see that if P7 has a token of colour C_{11}, then, T5 has priority over the firing of any other transition. The coloured Petri net correctly models the dynamic priority hierarchy specified. The coloured Petri nets may also be used to model reentrency in computer software [38,

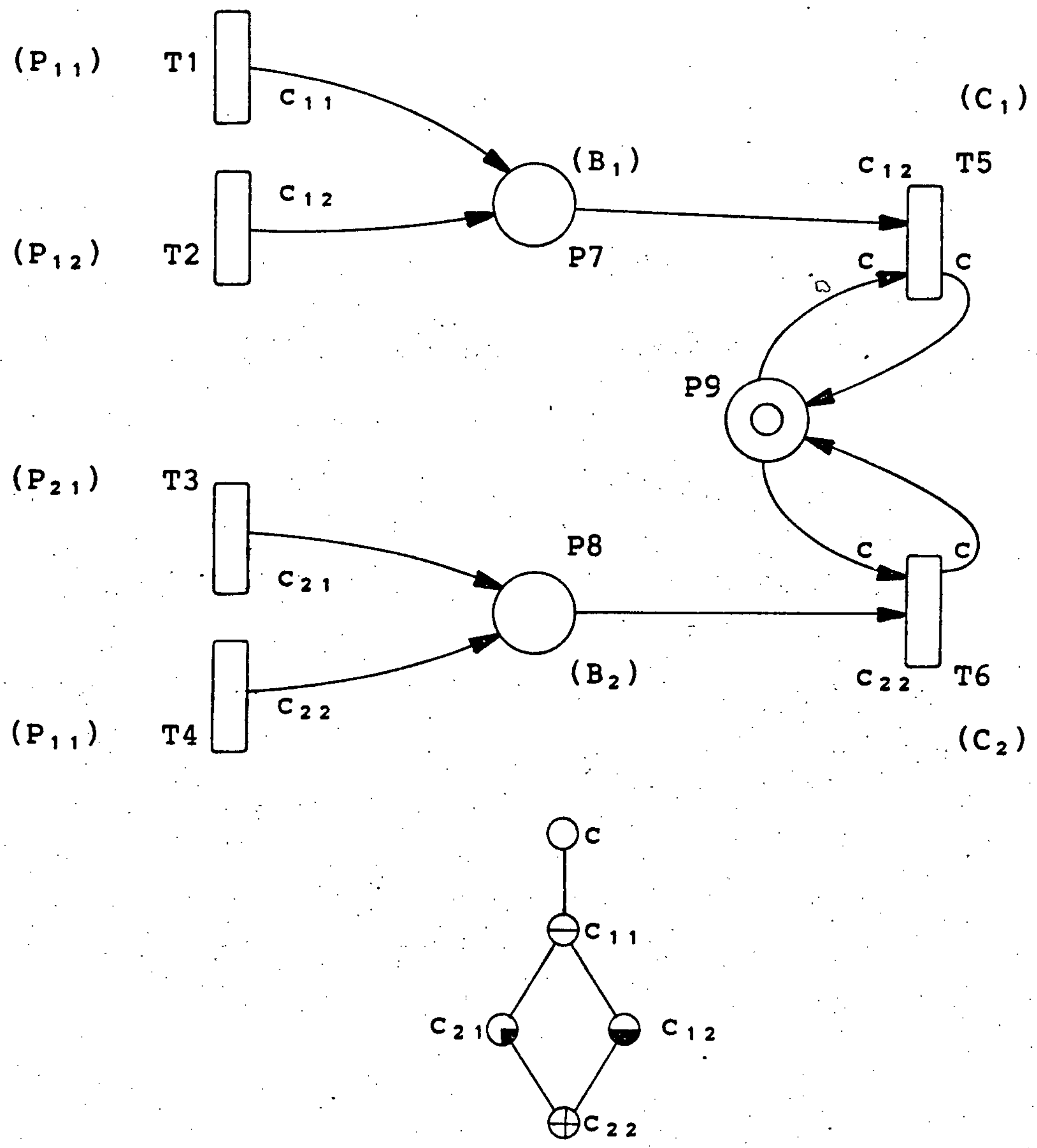


Figure 2.15: A coloured Petri net

101].

A further abstraction of the coloured Petri net model is the introduction of transition schemes [32, 98]. Transitions are similar if they are connected to the same places in the same manner except for the labels on the connecting edges. An example of similar transitions is shown in Figure 2.16. The two transitions may be thought to be two instances of the transition shown in Figure 2.17. A transition scheme may be thought of as a mapping of elements of P* onto other elements of P*. Thus, transition schemes allow a coloured net to retain the natural representation of a system.

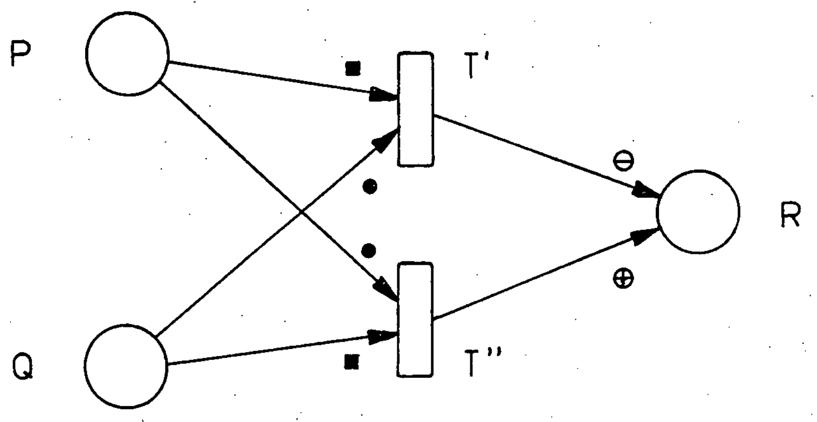
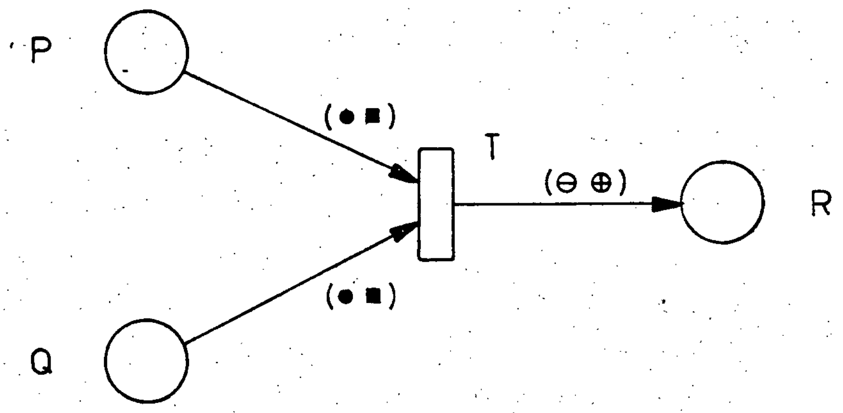


Figure 2.16: Similar transitions



P	Q	R
●	■	⊖
■	●	⊕

Figure 2.17: A transition scheme

CHAPTER 3
THE PETRI NET FORMULATION

A scheduling problem is comprised of a task system, the resources for its execution and a strategy for the scheduling of tasks. A Petri net is used here to formulate the structure of the task system and the structure of the resources. The tools of analysis of Petri nets are used to model the behavior of the scheduling strategy. The approach which is defined here differs significantly from previous Petri-net-based approaches to scheduling problems.

3.1 The General Scheduling Model

The general scheduling model presented here differs slightly from a more conventional treatment of the model [16, 26, 27, 28]. The formalization for the model is presented in Appendix A. The main difference is that the processors are to be considered as just another resource and not as a special resource. The reason for the treatment of the processors as another resource is demonstrated during the construction of the Petri net model. The scheduling problems are drawn from the general scheduling model.

The resources of the general scheduling model are any physical resource which a task may require in order for it to be executed. In general, the resources include at least one set of processors. Additional resources may represent primary or secondary storage, input/ouput devices or subroutine libraries. A set of resources may contain identical units, units of different functionality, units of different speed or a combination of units of varying functionality and/or speed. For example, several subroutine libraries may each contain different sets of subroutines. The subroutine libraries may be considered to be a single type of resource with units of varing functionality. It is convenient to consider the resources to be available in discrete units, where a unit is the smallest amount which can be aquired by a task.

The task system of the general scheduling model is defined on a set of tasks, $\mathcal{T} = \{T_1, T_2, \ldots, T_r\}$, where r is the cardinality of the set. Each task of the set, $\mathcal{T}$, is assumed to be executed once. The operational precedence constraints specify the data dependencies between the tasks. The precedence constraints are a partial order, $<$, on the tasks. If $T_i < T_j$, then, task T_i must complete execution before task T_j begins execution. The partial order is represented by a directed acyclic graph with no transitive edges. The directed acyclic graph is assumed to be given as a list of edges which is $O(r)$ in general. Note that if the graph is specified in terms of an $r \times r$ matrix, any operations on the matrix are $O(r^2)$.

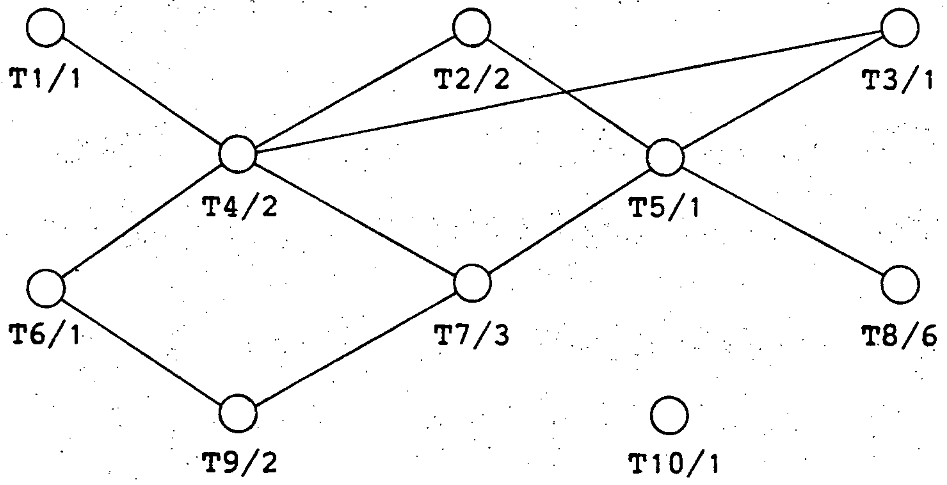
Each task in the set, $\mathcal{T}$, is executed in a finite amount of time. An $n \times r$ matrix, $\{\tau_{ij}\}$, is the matrix of execution times. $\tau_{ij} \geq 0$ is the execution time of task j on processor i.

The equality is introduced since two dummy tasks are used in the construction of the Petri net formulation. If a task T_i is unable to be executed on a processor j, then, τ_{ij} is infinite. Each task is assumed to execute on at least one processor.

A task system is shown in Figure 3.1. The task set consists of ten tasks, T1 through T10. Each node in the graph represents a task. The labels of the nodes indicate the task and the execution time of the task. The directed edges of the graph indicate that data is transferred between tasks and the direction of the transfer.

The tasks require resources to execute. The resource requirements are specified by $\mathcal{R} =[R_1(T_j), R_2(T_j),..., R_s(T_j)]$ for each task j, where s is the number of sets of resources. The component $R_i(T_j)$ specifies the the amount of resource i required for the execution of task T_j. It is assumed that the maximum requirement of any task may be satisfied by the initial resource configuration. The resource requirements are specified in the discrete units in which the resources are available. It is assumed that a task requires an amount of resource for the duration of its execution. The task neither requires more resources during the execution nor does the task release any resources until it completes execution. Tasks whose resource requirements are such may be modeled as separate tasks.

The class of scheduling strategies to be considered here



Task/execution time

Figure 3.1: Task system $(\mathcal{T},<,\{\tau_i\})$

is restricted to strategies which are nonpreemptive. Nonpreemptive scheduling strategies are characterized by tasks being executed without interruption, that is, once a task begins execution, it runs to completion without stopping. The performance criteria used here is to minimise the execution time, w, which is the time to execute all of the tasks of the task set $\mathcal{T}$. The optimal execution time is denoted by w*. The performance of a scheduling strategy on a given problem is expressed as a ratio, w/w*.

3.2 The Petri Net Formulation

The Petri net based approach to the scheduling problem is separated into two phases. First, the scheduling model is formulated as a timed-coloured Petri net. The Petri net models the structure of the scheduling problem, the data relationships between the tasks and the properties of the resources. The second phase models the behavior of the scheduling problem. The behavior of the scheduling problem is determined by the scheduling strategy and the initial set of resources. The scheduling strategy is modeled by modifications to the behavioral analysis of the Petri net. Hence, a schedule is generated by the execution of the net for an initial marking.

3.2.1 The Construction of the Petri Net Model

The Petri net model is a coloured Petri net extended with the use of transition firing times. The net structure is derived from the precedence contraints of the task system. The coloured tokens are used to model the resources, both the data and physical resources, required for the execution of the task system. Each task is represented by a transition scheme. Each instantiation of the transition scheme is assigned a firing time, the execution time of the task given the particular set of resources. The coloured-timed Petri net provides a natural representation for the task system and for the resources used for the execution of the task system.

Algorithm 1 is an algorithm for the construction of the Petri net model given the task system and the resources. The algorithm is shown in Figure 3.2. The initial portion of the construction may be thought of in terms of manipulations to the graphical representation of the precedence constraints.

The construction of the Petri net model begins with the directed acyclic graph representing the precedence constraints of the task system. The general precedence relation is first converted to a single-entry node, single exit node precedence graph. The addition of the STOP-START edge completes the augmented precedence graph. Recall, a marked graph may be defined in terms of a directed graph. Hence, the augmented precedence graph defines a marked graph. The nodes and edges

1. To the precedence graph, add two nodes labelled "START" and "STOP".

2. Add a directed edge from the START node to each node which has no incoming edges except the STOP node.

3. Add a directed edge from each node which has no outgoing edges to the STOP node.

4. Add a directed edge from the STOP node to the START node. Label the edge "READY" and mark it with a single uncoloured token.

5. Label each of the edges with a place name.

6. Label each of the nodes with a transition name.

7. Repeat Steps 8-11 for each resource.

8. Define a set of colours, $C_i=(X_i,<_i)$.

9. Add a place and label it with the name of the resource.

10. Add directed edges to and from each node(task) which requires that resource and label the edges with C_i.

11. Mark the place with the number of tokens representing the initial resource conditions.

12. Define a set of colours, $C_d=(X_d,<_d)$, for the data tokens.

13. Define a transition scheme for each node(task) in the graph.

14. Define a set of colours for the net, $C=\bigcup_i C_i \cup C_0 \cup C_d$, where C_0 is the uncoloured token.

Figure 3.2 Algorithm 1

are labeled for the formal representation. The directed acyclic graph of the precedence constraints of a task system is shown in Figure 3.1 [16]. The precedence is assumed to be from the top to the bottom. The marked graph resulting from Steps 1-6, is shown in Figure 3.3. The marked graph retains the natural representation of the precedence constraint of the task system.

The remainder of the construction deals with the resources necessary for the execution of the task system. It is convenient to consider this portion of the construction as operating on the formal representation of the net.

The resources are represented by a place in the Petri net marked with tokens from a colour set. A place is added to the net which is generated by Steps 1-6 of Algorithm 1. The place is connected to and from each transition(task) which requires that resources. The modeling of the resources in this manner is based on the assumption that a task uses the resource for the duration of its execution time and does not release the resource until the execution is completed. The edges are labeled with the colour set defined for the resource. The Petri net model allows the modeling of a variety of resources and processor situations.

The modeling of the processors as a resource allows a slightly more flexible modeling of functionally dedicated processors [57]. Groups of functionally dedicated processors

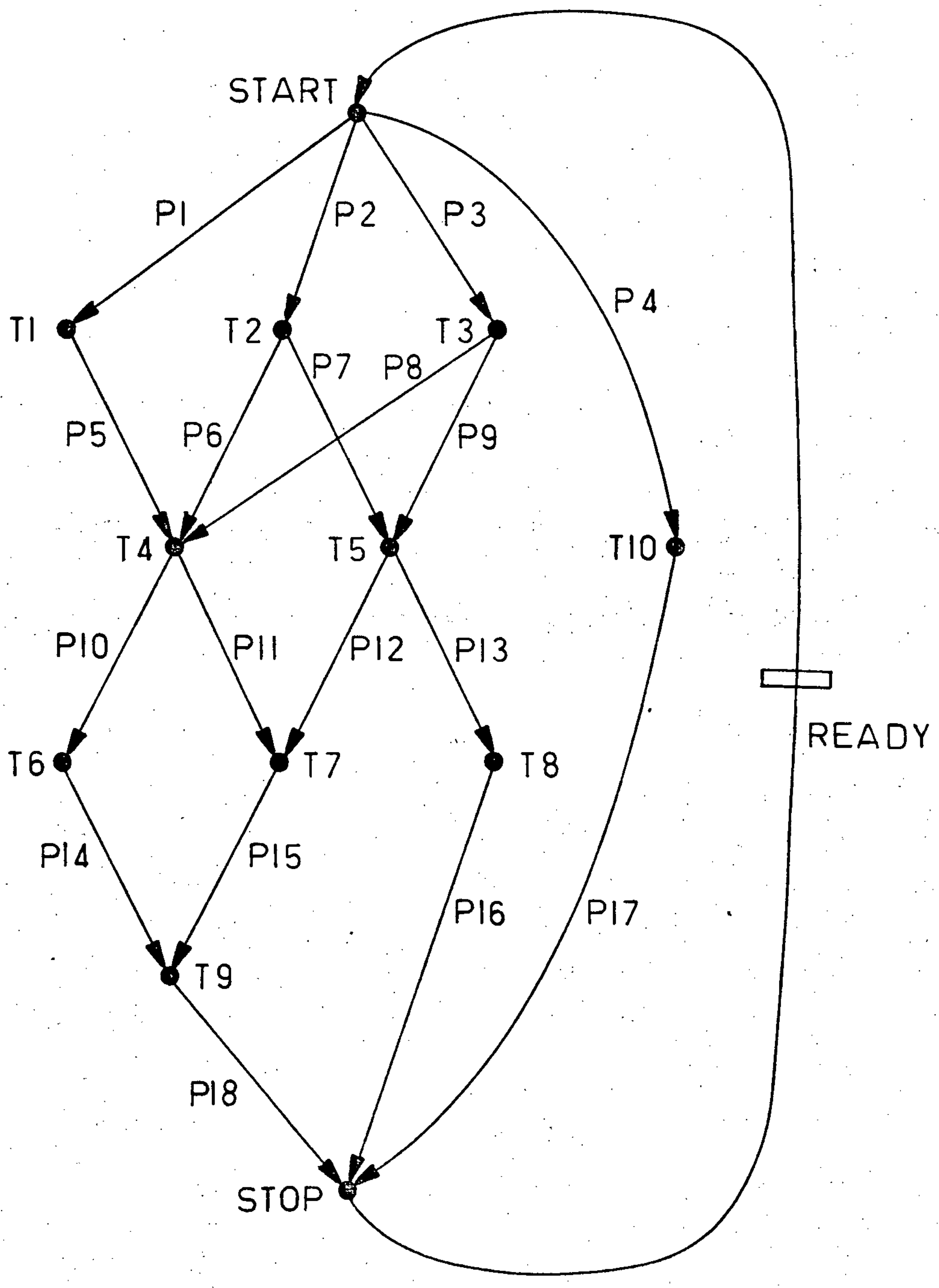


Figure 3.3: The marked graph after Step 6

may be modeled by different places in the net. A colour set and partial order is defined for each group allowing for the modeling of variation within the group of processors.

A colour set is defined for each resource place added in Step 9. A colour i defined for each attribute to be considered for the resource. A partial order is defined on the colour set reflecting the relationships between the attributes of the resources. The special case of identical resources, either resources or processors, is modeled by uncoloured tokens. If the resources are not interchangable, a null partial order is specified.

A transition scheme is defined for each transition(task) of the net after all of the resource places have been added. The transition scheme is the final step in the specification of the behavior of the transition(task). The instances of a transition scheme map a set of input tokens on to a set of output tokens. This corresponds to different input conditions for the execution of the task resulting in different output conditions. The transition schemes are defined in such a manner that none of the colours defined in Step 8 are redundant.

A simple transition scheme is shown in Table 3.1. The task is task T4 from the example shown in Figure 3.1. The task system is assumed to require no additional resources for the execution except the processors. The processors are of speeds

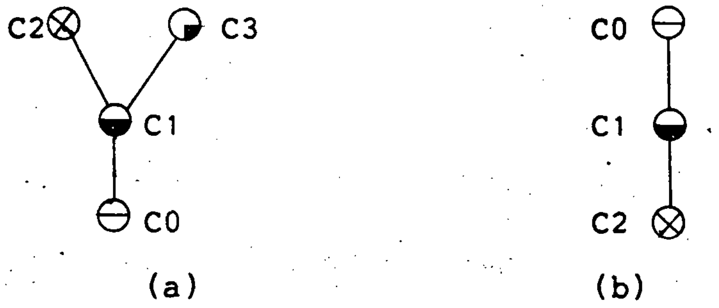


Figure 3.4: Colour sets

P5	P6	P8	Processor	P10	P11	T_4
●	●	●	C0	●	●	2
●	●	●	C1	●	●	3
●	●	●	C2	●	●	4

Table 3.1: Transition scheme, T4

$b_1:b_2:b_3=1:2/3:1/2$. The data tokens are all uncoloured. The top to bottom order may be used to define a priority hierarchy between the instantiations of the transition scheme.

Three examples illustrate the use of colour sets, partial orders and the transition scheme in modeling resources. First consider the example of a set of processors of different speeds. A colour set and partial order are shown in Figure 3.4(b). The colours are depicted by the different shading of the tokens. C0 and C2 represent the fastest and the slowest processors, respectively. It is assumed that the faster processor is to be used before a slower processor, so the partial order is directed downwards.

Figure 3.4(a) shows a slightly more complex colour set and partial order. The colour set models a group of processors of different speeds and some of which are functionally dedicated. C2 and C3 represent processors with special hardware, for example, a floating-point processor and an array processor. C1 and C0 are general purpose processors of different speeds. C1 is assumed to be faster than C0. The general purpose processors can perform the functions of C2 and C3 in software, and hence at a slower speed. The partial order reflects the differences between the processors both in the functionality and the speed.

The final example is a dynamically reconfigurable architecture [45, 46, 97]. The architecture to be modeled,

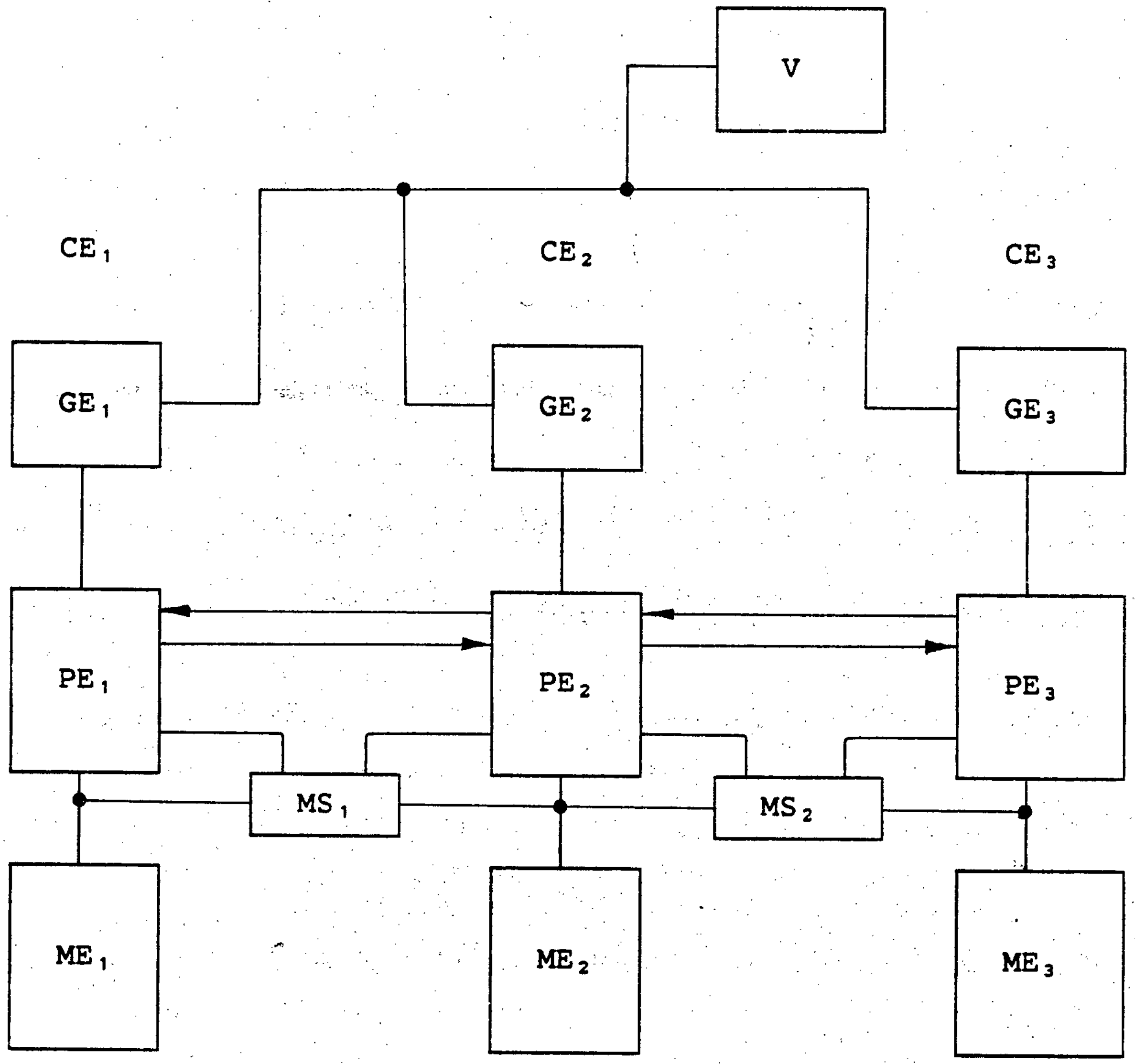


Figure 3.5: A dynamically reconfigurable architecture

shown in Figure 3.5, consists of three 16-bit computing elements, CE_i. Each CE_i consists of a processing element, PE_i, a memory element, ME_i, and an input/output element, GE_i. The computing elements are connected by two connecting elements, MS_i. The connecting elements control the size of the processors created by the computing elements. The reconfiguration is controlled by the control element, V.

The architecture can be reconfigured into 16-bit, 32-bit or 48-bit processors. If a connecting element allows a connection, then, the two computing elements act as a single 32-bit processor. If the connecting element does not allow a connection, then, the two computing elements are considered to be two separate processors. The connecting elements allow the architecture to be reconfigured into combinations of the 16-bit computing elements. The only restriction is the two outermost computing elements cannot be connected to form a 32-bit processor.

Consider the precedence constraints shown in Figure 3.1. The processor requirements and new task execution times appear in Table 3.2. The task execution times are random integers between 1 and 5. The marked graph portion of the Petri net model remains the unchanged as shown in Figure 3.3. A single processors place is added to the marked graph. Three coloured tokens, C1, C2 and C3, are used to represent the three computing elements, CE_1, CE_2 and CE_3, respectively. The partial order on the colours is null since the relationships

Task	No. of bits	τ_i
T1	16	3
T2	32	3
T3	16	5
T4	48	3
T5	32	4
T6	32	5
T7	32	3
T8	16	1
T9	32	2
T10	16	1

Table 3.2: Task processor requirements and execution times

between the computing elements is more clearly modeled in the transition schemes.

For the specification of the transition schemes, consider task T3. The execution of task T3 requires a 16-bit processor. This requirement is satisfied by any of the three computing elements acting as a 16-bit processor. If the middle computing element, CE_2, is assignd task T3, the remaining computing elements could only be configured as two 16-bit processors. A better configuration would be to configure an end computing element as a 16-bit processor, which allows the possibility of a 32-bit processor to be configured. The transition scheme for task T3 is shown in Table 3.3 in which the priority is assumed to be top to bottom. A task requiring a 32-bit processor, for example task T7, is executed on a processor comprised of the computing elements, CE_1-CE_2 or CE_2-CE_3. The computing elements must all be connected in order to execute task T4 which requires a 48-bit processor. The transition schemes for tasks T7 and T4 are shown in Tables 3.4 and 3.5, respectively.

In contrast to the transition scheme approach to the modeling of the dynamically reconfigurable architecture, consider modeling the three computing elements by three uncoloured tokens. The Petri net would be a generalised Petri net (see Appendix 1). The weights on the edges from the processor place indicate the number of computing elements to create the processor of correct word width. A partial net of tasks T3 and T7 modeled in this fashion is shown in Figure

Input Data	Processor	Output Data
P3	C1	P8+P9
P3	C3	P8+P9
P3	C2	P8+P9

Table 3.3: Transition scheme for task T3

Input Data	Processor	Output Data
P11+P12	C1+C2	P15
P11+P12	C2+C3	P15

Table 3.4: Transition scheme for task T7

Input Data	Processor	Output Data
P5+P6+P7	C1+C2+C3	P10+P11

Table 3.5: Transition scheme for task T4

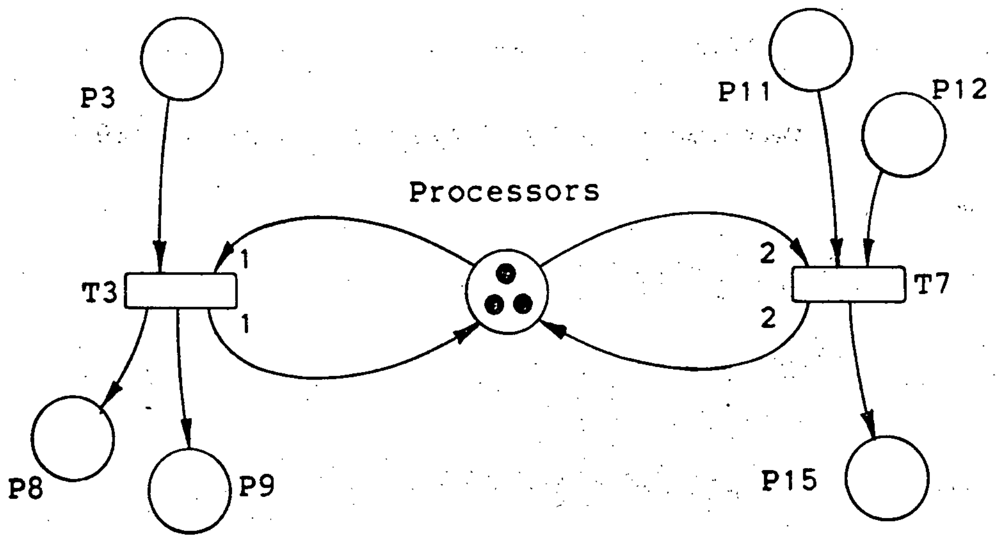


Figure 3.6: Tasks T3 and T7

3.6. However, this approach does not allow the modeling of the special interconnections between the processors, that is, the end computing elements, CE_1 and CE_3, cannot be used to form a 32-bit processor. The uncoloured tokens cannot model the priority given to assigning a task requiring a 16-bit processor to an end computing element, CE_1 or CE_3.

The final step in the construction of the Petri net model is the assertion that a valid Petri net has ben constructed. A valid Petri net model is a live and bounded net, since the problem is assumed to be executable on finite resources. The modeling of the scheduling strategy depends on the state space of the net, the graph of instantaneous descriptors, GRID. The GRID for a live and bounded timed net is finite. Theorem 4 asserts that the Petri net model constructed by Algorithm 1 is live and bounded.

<u>Theorem 4</u>

Algorithm 1 constructs a live and bounded Petri net from the acyclic directed graph reprentation of the precedence constraints of a task system.

PROOF: First it is shown that steps 1-6 yields a live and safe marked graph. Then it is to be demonstrated that the addition of the resource places does not affect the structural liveness property and the net remains bounded.

Steps 1-3 converts an arbitrary precedence relation into a single-entry node, single-exit node relation. If one considers the graphical representation of the precedence relation, an acyclic graph, then, there exists a directed path from the START node to any node in the graph. If a node had no predecessors, then, it would be connected to the START node by step 2. If a node had predecessors, then, one could trace back along a directed path to a node with no predecessors which would be connected to the START node by step 2. Similarily, each node is on a directed path to the STOP node. Therefore each node and each edge is on a path from the START node to the STOP node.

Step 4 adds an edge from the STOP node to the START node. The directed paths of the precedence graph become directed cicuits in which all the nodes and all of the edges lie on at least one such circuit. All of the directed circuits contain the STOP-START edge since the original graph was assumed to be acyclic. Now consider the graph to be a marked graph. When the STOP-START edge is marked with a READY token, every directed circuit has a token count of 1. Hence, the net is live and safe by Theorems 2 and 3.

The remainder of the proof is based upon the structural liveness of the net. Consider the initial state of the net with the READY place marked with single token. The firing of the START node enables the execution of the tasks. By definition of the problem, all tasks execute once, hence, all transitions must fire once. Firing the STOP node returns the net to the original state. Hence, the net is strongly repetitive and $\Gamma \bullet g=0$ has a positive integer solution , where Γ

is the characteristic matrix. Since the net is safe, $f \bullet \Gamma = 0$ has a positive integer solution.

Now consider the characteristic matrices before and after the addition of the resource places, Γ and Γ', respectively. If Γ is an $n \times m$ matrix, then, the first n rows of Γ' are identical to the first n rows of Γ. The rows in Γ' for the resource places are rows of zeroes. Since the edges are drawn to and from each task which requires the resource and the tasks do not consume the resources, then, $a_{ij} = b_{ij}$. a_{ij} is the number of tokens from place i which is required for the firing of transition j. b_{ij} is the number of tokens deposited in place i by the firing of transition j. Since the entries Γ_{ij} only reflect the net result on the token count of a place, therefore $\Gamma_{ij} = 0$ for the entries for the resource places.

Consider the solution to the equation $\Gamma' \bullet g' = 0$. Since $\Gamma \bullet g = 0$ and only rows were added to Γ to get Γ', then $\Gamma' \bullet g = 0$. Therefore g is a positive integer solution to Γ'.

Consider the solutions to the equation, $f' \bullet \Gamma' = 0$. Since $f \bullet \Gamma = 0$ and only zero rows are added to Γ to get Γ', then the first n elements of f' are the same as f. The remaining elements of f' are abitrary positive integers for the solution to be a positive integer solution.

$$f' = (f_1, f_2, \ldots, f_n, \sigma_1, \sigma_2, \ldots, \sigma_s)$$

where σ_i are arbitrary positive integers and s is the number of resource places added.

Since $f' \bullet r' = 0$

and $r' \bullet g' = 0$,

therefore the net with the resource places is well-behaved and hence, structurally live.

Since the initial state of the marked graph portion of the net is live, only the resource places must be marked in such a fashion that a live state is created. The resource places are self-loops, hence, each resource place must be marked by at least the minimum set of enabling tokens from the colour set. The minimum set of enabling tokens is the maximum for any single task. The presence of the minimum set of enabling tokens ensures that each transition scheme may fire. Since the net is well-behaved, and the initial state is live therefore the net is live and bounded. Q.E.D.

This completes the construction of the Petri net model for the scheduling model. The Petri net is live and bounded for a live initial marking. The initial marking for the net represents the initial resource conditions for the execution of the task system. A live initial marking is a token in the READY place and at least one token in each of the resource places. The execution of the Petri net simulates the execution of the task system.

3.2.2 Modeling the Behavior of a Scheduling Strategy

Consider a scheduling problem, a task system and the resources for its execution. The state space for the problem is all of the combinations of completed, partially completed and unexecuted tasks. The transitions between the states are the tasks which began execution to lead to the next state. One may consider the state space to be a directed graph where the nodes and edges are the states and the state transitions, respectively. The paths between the initial state in which no tasks have been executed and a final state in which all tasks have been executed represent possible schedules for the task system.

Since the schedules are paths in the state space, the optimal schedule is a path with the shortest execution time along its length. To find the optimal schedule all of the paths between the initial state and a final state in general must be searched. A scheduling heuristic may be applied to the problem to reduce the amount of the searching of the state space to achieve optimal or near optimal schedules. A scheduling heuristic may be thought of as selecting a path through the state space based on information from only a portion of the state space.

If the formulation of the scheduling model is a timed Petri net, then, the state space of the problem is the state space of the Petri net. A valid schedule may be thought of as a path through the state space of the Petri net. The state space of a timed Petri net is searched by the construction of the graph of instantaneous descriptors, GRID. A valid schedule may be found by finding a path from the initial descriptor, d_0, to a descriptor in which all transitions have been fired once.

An algorithm for the construction of a GRID of a live and bounded timed Petri net is shown in Figure 3.7. The Algorithm constructs the complete graph representing all of the states reachable from the initial descriptor. The complete graph is necessary in the original application of the analysis of timed nets [102], where the timed nets were used for preliminary performance evaluation. The construction of the complete graph in the context of the scheduling model would yield the optimal schedule. However, since it is known that both the analysis of Petri nets and the general scheduling problem are NP-complete [16, 27, 55], heuristic methods are an approach to reducing the amount of searching and yet achieve optimal or near optimal results. The algorithm may be modified to search a restricted amount of the state space of the net. The modifications to the algorithm are used to model the behavior of the heuristic approach.

The behavior of the scheduling heuristic is modeled by Step 4 of the algorithm. The problem of finding which transitions to fire on the current descriptor is exactly the problem of finding which tasks to execute on the current

1. Label the root node $d_0=(m_0, r_0)$, where m_0 is the initial marking and $r_0=\{\}$ is the initial remaining time.
2. $d_i:=d_0$, where d_i is the current node.
3. Repeat Steps 4-9 until there are no unmarked leafs.
4. Find the selectors, $\{s_j\}$, enabled on m_i of d_i
5. Repeat Steps 6-8 for each s_j.
6. Generate a new descriptor, d_{i+1}.
7. Create a new node and label it d_{i+1}.
   Join the node with an edge and label it s_j.
8. If d_{i+1} is an existing node, then mark it with a *.
9. Set the current node, d_i, to any unmarked leaf.

Figure 3.7: An algorithm for the generation of the GRID of a live and bounded net

state. This is easily seen since the tasks are modeled as transitions in the Petri net formulation of the scheduling problem. The number of selectors which is generated by Step 4 controls the amount of the graph searched, and the particular selectors control the paths searched. The modeling of the behavior of a scheduling heuristic may be thought of in terms of a static or a dynamic scheduling heuristic.

Consider the modeling of the behavior of a simple scheduling heuristic, list scheduling. The list scheduling heuristic schedules the first unexecuted task from a prioritzed list, L, as soon as a processor becomes free to execute a new task. The performance of the list scheduling heuristic is dependent upon the generation of the list, L. The Coffman and Graham Algorithm A for the generation of the list, L, is optimal in the two identical processor case [16]. The behavior of the list scheduling is modeled simply by implementing the search for selectors as a scan of the list, L.

The list scheduling heuristic involves no searching of the state space other than the states along the path selected. Another class of heuristics, lookahead heuristics, involves a limited search of the state space [4]. If the paths represent schedules, then, paths not ending in a final state are partial schedules. Several paths may be searched until they are found to be dominated by other partial schedules. Consider a subset of the task set, $\mathcal{T}$. Possible schedules are permutations of the subset of tasks. If a permutation has a smaller completion time, then, it dominates the other permutations [4, 80]. The idea of sequence dominance may be used to control the amount of the graph which is constructed.

3.3 Related Work

The extension of Petri nets to include the notion of time is directly founded on the desire for useful information from the Petri net model. The use of timed Petri nets for the performance evaluation of system is well documented [10, 21, 33, 70, 81, 82, 86, 102]. However much of the work is limited to system which may be modeled by well-behaved Petri nets. The use of timed Petri nets for the performance evaluation of general systems is beyond the scope of the present work. Within the scope of the present work, timed Petri nets have also been used in the optimization of sequencing decisions [84, 92, 93]. The notion of time constrains the net, precluding some of the sequences possible in the untimed case.

An example of the use of Petri nets for the optimization of sequencing decision is presented by Shapiro and Saint [84]. The problem addressed is that of generating efficient programs for machines with parallel operation capabilitites. The algorithms are expressed in a high level algorithmic language. The example detailed in the paper is a FORTRAN DO-loop executed on a CDC 6600.

The approach to the sequencing problem begins with the decompilation of the algorithm. The decompilation is the removal of the incidental sequencing constraints introduced by the high level language. The decompiled version of the algorithm is modeled as a Petri net. Next the hardware contraints are added to the Petri net model. The optimization is performed by a simulation of the algorithm, that is, executing the net. The net is executed for each set of initial conditions. The simulation is stopped when all sets of initial conditions have been tried or when a solution near the theoretical optimal is achieved.

The scope of the problem which is addressed by Shapiro and Saint is more restricted than the scope of the problem which is addressed by the present work. The Petri net model of Shapiro and Saint models the FORTRAN DO-loop and the hardware at the level of machine operations and machine functional units. Thus, the Shapiro and Saint model is at a lower level than the Petri net formulation of Section 3.2.1. The objective of the work of Shapiro and Saint is the optimization of the machine code. In contrast, part of the objective of the present work is the modeling of scheduling strategies, whether the strategy is a heuristic strategy or an optimal strategy. The FORTRAN DO-loops imply repetition of the machine operations. The tasks in the general scheduling problem, which is considered in the present work are assumed to be executed a single time. The problem addressed by Shapiro and Saint is a form of the general scheduling problem, and so, the present work is applicable.

A more recent approach to the problem of the optimization of the sequencing of events using a Petri net approach is presented by Tani et. al. [92, 93]. The basis for the Petri net model used in [92, 93] is a theoretical model of parallel computation, hence, the name capacitated computation graph. The problems addressed for capacitated computation graphs are the existence of admissable schedules, algorithms for the earliest and latest schedules, the termination property and the maximum computation rate for periodic schedules.

A computation graph, CG, is a marked graph in which self-loops are allowed. A self-loop is an edge which is input and output to a single node. A capacitated computation graph, CCG, is a computation graph with the edges limited in the total number of tokens which may be held on an edge. The firing rules must be altered since a node cannot fire if any of its output edges holds the maximum number of tokens. Each edge in a CCG, is assigned a time. The time represents the delay in placing the token on the edge after the initiation of the firing of a node. Each node is fired a maximum of X_i times, where X_i is a positive integer.

A capacitated computation graph is shown in Figure 3.8. The labels on the edges, (c_e, d_e^0, τ_e), indicate the capcity of the edge, c_e, the inital marking of the edge, d^0, and the time

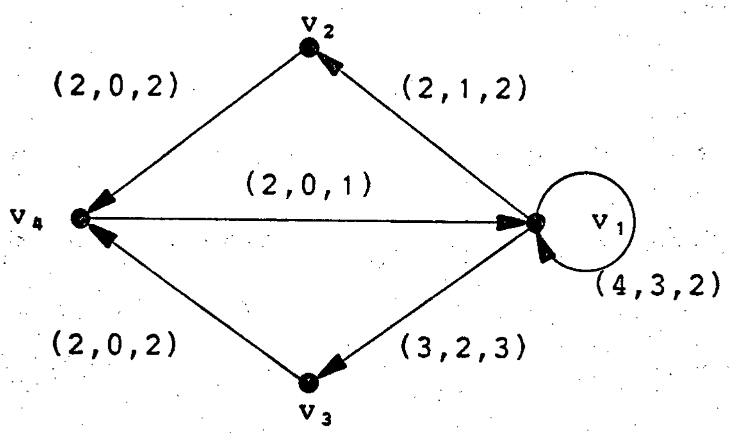


Figure 3.8: A capacitated computation graph

	k=1	k=2	k=3	k=4	k=5	...
$s_1(k)$	3	8	13	18	23	...
$s_2(k)$	0	5	10	15	20	...
$s_3(k)$	0	2	5	10	15	...
$s_4(k)$	2	7	12	17	22	...

Table 3.6: An admissable schedule for CCG of Figure 3.8

delay of the edge, τ_e. The schedule for the CCG is expressed in terms of the $s_i(k)$ which is the time of the k^{th} initiation of the firing of node i. Table 3.6 shows an admissable earliest schedule for the CCG of Figure 3.2. An algorithm for the finding of the $s_i(k)$ is shown in Figure 3.9.

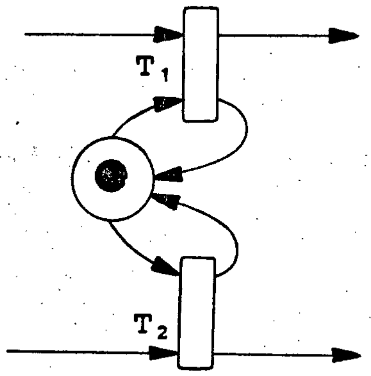
The capacitated computation graph model does not have the ability to model resource contention between operations [93]. A system with resource contention is first modeled as a generalized Petri net. The conflict is resolved arbitrarily. Each possible resolution of the conflict is modeled using a combination of the duplication of places and edges, the initial marking and the time delay in the places.

Figure 3.10 illustrates the resolution of a conflict by the duplication of the places and two initial markings. Each resolution of the conflict is converted to a CCG model for which a schedule is found. The best schedule, the schedule with the smallest completion time, specifies the conflict resolution in the structure of the CCG.

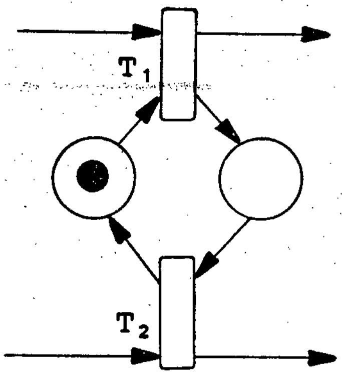
Part of the Petri net model which is presented in Section 3.2 is a special case of the capacitated computation graph model. Consider the marked graph representation constructed by Step 1-6 of Algorithm 1. Each task executes a single time by definition of the problem, hence, $X_i=1$, $1 \le i \le r$, where r is the number of tasks. The marked graph is safe, hence, the capacity on each edge may be arbitrarily assigned any non-zero value.

```
1.  Initialize $s_j(1) := 0 \quad 1 \le j \le m$

2.  Repeat Step 3 until no more changes are possible
    for all nodes.

3.  $s_j(1) := \max \left( \begin{array}{l} s_j(1) \\ s_i(1)+\tau_{ij} \text{ for all input edges which are unmarked} \\ s_i(1)-\tau_{ji} \text{ for all output edges which are marked to capacity} \end{array} \right)$

4.  Repeat Step 5 for $2 \le k \le \max_j X_j$.

5.  $s\ (k) := \max \left( \begin{array}{l} s_j(k-1)+\tau_{jj} \\ s_i(k-d_o)+\tau_{ij} \text{ for all input edges } ij \\ s_i(k-c_{ji}+d^o_{ji})-\tau_{ji} \text{ for all output edges} \end{array} \right)$
```

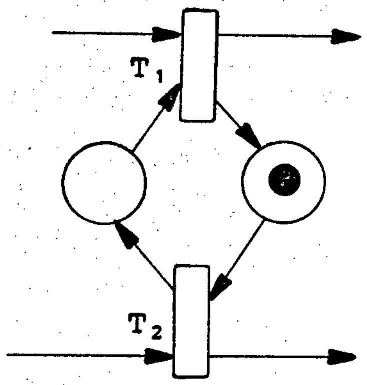
Figure 3.9: An algorithm to construct the earliest schedule of a CCG



(a) A resource conflict



(b) T_1 precedes T_2



(c) T_2 precedes T_1

Figure 3.10: Resource conflict resolution

This is the limit of the similarity between the two approaches. The methodology described above for the resolution of conflicts becomes unmanagable and awkward for even the simple case of a task system executing on a set of identical processors. The number of CCG's possible when a number of resource conflicts exists is the product of the number of resolutions for each conflict. In general, during the execution of a task system, a conflict exists each time a task is assigned to a processor. A large number of CCG models would be needed to find the best schedule or even a good schedule. The methodology also does not adapt easily for the evaluation of varying initial resource conditions. The whole process of conflict resolution, the conversion to a CCG and generating schedules must be repeated for each set of initial resources.

The scope of the work of Tani et. al. is more applicable to the problem which is addressed by Shapiro and Saint. The transitions of a CCG are allowed to fire an arbitrary number of times. Due to the choice of a restricted class of Petri nets as a basis for the model, the limited ability to adequately model resource contention restricts the use of the model. The use of the coloured Petri nets as a basis for the model in the present work provides the ability to model resource contention and the ability to model hierarchies in the resources. The present work defines a formulation which is more flexible model if resource contention is an important factor.

CHAPTER 4

THE ANALYSIS OF A SCHEDULING PROBLEM

The scheduling problem to be addressed here is the execution of a task system on a set of processors of different speeds. The scheduling strategy to be considered here is list scheduling. The performance criteria for the scheduling problem is the minimization of the execution time for the task system.

List scheduling in a multiprocessor environment is well-studied [1, 14, 15, 16, 24, 26, 27, 28, 30, 57, 56]. List scheduling is known to yield good performance for the case of identical processors [1]. Three methods of generating ordered lists of tasks are compared, highest levels first, Algorithm A of Coffman and Graham, and a proposed heuristic, maximum chain length maximum time. The three heuristics are compared using sample precedence graphs from the literature and from randomly generated precedence graphs.

Much of the research in the use of list scheduling for processors of different speeds is in the area of upper bounds on the performance of the list scheduling. A summary of the bounds is shown in Table 4.1. From the work of Liu and Liu [56], the bound suggests poor performance for sets of processors in which the difference between the fastest and slowest speed is large. A much tighter bound is presented for the case of a null precedence relation, that is, independent

Partial Order	Bound w/w*	Reference
arbitrary	$1 + \frac{b_1}{b_n} - \frac{b_1}{B_n}$	Liu and Liu [56]
ϕ	$\frac{3}{2} - \frac{1}{2n}$	Gonzales et. al. [28]
arbitrary	$1 + 2\sqrt{n}$ $\sqrt{n} + O(n^{1/4})$	Jaffe [40]

Notes

1. n is the number of processors.
2. b_i is the speed of processor i.
3. B_n is processing power of n processors.

Table 4.1: Bounds for list scheduling on processors of different speeds

tasks. The bound of Jaffe is a bound for the execution of any task system on a machine of n processors using the i fastest processors for executing the task system [40]. The i fastest processors are found by the minimization of $B_n/B_i+b_1/b_i$. The bounds suggest poor performance for a general list.

4.1 The Scheduling Model

The scheduling model for the problem is a subset of the model presented in Section 3.1. The resources for the problem are restricted to a single set of processors, $P=\{P_1,P_2,...,P_n\}$. The task system is completely as described in Section 3.1. The performance criteria to be used is the minimization of the execution time of the task system.

The set of processors, P, is a set of processors of different speeds. The speed of a processor i, b_i, is the speed of execution relative to the speed of the fastest processor. P_1 is assumed to be the fastest processor of speed $b_1=1$. The speed of the remaining processors is restricted to the range $1\geq b_i>0$. The set of processors is assumed to be ordered , such that,

$$b_1\geq b_2\geq ...\geq b_n.$$

A measure of a set of processors is the processing power, B. The processing power of the i fastest processors, B_i, is the sum of the speeds of the first i processors. The processing power of P is B_n.

The set of processors P, is modeled by a colour set, $C_P=(X_P,<_P)$. Each colour, c_i, in the set X_P represents processors of a certain speed. Since it is assumed that the set P is ordered, the colour set models the ordering. For convenience, c_1 represents the processors which have a speed of $b_i=1$. In the colour set, a larger subscript indicates a slower processor.

The list scheduling assigns tasks to the fastest available processor. The assignment of the tasks to the fastest processors is modeled by the partial order, $<_P$. Under the assumptions for coloured Petri nets, the least enabling coloured token from each input place is used to fire a transition. The partial order has c_1 at the bottom and c_k at the top, where k is the number of different speeds and c_k is the slowest processors. The general form of the partial order is shown in Figure 4.1. The partial order is from top to bottom. Thus, the least enabling coloured token from the processor place represents the fastest processor available at the time of firing the transistion.

The execution times of the tasks are $\{\tau_i\}$, $1 \le i \le r$. τ_i is a general execution time for the task. The execution time of a task i on processor j is given by τ_i/b_j. The processors are assumed to differ only in their speed of execution, hence, τ_i/b_j is valid for all processors.

List scheduling assigns the execution of a task to a

$C_p = (X_p, <_p)$

$X_p = \{c_1, c_2, \ldots, c_k\}$

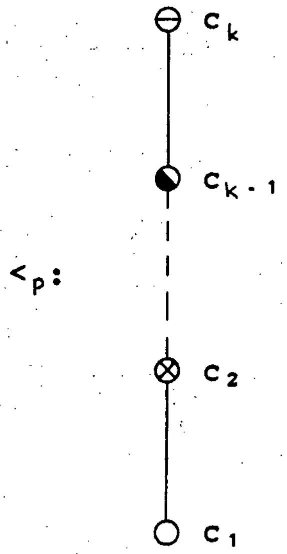


Figure 4.1: Colour set for processors of different speeds

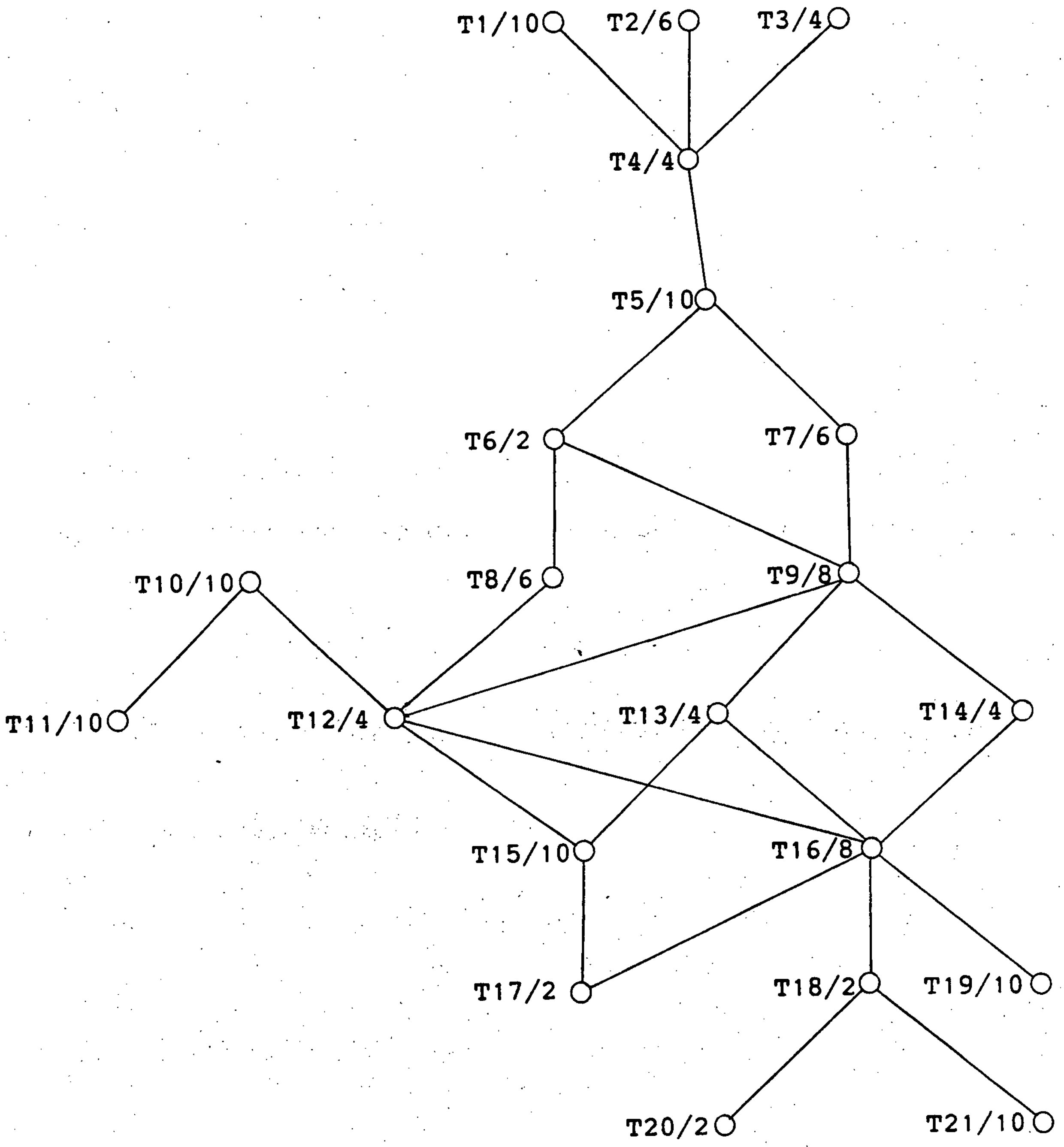
processor by scanning a list of tasks. The tasks in the list, L, are assumed to be in order of decreasing priority. A task closer to the head of the list is of higher priority than a task near the tail of the list. When a processor becomes available to execute a new task, the list is scanned from head to tail. As tasks which are ready to execute are found on the list, they are assigned to the fastest processor available. The assignment continues until there are no processors available or until there are no tasks ready to execute. List scheduling is relatively simple but relies on the ordered list to yield good results.

A characteristic of list scheduling is that no processors are idle if there are tasks ready to execute. This is a cause of non-optimality of list scheduling in general. An optimal schedule may contain tasks which are are delayed in order to "fit" better. This problem is compounded for processors of different speeds. For a simple case, suppose the next task on the list is the last task to be executed and the tasks is assigned to a slow processor. If the slow processor extends the execution time of the task beyond the time of waiting for a faster processor plus the execution time on the faster processor, then, a better schedule would result from delaying the task until the faster processor becomes available.

4.2 Algorithms for List Ordering

The effectiveness of list scheduling is dependent upon the prioritized list L, which is scanned to find the next task(s) to be executed. Three heuristic algorithms for the generation of the list L, are to be compared for the problem considered here. The highest levels first, HLF, heuristic is known to perform well in the case of identical processors [1]. The second heuristic to be compared is Algorithm A of Coffman and Graham, CG [14]. The lists generated by Algorithm A produce optimal schedules for the case of identical processors and unit execution times. A heuristic is proposed for the case of processors of different speeds which considers the effects of the execution times and the processor speeds.

The following definitions are necessary for the discussion of the heuristics, [40]. A chain is a sequence of tasks for which each task of the chain is the immediate sucessor to the task preceeding it in the chain. In Example 1 shown in Figure 4.2, C=(T5,T6,T9,T14) is a chain. The length of a chain is the sum of the execution times of the tasks in the chain. The height of a task T is the length of the longest chain starting with the task T. The height of task T10 in Example 1 is 34. The height, h, of a task system, $(\mathcal{T},<,\{\tau_i\})$, is the length of the longest chain starting with $T \in \mathcal{T}$. The height, h, of Example 1 is 62. The subsequent chain of a task T is the longest chain starting with an immediate sucessor of task T. The subsequent chain of task T5 in Example 1 is (T7,



task/execution time

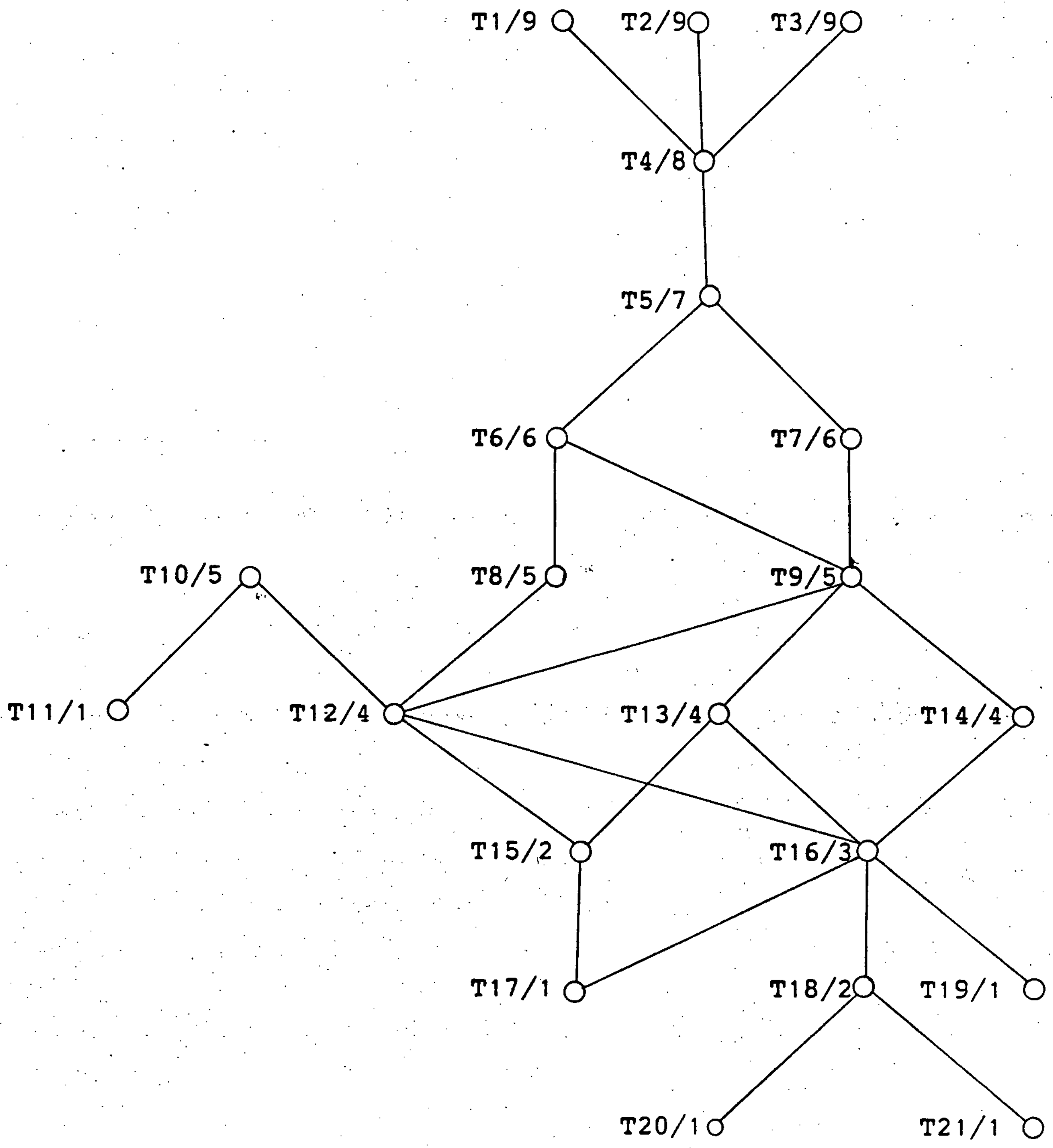
Figure 4.2: Example 1

T9, T13, T16, T18, T21). μ, the total execution time for a task system, is the sum of the execution times of the tasks in $\mathcal{T}$. μ for Example 1 is 132.

The highest levels first heuristic orders the tasks in a list in decreasing order of levels. The level of a task T is defined simply by the cardinality of the longest chain starting with task T. Figure 4.3 shows the levels of the tasks in Example 1. Tasks at the same level are ordered in the list arbitrarily. Since in general a task system has more than one task at each level, there are many lists which may result from a highest levels first ordering on the tasks.

Algorithm A of Coffman and Graham assigns an integer, $1 \le \alpha(T) \le r$, to each task T of $\mathcal{T}$. The tasks are ordered in a list according to decreasing $\alpha(T)$. Algorithm A is shown in Figure 4.4. S(T) is defined to be the set of immediate sucessors of task T. Note that in Algorithm A arbitrary decisions must be made only when S(T)=S(T'). Thus, Algorithm A in general does not generate a unique list L. An $\alpha(T)$ assignment for Example 1 is shown in Figure 4.5.

The proposed heuristic, maximum chain length maximum execution time, MCLMT, combines notions from two heuristics, highest levels first and longest processing time. From the highest levels first heuristic, a task which has a longer chain should be of higher priority, such that, it is assigned to a faster processor than a task with a shorter chain length.



task/level

Figure 4.3: Example 1 levels

1. An arbitray task T_0 with $S(T_0)=\phi$ is chosen and $\alpha(T_0)$ is defined to be 1.

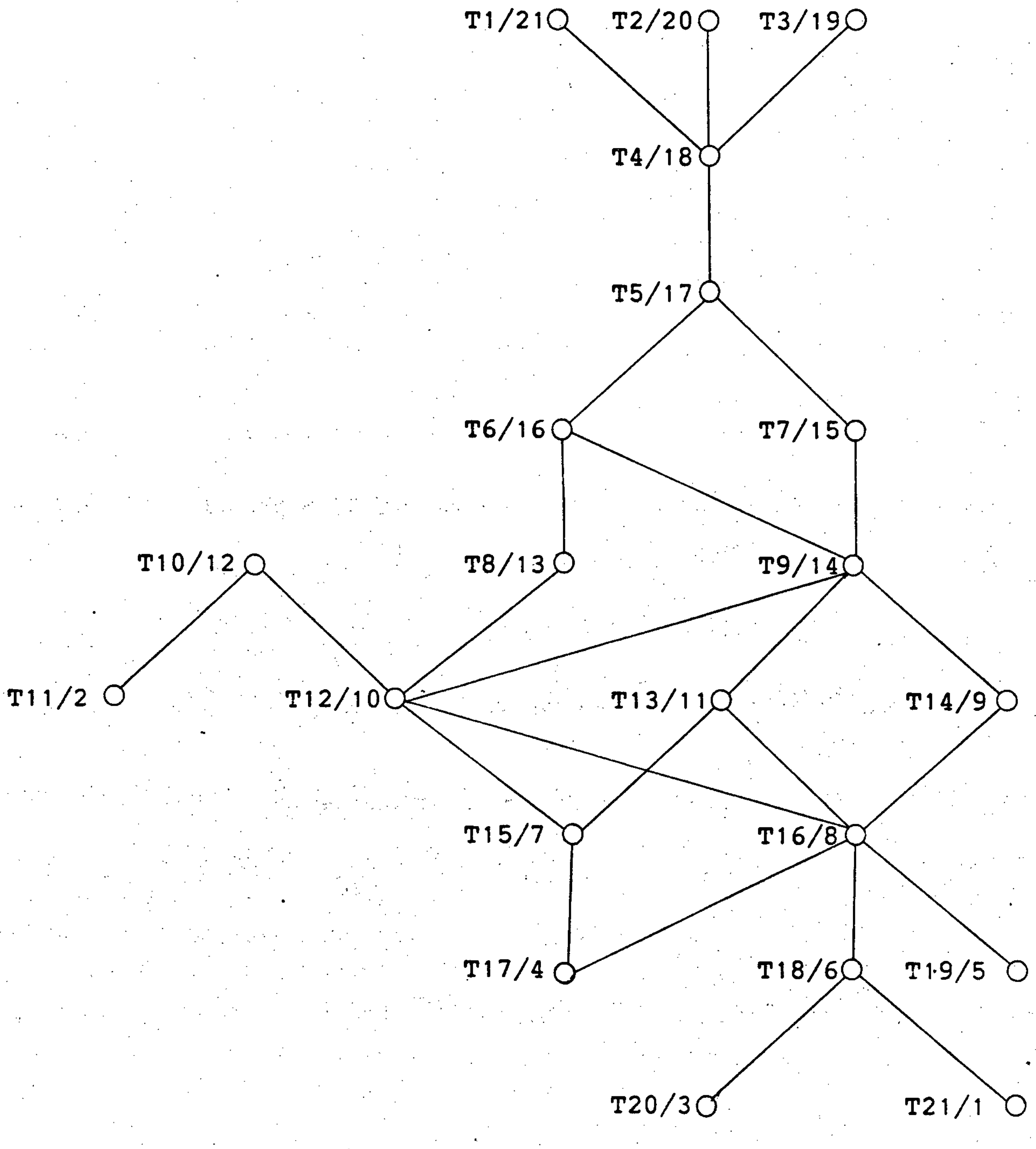
2. Suppose for some $k \le r$, the integers $1,2,\ldots,k$ have been assigned. For each task T for which α has been defined on all elements of S(T), let N(T) denote the decreasing sequence of integers formed by ordering the set $\{\alpha(T'):T' \epsilon S(T)\}$. At least one of these tasks T* must satisfy $N(T^*) \le N(T)$ for all such tasks T. Choose one such T* and define $\alpha(T^*)$ to be k.

3. Repeat the assignment in Step 2 until all tasks of $\mathcal{T}$ have been assigned some integer.

where $N=(n_1,n_2,\ldots,n_t)<N'=(n_1,n_2,\ldots,n_{t'})$, $t,t' \ge 0$,

if (i) for some $i \ge 0$, $n_j=n_j$, for all j, $1 \le j \le i-1$ and $n_i<n_i'$.
or (ii) $t<t'$ and $n_j=n_j'$, $1 \le j \le t$.

Figure 4.4: Coffman-Graham Algorithm A



task/σ(T)

Figure 4.5: Example 1, CG Algorithm A

For example, consider the tasks on the chain defining the height of the task system. If there is sufficient processing power, then, h/b_1 is the lower bound of the optimal schedule w^*. All the tasks of the longest chain must be executed on the fastest processor if the bound is to be achieved. From the longest execution time heuristic, the tasks are ordered by decreasing execution time for the case of equal chain lengths. It is easily shown that the shortest execution time is achieved by assigning the task with the longest execution time to the fastest processor available.

For the derivation of the proposed heuristic, MCLMT, consider some point in the execution of the task system. Suppose two processors, i and j, are available to execute new tasks. The speeds of the processors, i and j, are b_i and b_j, respectively. Processors i is assumed to be faster than processor j, $b_i > b_j$. Two tasks, T_k and T_l, are ready to execute and are to be assigned to the processors. The execution times for the tasks are τ_k and τ_l, respectively.

To derive the first sorting condition of the heuristic, define a quantity, Δ. Let the chain length of the tasks T_k and T_l be l_k and l_l, respectively. Suppose task T_k is assigned to execute on processor i and task T_l is assigned to execute on processor j. Let δ_{ki} be the length of the chain of task T_k when it is assigned to the processor i.

$$\delta_{ki} = l_k - \tau_k + \tau_k/b_i$$

To consider only the effect of the current assignment, assume the remainder of the chain is executed on the fastest processor. Similarily for task T_l,

$$\delta_{lj} = l_l - \tau_l + \tau_l/b_j$$

Consider the difference, Δ, between the chain lengths.

$$\Delta=\delta_{ki}-\delta_{lj}$$

$$\Delta=l_k-\tau_k+\tau_k/b_i-(l_l-\tau_l+\tau_l/b_j) \quad (1)$$

If the assumption that T is assigned to processor i is valid, then, $\Delta>0$.

$$(l_k-\tau_k+\tau_k/b_i)-(l_l-\tau_l+\tau_l/b_j)>0 \quad (2)$$

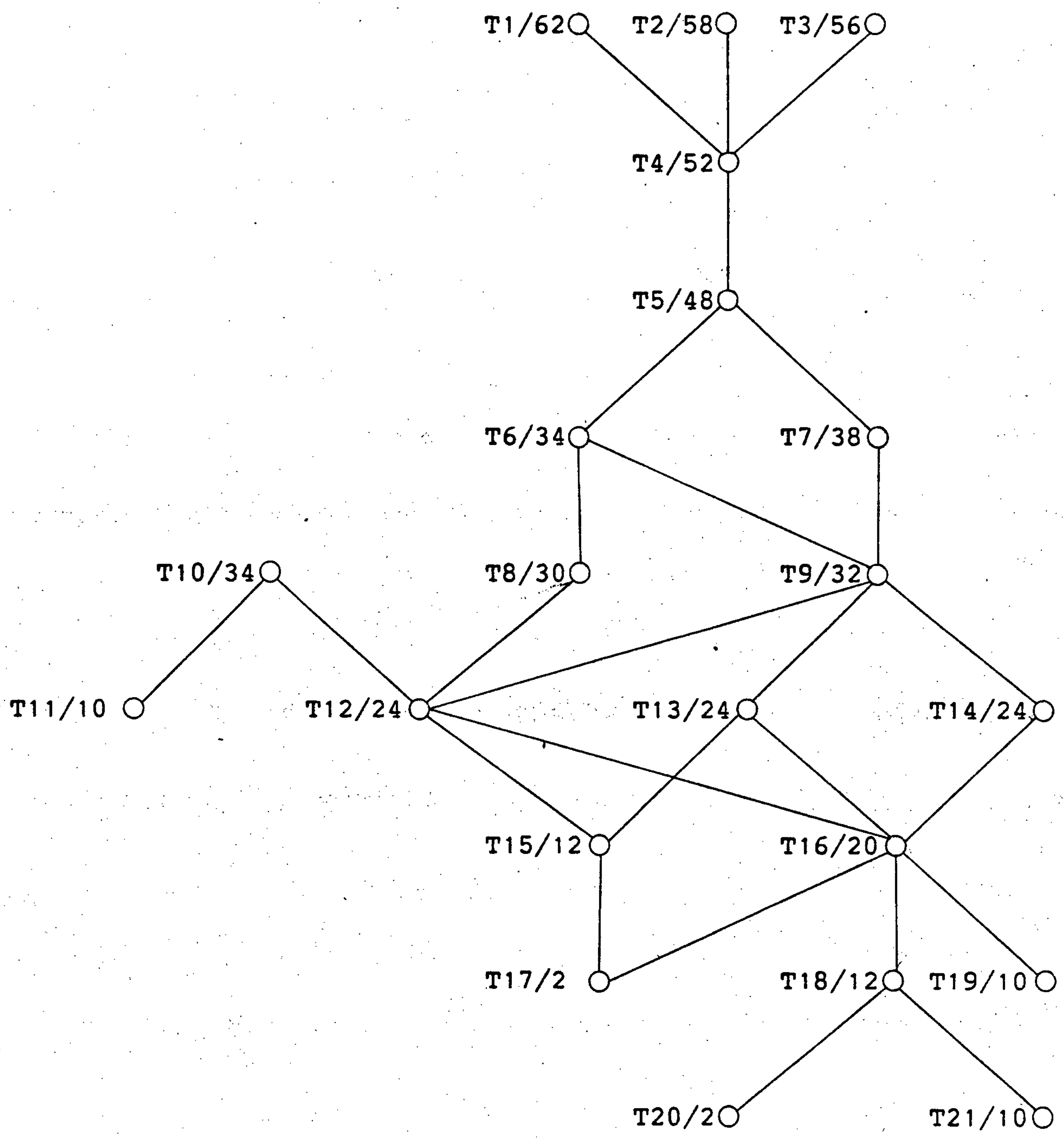
$$(l_k-l_l)>\tau_k-\tau_k/b_i-(\tau_l-\tau_l/b_j) \quad . \quad (3)$$

Equation (3) states that the the difference in the chain lengths of the tasks must be greater than the difference between the amount the execution times are extended by the execution on processor i or j. Since, the list should be ordered such that task T_k is higher priority, therefore, equation (3) may be interpreted to order the list in decreasing chain lengths. The chain lengths for Example 1 are shown in Figure 4.6.

The second criteria for sorting the list of tasks resolves the case of equal chain length. Consider the situation with two processors ready to execute two tasks, as described above. Assume task T_k is assigned to processor i and task T_l is assigned to processor j. From equation (1),

$$\Delta=\tau_k(1-1/b_i)-\tau_l(1-1/b_j).$$

If the assumption that task T_k is assigned to processor i is valid, then $\Delta>0$.



task/chain length

Figure 4.6: Example 1 chain length

$$\tau_k/\tau_l > (1-1/b_j)/(1-1/b_i)$$

But, $|1-1/b_j| > |1-1/b_i|$

Hence, $(1-1/b_j)/(1-1/b_i) \geq 1$

Therefore, $\tau_k/\tau_l \geq 1$,

and, $\tau_k \geq \tau_l$.

The second criteria is to order the tasks in order of decreasing execution time when the chain lengths are equal.

Note equation (1) can also be interpreted to order the list L according to decreasing subsequent chain length. Similarily, a second sorting criteria is the execution time of the tasks. The maximum subsequent chain length maximum execution time, MSCMT, heuristic and the maximum chain length maximum execution time, MCLMT, heuristic are identical, in general, except in certain situations when either heuristic may yield the wrong schedule.

Consider two processors ready to execute tasks and two tasks, T_k and T_l, as above. Suppose the chain length of T_k is greater than the chain length of T_l.

$$l_k > l_l$$

Further suppose the subsequent chain length of task T_k is less than the subsequent chain length of T_l.

$$l_k - \tau_k < l_l - \tau_l$$

Using the MSCMT list, T_l has a higher priority and is assigned to a faster processor. Using the MCLMT list, T_k has a higher priority and is assigned to a faster processor. Either list may result in a worse schedule than the reversed schedule depending on the differences in the lengths and the differences in the processor speeds. This is a possible source of non-optimality for either heuristic.

The heuristic reduces to simpler forms if restricted problems are considered. If the execution times are restricted to be unit times, then the heuristic reduces to highest levels first. Each task can only add a single unit to the length, therefore, the length is identical to the level. If the partial order is restricted to a null partial order, then, the heuristic reduces to a longest processing time heuristic. A null partial order means the tasks are independent. In this case the subequent chain lengths of all tasks is 0. The tasks are then, ordered using the second criteria into a list of decreasing execution times.

The lists for Example 1 resulting from each of the three heuristics and a random list are shown in Figure 4.7. The lists are sorted from the random list shown. The Gantt charts of the schedules for the execution of Example 1 on a set of processors $P=\{P_1,P_2,P_3,P_4\}$ are shown on Figures 4.8 and 4.9. The speeds of the processors are $b_1=1$ and $b_2=b_3=b_4=2/3$.

(T1 T2 T3 T4 T5 T7 T10 T6 T9 T8 T14 T12 T13 T16 T15 T18 T19 T11 T21 T17 T20)

(a) MCLMT

(T1 T2 T3 T4 T5 T7 T6 T9 T8 T10 T14 T12 T13 T16 T15 T18 T19 T17 T11 T20 T21)

(b) HLF

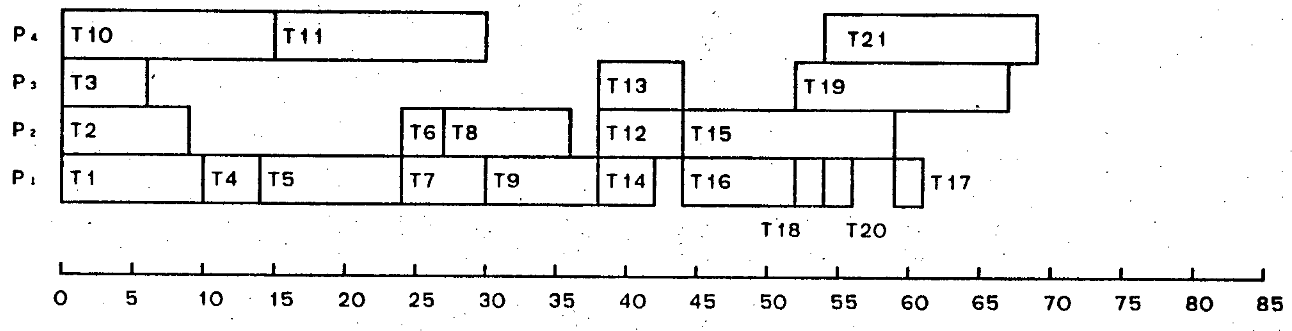
(T2 T1 T3 T4 T5 T6 T7 T9 T10 T8 T13 T12 T14 T16 T15 T18 T19 T17 T20 T11 T21)

(c) CG

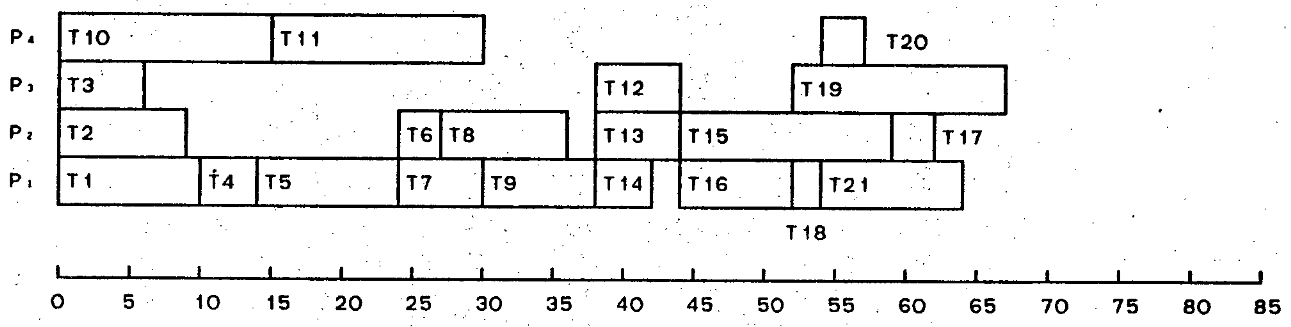
(T6 T13 T12 T18 T3 T5 T21 T2 T1 T16 T20 T14 T10 T11 T4 T17 T7 T8 T9 T19 T15)

(d) RANDOM

Figure 4.7: Ordered lists of tasks for Example 1

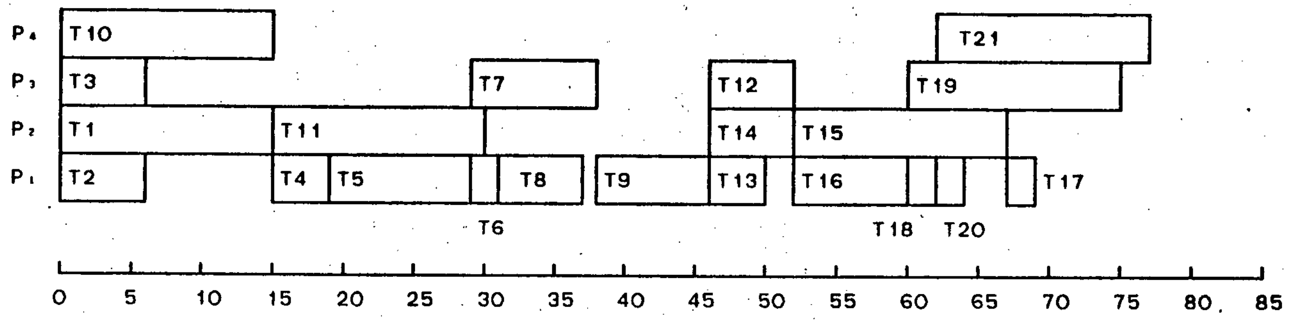


(a) HLF

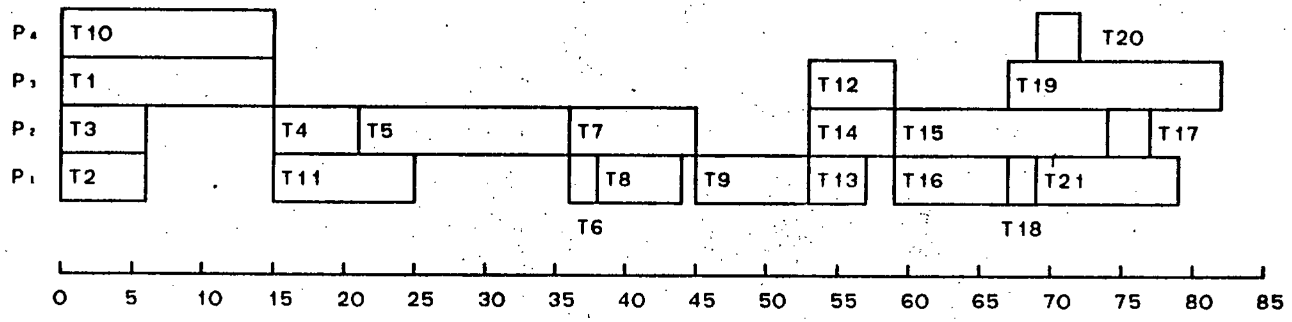


(b) MCLMT

Figure 4.8: Schedules for Example 1



(a) Coffman-Graham



(b) Random

Figure 4.9: Schedules for Example 1

4.3 Experimental Outline

The GRID analysis algorithm and the additional programs for the comparison of the heuristics are implemented in LISP. The symbolic computational capabilities of LISP allow the retention of the grammar-style definition of the transitions (see Appendix A and Figure 2.1(b)). The grammar-style definitions also allow the use of self-loops, an essential part of the Petri net formalization.

Transitions can also be defined as vectors of length n, where n is the number of places. A vector is defined for the input places and also for the output places. The non-zero elements of the vector indicate which places are connected to the transition. Since in general, a transition is connected to a small number of places, relative to n, many of the elements of the vector are zero.

The three heuristics described in Chapter 4.2, and a randomly generated list, are compared for 15 partial orders. The sources of the partial orders are as noted in Table 4.3. Three special cases of the partial orders are used in the comparison, a null partial order, a most constrained partial order and a least constrained partial order, the latter two partial orders are shown in Figures B.1 and B.2 of Appendix B. The size of the task set, $\mathcal{T}$, ranges in size from 10 tasks to 85 tasks. Four randomly generated partial orders of sizes, 40 tasks, 50 tasks, 65 tasks and 75 tasks, are included for the comparison.

Each task in the partial orders is assigned an execution time chosen from a uniform distribution between a minimum execution time and a maximum execution time. The execution time is assumed to be a positive integer. For the examples taken from other sources, the comparisons are made for the defined execution times in addition to the random execution times if the execution time are not defined to be unit execution times.

The colour set, X_ρ, which is used for the comparisons is shown in Table 4.2. The assumption that the speeds are rational numbers and the execution times are positive integers avoids the problems associated with finite machine arithmatic for real numbers. This assmption is made without loss of generality.

A task system is defined by a task set, a partial order, and the execution times for the tasks. The tasks of the partial orders are assigned different execution times, thus creating several task systems based on a single partial order. For each task system, the schedule is found for its execution on several different sets of processors. In terms, of the Petri net formalization, the Petri net is executed with several initial markings, m0, of the form,

$$m0=(\text{READY } c_1\ c_i\ \ldots\ c_j).$$

Each initial marking is a list of tokens with a single READY

Colour	Processor Speed
c_1	1
c_2	2/3
c_3	1/2
c_4	1/3
c_5	1/5
c_6	1/6

Table 4.2: Colour set for experiments

token and two or more processor tokens. The READY token is necessary for the portion of the Petri net modeling the data dependencies (see Step 4 of Algorithm 1). c_1 and c_i are elements of the colour set, X_p. At least one token in the initial marking is the token c_1, since it is assumed that the fastest processor has its speed normalized to 1.

4.4 Results and Discussion

A summary of the results is shown in Table 4.3. n_0 is the number of different sets of processors tested for the task systems based on the partial order. n_1 is the number of times the list from the heuristic generates the schedule with the shortest execution time. n_2 is the number of times the list from the heuristic generates a schedule as good as at least one other list.

The maximum chain length maximum time, MCLMT, list is found to generate a schedule whose length is shorter than the length of the schedules which are generated by the other heuristics in 98 of 160 cases for the fifteen examples. The MCLMT list is also found to generate a schedule whose length is no longer than one or more of the heuristics compared in 39 of 160 cases tested. Overall, the MCLMT list generated the best schedule of the four lists compared 86 per cent of the cases tested.

Note Example 1 accounts for over 20 per cent of the cases

Example	r	n_0	MSCMT		HLF		CG		RANDOM		Notes
			n_1	n_2	n_1	n_2	n_1	n_2	n_1	n_2	
1	21	34	23	9	0	7	2	2	0	2	Example 2 less 1
2	22	11	10	1	0	1	0	0	0	0	[14] p. 88
3	19	21	9	3	4	0	4	3	1	0	[24]
4	85	10	6	2	1	1	1	1	0	0	[57]
5	16	6	5	0	1	0	0	0	0	0	<:ϕ
6	10	13	4	6	0	1	3	4	0	6	[15] p. 6
7	12	20	13	6	0	2	1	6	0	1	[1]
8	16	6	2	2	0	2	2	2	0	0	<:most constrained
9	15	6	5	1	0	1	0	1	0	1	<:least constrained
10	19	7	2	4	1	4	0	1	0	1	[14] p. 92
11	30	6	3	3	0	3	0	2	0	0	[59]
12	40	6	5	0	1	0	0	0	0	0	Randomly generated
13	50	5	5	0	0	0	0	0	0	0	Randomly generated
14	65	6	6	0	0	0	0	0	0	0	Randomly generated
15	75	3	1	1	1	1	0	1	0	0	Randomly generated
Total		160	98	39	9	23	14	23	1	11	

Notes

1. r is the number of tasks.
2. n_0 is the number of cases.
3. n_1 is the number of best schedules.
4. n_2 is the number of schedules at least as good as one other.

Table 4.3: Sumary of Results

tested. Since, Example 1 is Example 2 less a single independent task, consider only the results for examples 2 through 15. In this case, the MCLMT list is found to generate the shortest schedule in 75 of 127 cases and to generate a schedule at least as good as one other list in 30 of 127 cases. The MCLMT is found to generate the best schedule in 83 per cent of the cases if Example 1 is ommitted.

The results are heavily weighted by task system with fewer than 20 tasks. Examples 2, 4, and 11-15 are the examples in which the number of tasks ranges from 20 to 85, excluding Example 1. For these Examples, it is found that the MCLMT list generated the shortest schedule in 35 of 47 cases and a schedule at least as good as one other list in another 8 of 47 cases. For the larger task systems, the MCLMT is found to generate the best schedule in 91 per cent of the cases which were tested.

The lengths of the schedules generated by each of the heuristics is compared to Jaffe's lower bound on the length of an optimal schedule. The ratio is an indication of the absolute preformance of the heuristics. Jaffe's bound is used since the bound is easily calculated. The ratios w/w*' are shown in Table 4.4. The lower bound, w*', is defined by the following lemma.

Example	r	MCLMT	HLF	CG	RANDOM
1	21	1.17	1.35	1.35	1.38
2	22	1.10	1.18	1.27	1.38
3	19	1.26	1.50	1.30	1.42
4	85	1.14	1.20	1.19	1.37
5	16	1.05	1.15	1.25	1.29
6	10	1.21	1.45	1.29	1.35
7	12	1.19	1.71	1.26	2.00
8	16	1.53	1.53	1.65	1.76
9	15	1.13	1.22	1.20	1.19
10	19	1.11	1.11	1.28	1.41
11	30	1.08	1.11	1.15	1.21
12	40	1.41	1.57	1.54	1.61
13	50	1.00	1.04	1.02	1.15
14	65	1.04	1.13	1.14	1.15
15	75	1.05	1.06	1.08	1.22

where $w^{*\prime}$ is the lower bound on the length of the optimal schedule

Table 4.4: Average $w/w^{*\prime}$

Lemma 1 [40]

> Let $(\mathcal{T},<,\mu)$ be a task system to be scheduled on a set of processors of different speeds. Let w* be the finishing time of an optimal schedule. Then,
>
> $$w^* \geq \max(\mu/B_n, h/b_1).$$

The ratios given in Table 4.4 are a simple mean of the cases tested for each example. The actual value of w/w*' ranges in value from 1.00 to 2.31. In general, the values are found to be in the range from 1.00 to 1.50.

The ratio w/w*' is an indicator of performance for the list ordering heuristics compared. A large value of w/w*' may be caused by two reasons. Under certain conditions, any list used for list scheduling results in poor performance. This case is to be discussed later in the specific test on Example 1. The second cause of a large value of w/w*' is that the lower bound may underestimate the length of an optimal schedule.

Jaffe's bound is a very loose bound for certain types of operational precedence constraints. If the partial order is very highly constrained, such as Example 8, then, there are many chains which are of approximately the same length as the chain defining the height, h. Consider the case where several such chains differ from the longest chain by only a few tasks. Assume the set of processors includes only a single fast processor and the bound for the optimal is defined by h/b_1. If the tasks on the longest chain are executed on the fastest processors, then the remaining tasks on the chains which share tasks with the longest chain must be executed on a slower processor so delaying the execution of the shared portions of the longest chain. Thus, an optimal schedule for the set of processors may be longer than the length of the longest chain in this case. An example of this situation is discussed later.

Table 4.5 is an example of a test run for a particular assignment of execution times for Example 1. The lengths of the schedules are compared to Jaffe's lower bound for the length of the optimal, w*', and the length of an optimal schedule, w*. The length of an optimal schedule is included as a comparison for Jaffe's lower bound.

In general, the problem of finding an optimal schedule is intractable. The structure of the precedence constraints of Example 1 and the small size of the set of tasks enables the construction of an optimal schedule. The method of construction of an optimal schedule is similar to the method of Fernandez and Bussell [25] for finding the lower bound in the case of identical processors. In general, the following procedure amounts to an exhaustive search.

The construction of an optimal begins with the inspection of the Gantt chart of the best schedule resulting from the four lists. The task assignments are swapped if the swapping shortens the length of the schedule. The swapping continues until no further improvemant can be made.

Processors	Execution Time w					
	MCLMT	HLF	CG	RANDOM	w*′	w*
C_1 C_1	70	72	72	72	66	68
C_1 C_1 C_1	62	64	64	62	62	62
C_1 C_2	88	88	95	93	80	82
C_1 C_2 C_2	67	67	82	82	62	67
C_1 C_2 C_2 C_2	67	69	77	82	62	67
C_1 C_3	90	92	92	92	88	90
C_1 C_3 C_3	78	94	78	94	66	78
C_1 C_3 C_3 C_3	78	80	92	104	62	78

Table 4.5: Results for Example 1

As an example, it is shown that the length of an optimal schedule for the set of processors defined by (c_1, c_2, c_2, c_2) is 68. From the Gantt chart shown in Figure 4.7(b), the task system can be partitioned into three partitions. T1 to T11, T12 to T14 and T15 to T21. There is no overlap between the execution of the tasks in each of the partitions except for tasks T10 and T11 due to the precedence constraints. T10 and T11 may overlap the second and third partitions but if they are executed as early as possible, then, there is no overlap. The length of the optimal schedule is the sum of the lengths the optimal schedule for each partition.

In the first partition, the longest chain is (T1, T4, T5, T7, T9). The longest chain is of length 38 units. Each task in the chain is exectuted on the fastest processor. All other tasks in the partition is executed on the slower processors and they complete execution before the tasks on the longest chain complete execution. Therefore the tasks not on the longest chain do not delay the the tasks on the longest chain. Since the partition cannot be executed in less time, the schedule of length 38 units is optimal.

In the second partition there are sufficient processors for each task to be executed concurrently. The tasks are of equal execution times and two of the tasks are assigned to the slow processors. Note that no benefit can result from delaying the execution of any of the three tasks since the the execution of two of the tasks on the fast processor would result in an execution time of 8 units. Hence, the optimal is 6 units.

In the third partition, consider the chains C1=(T16,T18,T21) and C2=(T16,T19). Chain C1 is of length 20 units and chain C2 is of length 18. Since only one of the chains may be executed fully on the fast processor, either T19 or T18 and T21 must be executed on a slower processor. Assume T16 is executed on the fast prcessor since it is common to both chains. If T19 is executed on a slower processor, then, the actual execution time for chain C2 is 23 units and for chain C1 is 22 units. If tasks T18 and T21 are executed on the slower processor, then, the actual execution time for chain C1 is 28 and for chain C2 is 20. Hence, the execution time is minimized if T19 is executed on a slower processor. Note the execution of tasks T15, T17 and T20 does not affect the execution time of the partition since the tasks may be executed serially on a slow processor in 21 units of time. The length of the optimal schedule for the third partition is 22 units.

The length of the optimal schedule is the sum of the lengths of the optimal schedules for each partition.

$$w^*=38+6+22=67$$

The length of an optimal schedule for the execution of Example 1 on the set of processors defined by (c_1, c_2, c_2, c_2) is 68 units.

A comparison between Jaffe's lower bound and the length of an optimal schedule indicates that, in general, the bound is relatively close to the length of an optimal schedule. However, note that for the last two sets of processors tested. The length of an optimal schedule exceeds the bound by a significant amount, $w^{*'}/w^{*}=1.26$. In this case, the bound yields a poor estimate of the length of an optimal, as described above. The tasks of Example 1 are highly constrained as illustrated in Figure 4.2. The second condition, that a single fast processor is included in the set of processors, is also met in the last two cases. This behavior is verified using the other precedence relations.

The processor speeds and the maximum execution times are arbitrary parameters in the experimental procedure outlined in Chapter 4.3. Sample results for test runs to examine the effect of altering these parameters are shown in Appendix B. The parameters are changed to alter the ratios, τ_{max}/τ_{min} and b_1/b_n. In the first test, $\tau_{max}/\tau_{min}=b_1/b_n$ and longer execution times and slower speeds are used. In the second test, $\tau_{max}/\tau_{min}<b_1/b_n$ is achieved by using very slow processors. In the third test, $\tau_{max}/\tau_{min}>b_1/b_n$ is achieved by the use of a large value for τ_{max}. In each test, the parameters did not significantly affect the dominance of the proposed heuristic, MCLMT.

The final test on Example 1 illustrates the restriction to unit execution times. Note that the HLF and the MCLMT lists generate the same schedule. This behavior is verified with the other precedence relations. The schedules for the sets of processors including c_1 are further examples of situations in which the list scheduling strategy yields poor performance. The Coffman and Graham Algorithm A list generates a schedule of the same length as the HLF list and the MCLMT list. This behavior is due to the unit execution times and the relatively simple structure of the partial order.

A final observation in the results is a comparison between the schedules generated by the MCLMT list and the MSCMT list, described in Chapter 4.2. Over the 160 cases tested, the MSCMT list and the MCLMT list generated schedules of the same length in 121 of the cases. In 20 of the cases the MCLMT list generated a better schedule and in 19 of the cases the MSCMT list generated a better schedule. This agrees with the notion that the two heuristics are almost identical.

CHAPTER 5
CONCLUSIONS AND SUGGESTIONS FOR FURTHER RESEARCH

5.1 Summary and Conclusions

A Petri net formulation for the general scheduling problem is developed in this thesis. The Petri net formulation is based upon three classes of Petri nets, marked graphs, timed Petri nets and coloured Petri nets. The general approach to the formulation begins by expressing the structure of the general scheduling problem in terms of a Petri net model. The scheduling strategy is modeled by modifying the algorithm for the analysis of the Petri net. A schedule is generated by the execution of the Petri net. The Petri net formulation may be thought of as the result of a transformation on the formal definition of the general scheduling problem.

The main advantages of the Petri net formulation are these of Petri nets in general. As presented in this thesis, the Petri net formulation has the ability to model a wide variety of resource and processor configurations. If a configuration arises which cannot be modeled under the formulation presented here, then the Petri nets are easily extended to improve their ability to model a particular situation. Schedules are easily generated for the execution of a task system on different sets of initial resources by simply executing the Petri net model with different initial markings. The Petri net formulation is a compact and natural description of the general scheduling problem.

The logical separation of the modeling of the structure and the modeling of the behavior in the scheduling problem allows changes to made in either the structure or the behavior of the problem without independently. Consider the problem analysed in Chapter 4, namely, list scheduling for a set of processors of different speeds. Suppose the processors are now to be functionally dedicated and to be of different speeds within a functional group. The set of processors is modeled by a set of coloured tokens but the analysis modeling the list scheduling remains unchanged. Suppose instead, a different scheduling strategy is desired to be modeled for the set of processors of different speeds. The GRID analysis is modified to model the scheduling strategy and the Petri net remains the same. The logical separation of the modeling of the behavior and of the structure of systems provides flexibility in the use of the formulation.

There are two disadvantages to their formulation. The Petri net formulation takes advantage of the assumption that a transition is defined to fire as soon as it is enabled, subject to conflict resolution. Since the tasks are modeled as transitions, the earliest schedules are generated under the Petri net formulation. The formulation does not include a natural representation for the modeling of arbitrary delays. A transition is also assumed to fire completely and to be uninterrupted once it begins firing. This notion prevents consideration of preemptive strategies. Note that both of these disadvantages may be overcome by modifications to the implementation of the behavioral analysis which is used for the study of the problem in Chapter 4.

The use of the Petri net formulation of the general scheduling problem is demonstrated in list scheduling for the execution of a task system on a set of processors of different speeds. A new heuristic is proposed for ordering the list of the tasks. The proposed heuristic, MCLMT, orders the list in order of decreasing chain length, and decreasing execution time in the case of equal chain lengths. The schedules which are generated by the proposed heuristic are compared to the schedules which are generated by the highest levels first heuristic, Algorithm A of Coffman and Graham, and a randomly generated list. In the comparisons, the proposed heuristic generated a better schedule in 98 of 160 cases tested and a schedule as good as at least one other schedule in an additional 39 of 160 cases.

5.2 Suggestions for Further Research

The major suggestion for further research is the further use of the Petri net formulation. The use of the formulation presented in this thesis is a relatively simple scheduling problem. The scheduling problem discussed list scheduling for a simple resource structure consisting of a single set of processors of different speeds.

The formulation may be used to evaluate scheduling strategies other than list scheduling. List scheduling, as noted in the comparison of the heuristics, is known to yield poor performance in the case where there exists a large difference the the speed of the fastest and slowest processor. The problem with list scheduling is that a task may be assigned to a slow processor when a better schedule would result if the execution of the task is delayed until a faster processor is available. This situation suggests a look-ahead strategy may perform better in this situation. A look-ahead strategy is within the modeling capabilities of the Petri net formulation as presented in Chapter 3.

The formulation may be used for the examination of the effects of more structured sets of resources. The set of processors which was used for the comparisons in Chapter 4 specified no interconnections between the processors. It is shown through the example of the dynamically reconfigurable architecture that the formulation is capable of modeling an arbitrary structure in the resources. The special relationship between the computing elements of the reconfigurable architecture is modeled naturally through the use of a set of coloured tokens and the transition schemes. Other relations in the resources, such as those imposed by a network architecture, may be modeled using a combination of colour sets for the resources, the transition schemes and the set of coloured data tokens.

The final suggestion for further research takes ideas from the scheduling problem to suggest changes in Petri net theory. The Petri net formulation is unable to model either arbitrary delays, or the partial firing of a transition which has a time duration associated with it. Extension of the Petri net theory to allow consideration of Petri nets which possess these modeling characteristics is desirable.

Bibliography

1. Adam, T.L., Chandy, K.M., Dickson, J.R., "A Comparision of List Schedules for Parallel Processing Systems," Comm. ACM, Vol. 17(12), Dec. 1974, p. 685–590.

2. Agerwala, T., "Some Applications of Petri Nets," Proc. National Electronics Conf., Vol. 32, 1978, p. 149–157.

3. Agerwala, T., Chued–Amphai, Y–C., "Synthesis Rule for Concurrent Systems," Proc. 15th Design Automation Conf., June 1978, p. 305–311.

4. Appelbe, W.F., Ito, M.R., "Scheduling Heuristics in a Multiprogramming Environment," IEEE Trans. Computers, Vol. C–27(7), July 1978, p. 628–637.

5. Araki, T., Kasami, T., "Some Decision Problems Related to the Reachability Problem for Petri Nets," Theor. Comput. Sci., Vol. 3, 1977, p. 85–104.

6. Araki, T., Kasami, T., "Decidable Problems on the Strong Connectivity of Petri Net Reachability Sets," Theor. Comput. Sci., Vol. 4, 1977, p. 99–119.

7. Ayache, J.M., Azema, P., Diaz, M., "Observer: A concept for On–line Detection of Control Errors in Concurrent Systems," Proc. 9th Int'l. Symp. Fault–Tolerant Computing, 1979, p. 79–86.

8. Azema, P., Valette, R., Diaz, M., "Petri Nets as a Common Tool for Design Verification and Hardware Simulation," Proc. 13th Design Automation Conf., 1976, p. 109–116.

9. Baer, J–L., Ellis, C.S., "Model, Design, and Evaluation of a Compiler for a Parallel Processing Environment," IEEE Trans. Software Engineering, Vol. SE–3(6), Nov. 1977, p. 394–405.

10. Berlin, F.B., "Time–extended Petri Nets," M.A. Thesis, University of Texas at Austin, 1979.

11. Berthelot, G., Roucairol, G., "Reduction of Petri Nets," Mathematical Foundations of Computer Science, 1976, p. 202–209.

12. Best, E., Schmid, H.A., "Systems of Open Pathes in Petri Nets," Mathematical Foundations of Computer Science, 1975, p. 186–193.

13. Brauer, W., ed., Net Theory and Applications, Springer—Verlag, (New York, 1980).

14. Coffman, E.G., Jr., Graham, R.L., "Optimal Scheduling for Two—Processor Systems," Acta Informatica, Vol. 1(1), 1972, p. 203—213.

15. Coffman, E.G., Jr., Denning, P.J., Operating Systems Theory, Prentiss—Hall, (Toronto, 1973).

16. Coffman, E.G., Jr., ed., Computer and Job Shop Scheduling Theory, John Wiley (Toronto, 1976).

17. Commoner, F., Holt, A.W., Even, S., Pnueli, A., "Marked Directed Graphs," J. Comput. Syst. Sci., Vol. 5(5), 1971, p. 511—523.

18. Contronis, J.Y., Lauer, P.E., "Verification of Concurrent Systems of Processes," Proc. Int'l. Computing Symp., 1977, p. 197—207.

19. Courvoisier, M., Valette, R., "Description and Realization of Parallel Control Systems," Proc. 15th IEEE Fall COMPCON, 1977, p. 167—172.

20. Courvoisier, M., Seek, J.P., "Hardware Implementation of Generalized Petri Nets," Electronic Letters, Vol. 15(24), 22 Nov. 1979, p.770—772.

21. Cox, L.A., Jr., "Predicting Concurrent Computer System Performance Using Petri Net Models," Proc. 1978 ACM Annual Conf., 1978, p. 901—913.

22. Crespi—Reghizzi, S., Mandroili, D., "Some Algebreic Properties of Petri Nets," Alta Frequenza, Vol. 45(2), Feb. 1976, p. 130—137.

23. Devy, M., Diaz, M., "Multilevel Specification and Validation of the Control in Communication Systems," Proc. Int'l. Conf. Distributed Computer Systems, 1979, p. 43—50.

24. Ecker, K., "Analysis of a Simple Strategy for Resource Constrained Task Scheduling," Proc. Int'l. Conf. Paralllel Processing, 1978, p. 181—183.

25. Fernandez, E.B., Bussell, B., "Bounds on the Number of Processors and Time for Multiprocessor Optimal Schedules," IEEE Trans. Computers, Vol. C—22(8), Aug. 1973, p. 745—751.

26. Garey, M.R., Graham, R.L., "Bounds for Multiprocessor Scheduling with Resource Constraints," SIAM J. Computing, Vol. 4(4), 1975, p. 187-200.

27. Garey, M.R., Johnson, D.S., "Complexity Results for Multiprocessor Scheduling Under Resource Constraints." SIAM J. Computing, Vol. 4(2), 1975, p. 397-411.

28. Gonzales, M.J., Jr., "Deterministic Processor Scheduling," ACM Computing Surveys, Vol. 9(3), 1977, p. 173-204.

29. Gonzales, T., Iberra, O.H., Sahni, S., "Bounds for LPT Schedules on Uniform Processors," SIAM J. Comput., Vol. 6(1), 1977, p. 155-166.

30. Goyal, D.K., "Scheduling Equal Execution Time Tasks Under Unit Resource Restriction," Proc. Int'l. Conf. Parallel Processing, 1978, p. 188-192.

31. Grabowski, J., "The Unsolvability of Some Petri Net Language Problems," Inf. Processing Lett. Vol. 9(2), 17 Aug. 1979, p. 60-63.

32. Grenrich, H.J., Lautenbach, K., "The Analysis of Distributed Systems by Predicate/Transition Nets," Semantics of Concurrent Computation, Springer-Verlag, (New York, 1979), p. 123-146.

33. Han, Y.H., "Performance Evaluation of a Digital System Using a Petri Net-Like Approach," Proc. National Electronics Conf., Vol. 32, 1978, p. 168-172.

34. Heimerdinger, W.L., Han, Y.H., "A Graph Theoretic Approach to Fault Tolerent Computing," Final Report, Air Force Office of Scientific Research, Contract No. F44620-75-C-0053, Sept. 1977.

35. Heimerdinger, W.L., "A Petri net Approach to System Level Fault Tolerence Analysis," Proc. National Electronics Conf., Vol. 32, p. 161-165.

36. Herzog, O., "Automatic Deadlock Analysis of Parallel Programs," Proc. Int. Computing Sypm., 1977, p. 209-216.

37. Herzog, O., "Static Analysis of Concurrent Processes for Dynamic Properties Using Petri Nets," Semantics of Concurrent Computation, Springer-Verlag, (New York, 1979), p. 66-90.

38. Irani, K.B., Zervos, C.R., "Modelling of Conflicts, Priority Hierarchies and Reentrancy in Concurrent Synchronization Structures," Proc. 1979 Int'l. Conf. Parallel Processing, 1979, p. 196-204.

39. Izbicki, H., "Marked Graphs With Generalized Firing Rules," Proc. Symp. Recent Advances in Graph Theory, 1974, p. 307-310.

40. Jaffe, J.M., "Efficient Scheduling without the Full Use of Processor Resources," Theor. Comput. Sci. Vol. 12(1), 1980, p.1-17.

41. Janicki, R., "A Characterization of Concurrency-like Relations," Semantics of Concurrent Computation, Springer-Verlag, (New York, 1979), p. 109-122.

42. Jantzen, M., "On the Hierarchy of Petri Net Languages," RAIRO Inf. Theor./Theor. Inf., Vol. 13(1), 1979, p. 19-30.

43. Jones, N.D., Landweber, L.H., Lien, Y.E., "Complexity of Some Problems in Petri Nets," Theor. Comput. Sci., Vol. 4, 1977, p. 277-299.

44. Karp, R.M., Miller, R.E., "Parallel Program Schemata," J. Comput. Syst. Sci., Vol. 3, 1969, p. 147-195.

45. Kartashev, S.I., Kartashev, S.P., "Dynamic Architecture: Problems and Solutions," Computer, Vol. 11(7), July 1978, p.26-40.

46. Kartashev, S.I., Kartashev, S.P., "A Multicomputer System with Dynamic Architecture," IEEE Trans. Computers, Vol. C-28(10), Oct. 1979, p. 704-721.

47. Kasami, T., Miller, R.E., "Homomrphisms Between Models of Parallel Computation," IBM Research Report, RC-7796, 1979.

48. Kwong, Y.S., "On Absence of Livelocks in Parallel Programs," Semantics of Concurrent Computation, Springer-Verlag, (New York, 1979), p. 172-190.

49. Lauer, P.E., Campbell, R.H., "A Description of Path Expressions by Petri Nets," Proc. 2nd ACM Symp. Principles of Programming Languages, 1975, p. 95-105.

50. Lauer, P.E., Campbell, R.H., "Formal Semantics of a Class of High Level Primatives for Coordinating Concurrent Processes," Acta Informatica, Vol. 5, 1975, p. 297-332.

51. Lautenbach, K., Schmid, H.A., "Use of Petri Nets for Proving Correctness of Concurrent Process Systems," Infomation Processing '74, 1974, p. 181-191.

52. Lien, Y.E., "A Note on Transition Systems," Information Sciences, Vol. 10, p.347-362.

53. Lien, Y.E., "Termination Properties of Generalized Petri Nets," SIAM J. Comput., Vol. 5(2), June 1976, p. 251-265.

54. Lipton, R.J., "Reduction: A Method of Proving Properties of Parallel Programs," CACM., Vol. 18(12), Dec. 1975, p. 717-721.

55. Lipton, R.J., "The Reachability Problem Requires Exponential Space," Yale University, Dept. of Computer Science, Research Report No. 62, Jan. 1976.

56. Liu, J.W.S., Liu, C.L., "Bounds on Scheduling Algorithms for Heterogeneous Computing Systems," Information Processing 74, North-Holland, (Amsterdam, 1974), p. 349-353.

57. Liu, J.W.S., Liu, C.L., "Performance Analysis of Multiprocessor Systems Containing Functionally Dedicated Processors," Acta Informatica, Vol. 10(1), 1978, p. 95-104.

58. Manacher, G.K., "Production and Stablization of Real-Time Task Schedules," J. ACM, Vol. 14(3), July 1967, p. 439-465.

59. Mandroili, D., "A Note on Petri Net Languages," Information and Control, Vol. 34(2), June 1977, p. 169-171.

60. Martin, D.E., Estrin, G., "Experiments on Computations and Systems," IEEE Trans. Electronic Computers, Vol. EC-16(1), Feb. 1967, p. 59-69.

61. Meiling, E., "On the Modelling Power of Petri Nets," Dept. of Computer Science, Cornell, TR79-403.

62. Merlin, P.M., "A Methodology for the Design and Implementation of Communication Protocols," IEEE Trans. Communications, Vol. COM-24(6), June 1976, p. 614-621.

63. Merlin, P.M., Farber, D.J., "Recoverability of Communication Protocols-Implications of a Theoretical Study," IEEE Trans. Communications, Vol. COM-24(9), Sept. 1976, p. 1036-1043.

64. Miller, R.E., Kasami, T., "Comparing Models of Parallel Computation by Homomorphisms," Proc. COMPSAC, 1979, p.794-799.

65. Memmi, G., "Notion de Duality et de Symetrie Dans les Reseaux de Petri," Semantics of Concurrent Computation, Springer-Verlag, (New York, 1979), p. 91-108.

66. Misunas, D., "Petri Nets and Speed Independent Design," CACM, Vol. 16(8), Aug. 1973, p. 474-480.

67. Murata, T., "State Equation, Controlability and Maximal Marking of Petri Nets," IEEE Trans. Automatic Control, Vol.AC-22(2), June 1977, p. 412-416.

68. Murata, T., "Circuit Theoretic Analysis and Synthesis of Marked Graphs," IEEE Trans. Circuits and Systems, Vol. CAS-24, July 1977, p. 400-405.

69. Murata, T., Koh, J.Y., "Reduction and Expansion of Live and Safe Marked Graphs," Proc. 1979 ISCAS, 1979, p. 841-844.

70. Murata, T., "Synthesis of Marked Graph Computation Models for Prescribed Resources and Performance," Proc. COMPSAC, 1979, p. 807-812.

71. Noe, J.D., Nutt, G.J., "Macro E-Nets for Representation of Parallel Systems," IEEE Trans. Computers, Vol. C-22(8), Aug. 1973, p.718-727.

72. Noe, J.D., Crowley, C.P., Anderson, T.L., "Design of an Interactive Graphical Net Editor," Proc. CIPS-ACM Pacific Regional Symp., 1974, p. 386-402.

73. Noe, J.D., "Hierarchical Modelling With Pro Nets," Proc. National Electronics Conf., Vol. 32, 1978, p. 155-160.

74. Pacas-Skewes, E., "A Design Methodology for Digital Systems Using Petri Nets," Ph.D. Thesis, University of Texas, Austin, 1979.

75. Patil, S.S., "Asynchronous Logic Array," Project MAC, TM-62, May 1975.

76. Patil, S.S., "Coordination of Asynchronous Events," MIT Project MAC, TR-62, June 1970.

77. Peterson, J.L., "Computation Sequence Sets," J. Comput. Syst. Sci., Vol. 13(1), 1976, p. 1-24.

78. Peterson, J.L., "Petri Nets," Computing Surveys, Vol. 9(3), Sept. 1977, p. 223-252.

79. Peterson, J.L., "A Note on Colored Petri Nets," Inf. Process. Lett., Vol. 11(1), 29 Aug. 1980, p. 40-44.

80. Ramamoorthy, C.V., Fox, T.F., Li, H.F., "Scheduling Parallel Processable Tasks for a Uniprocessor," IEEE Trans. Computers, Vol. C-25(5), May 1976, p. 485-495.

81. Ramamoorthy, C.V., Ho, G.S., "Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets," IEEE Trans. Software Engineering, Vol. SE-6(5), Sept. 1980, p. 440-449.

82. Ramachandani, C., "Analysis of Asynchronous Concurrent Systems By Petri Nets," MIT Project MAC, TR-120, Feb. 1974.

83. Schiffers, M., Wedde, H., "Analysing Program Solutions of Coordination Problems by CP-Nets," Mathematical Foundations of Computer Science, Springer-Verlag, (New York,1978), p. 462-473.

84. Shapiro, R.M., Saint, H., "A New Approach to Optimization of Sequencing Decisions," Automaatic Programming, Vol. 6(5), 1970, p. 257-288.

85. Shapiro, S.D., "A Stochastic Petri Net with Applications to Modelling Occupancy Times for Concurrent Task Systems," Network, Vol. 9, 1979, p. 375-379.

86. Sifakas, J., "Use of Petri Nets for Performance Evaluation," Proc. 3rd Int'l. Symp. Measuring, Modelling and Evaluating Computer Systems, 1977, p. 75-93.

87. Sifakas, J., "Realization of Fault Tolerent Systems by Coding Petri Nets," Proc. 8th Int'l. Symp. Fault Tolerent Computing, 1978, p. 205.

88. Sifakas, J., "Structural Properties of Petri Nets," Mathematical Foundations of Computer Science, Springer-Verlag, (New York , 1978), p. 474-483.

89. Silva, M., David, R., "Programmed Synthesis of Automatic Logic Described by Petri Nets: A Method of Implementation on a Microcomputer," RAIRO Autom/Syst. Anal. σ Control, Vol. 13(4), p. 369-393

90. Stark, P.H., "Free Petri Net Languages," Mathematical Foundations of Computer Science, Springer-Verlag, (New York, 1978), p. 506-515.

91. Szlanko, J., "Petri Nets for Proving Some Correctness Properties of Parallel Programs," Proc. IFAC/IFIP Workshop on Real Time Programming, 1977, p. 75-83.

92. Tani, K., Murata, T., "Scheduling Parallel Computations with Storage Constraints," Proc. 12th Asilomar Conf. Circuits, Systems and Computers, 1978, p. 736-743.

93. Tani, K., Onaga, K., Kakusho, O., "Modeling and Analysis of Resource-Constrained Network Problems by Petri Nets," Proc. Int'l. Conf. Cybernetics and Society, 1978, p. 884-888.

94. Valette, R., "Analysis of Petri Nets by Stepwise Refinements," J. Comput. Syst. Sci., Vol. 18, 1979, p. 35-46.

95. Valk, R., "On Computational Power of Extended Petri Nets," Mathematical Foundations of Computer Science, Springer-Verlag, (New York, 1978), p. 526-535.

96. Van Leuwen, J., "A Partial Solution to the Reachability Problem for Vector Addition Systems," Proc. 6th Symp. Theory of Computing, 1974, p. 303-304.

97. Vick, C.R., Kartashev, S.I., Kartashev, S.P., "Adaptable Architectures for Supersystems," Computer, Vol. 13(11), Nov. 1980, p. 17-35.

98. Voss, K., "Using Predicate/Transition Nets to Model and Analyse Distrbuted Database Systems," Proc. COMPSAC, 1979, p. 801-806.

99. Winter, R., "An Evaluation Net Model for the Performance Evaluation of a Computer Network," Proc. 3rd Int'l. Symp. Measuring, Modelling and Evaluating Computer Systems, 1979, p. 95-113.

100. Yoeli, M., Ginsberg, A., "Petri Net Languages and Their Applications," University of Waterloo, Faculty of Mathematics, Research Report CS-78-45, Nov. 1978.

101. Zervos, C.R., Irani, K.B., "Colored Petri Nets: Their Properties and Applications," RADC-TR-77-246, Aug. 1977.

102. Zuberek, W.M., "Timed Petri Nets and Preliminary Performance Evaluation," Proc. 7th Symp. Computer Architecture, 1980, p. 88-96.

Appendix A Definitions and Notation

Petri Net Notation

n is the number of places.

m is the number of transitions.

Scheduling Notation

n is the number of processors.

r is the number of tasks.

s is the number of resources.

Definition 1 [53]

If $P=\{A_1,A_2,...,A_n\}$ is a set symbols, then P* is defined recursively, as follows.

(i) Λ, the empty state, is in P*.

(ii) A_i is in P*, $1 \le i \le n$.

(iii) If x and y are in P*, so is x+y.

where + is a commutative, associative binary operator.

A general form $x=\sum_i x_i A_i$, where x_i is the number of occurrences of A_i in x.

Definition 2 [53]

A generalized Petri net, GPN.

GPN=<P,T>

$P=\{A_1,A_2,...,A_n\}$ is a set of places.

$T=\{t_1,t_2,...,t_m\}$ P* P* is a set of transitions

$t_j: \sum_i a_{ij} A_i \rightarrow \sum_i b_{ij} A_i$

Definition 3 [53]

A marked Petri net, MPN.

MPN=<P,T,m>

m is an element of P*

Definition 4 [53]

The characteristic matrix, Γ.

$(\Gamma_{ij}) = a_{ij} - b_{ij}$

Definition 5 [80]

A timed Petri net, TPN.

TPN=<GPN,Ω>

Ω: $T_i \rightarrow (T_i, \tau_i)$ a function mapping a transition onto a positive real number, τ.

Definition 6 [102]

An instantaneous descriptor, d .

$d_i = (m_i, r_i)$

$m_i \in$ P* is the marking.

$r_i \in T \times R$ is the remaining time.

Definition 7 [38]

A coloured Petri net, CPN.

CPN=<GPN,C,F>

C=(X,<): a set of colours X and a partial order, <.

F: $A \rightarrow X$ a function mapping each edge $\in$ A onto a colour, where A is the set of edges.

Definition 8

A state y is directly reachable from a state x, if there exists a transition, t, enabled in x which when fired results in the state x, $x = t \Rightarrow y$.

Definition 9

A state y is reachable from a state x through a sequence of transitions, $\sigma = \sigma_1 \sigma_2 \ldots \sigma_k$, if there exists a sequence of states, x_i, $1 \le i \le k$, such that,

$$x_i = \sigma_i \Rightarrow x_{i+1}$$

$$x_k = \sigma_k \Rightarrow y$$

$$x = \sigma_1 \Rightarrow x_1$$

Definition 10

A path in a directed graph is a sequence of edges such that the edges are connected head to tail.

Definition 11

A directed circuit is a path with one common node, the first and last node.

Definition 12 [16]

A task system $(\mathcal{T}, <, \{\tau_{ij}\}, \{\mathcal{R}\})$

$\mathcal{T} = (T_1, T_2, \ldots, T_r)$: a set of tasks

$<$: an irreflexive partial order on $\quad$.

τ_{ij} : execution time of task i on processor j.

$\mathcal{R} = [R_1(T_j), R_2(T_j), \ldots, R_s(T_j)]$: resource requirement of task j.

Definition 13 [40]

The total execution time, μ, of a task system,

$$\mu = \sum_i \tau_i \, .$$

Definition 14 [40]

A chain starting with task, T_i, is a set of tasks, $(T_i, \ldots, T_j, T_k, \ldots, T_l)$, such that, T_k is an immediate sucessor of T_j, $T_j < T_k$, for all tasks in the chain.

Definition 15 [40]

The height of a task system, h, is the length of the longest chain in the task system.

Definition 16

The level of a task, T_i, is the cardinality of the longest chain starting with task, T_i.

Definition 17 [40]

The total processing power of i processors, B_i,

$$B_i = \sum_{n=1}^{i} b_n.$$

Appendix B

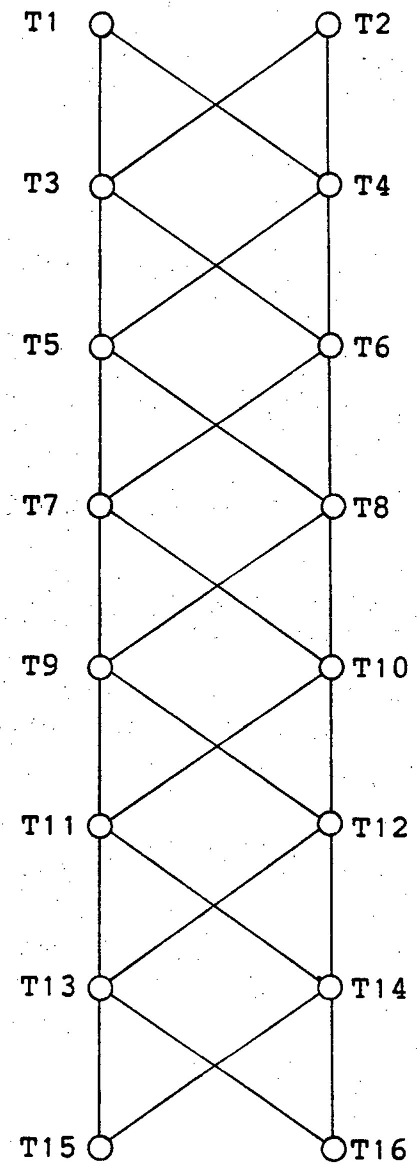


Figure B.1: Example 8 - <:most constrained

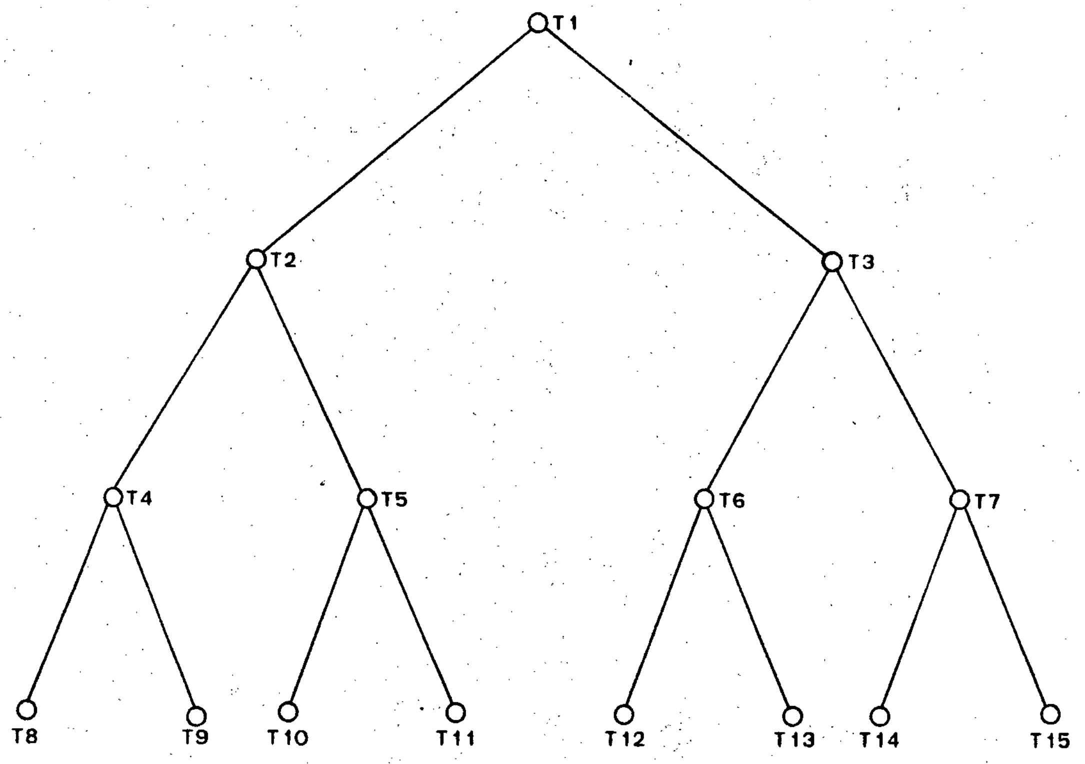


Figure B.2: Example 9 - <:least constrained

Processors	Execution Time w				
	MCLMT	HLF	CG	RANDOM	w*'
C_1 C_4 C_5	298	334	380	408	226
C_1 C_5 C_5	380	388	388	388	248
C_1 C_4	304	328	328	304	260
C_1 C_4 C_4	300	284	328	328	208
C_1 C_5	432	472	384	432	289
C_1 C_1	198	208	208	200	190
C_1 C_1 C_1	190	190	190	190	190

Table B.1: Example 1 with longer execution times and slower processors

Processors	Execution Time w				
	MCLMT	HLF	CG	RANDOM	w*'
C_1 C_1	68	68	70	70	68
C_1 C_5	98	152	110	120	89
C_1 C_6	104	178	102	138	91
C_1 C_6 C_6	142	178	176	172	80
C_1 C_5 C_5	90	170	138	148	76
C_1 C_5 C_6	130	152	180	162	78

Table B.2: Example 1 executed on slower processors

Processors	Execution Time w				
	MCLMT	HLF	CG	RANDOM	w*'
C_1 C_1	120	126	128	126	104
C_1 C_2	135	145	146	139	123
C_1 C_2 C_2	119	130	131	133	104
C_1 C_3	140	168	154	144	136
C_1 C_3 C_3	138	160	158	156	104
C_1 C_3 C_3 C_3	138	160	158	156	104
C_1 C_2 C_3	129	148	149	154	104
C_1 C_1	186	188	188	190	177
C_1 C_2	238	244	248	243	213
C_1 C_2 C_2	184	184	222	236	168
C_1 C_3	244	246	290	248	236
C_1 C_3 C_3	210	210	286	282	177
C_1 C_2 C_3	210	210	251	258	168

Table B.3: Example 1 with longer execution times

Processors	Execution Time w				
	MCLMT	HLF	CG	RANDOM	$w^{*\prime}$
C_1 C_2	12	12	12	12	11
C_1 C_6	24	24	24	24	18
C_1 C_3	15	15	15	15	14
C_1 C_6 C_3	24	24	24	24	13
C_1 C_6 C_6	24	24	24	24	15
C_1 C_3 C_3	13	13	13	14	11

Table B.4: Example 1 with unit execution times