Conception préliminaire des systèmes de production à l'aide des réseaux de Pétri

Vânio Murilo Savi

N° d'ordre : Année 1994

THESE

présentée à

UNIVERSITE DE METZ, FACULTE DES SCIENCES
L'UFR MATHEMATIQUE, INFORMATIQUE, MECANIQUE, AUTOMATIQUE

pour obtenir le titre de

DOCTEUR DE L'UNIVERSITE DE METZ

Spécialité :
SCIENCES DE L'INGENIEUR (Mention : AUTOMATIQUE)

par

Vânio Murilo SAVI

CONCEPTION PRELIMINAIRE DES SYSTEMES DE PRODUCTION A L'AIDE DES RESEAUX DE PETRI

Soutenance le 30 Mai 1994 devant le Jury :

MM.	J.B.	LASSERRE	Rapporteur
	C.	LAURENT	Examinateur
	G.	MOREL	Rapporteur
	J-M.	PROTH	Directeur de thèse
	F.	VERNADAT	Examinateurs
	X-L.	XIE	

Cette thèse a été préparée au sein du projet SAGEP de l'INRIA-LORRAINE
4, rue Marconi - 57070 Metz

Table de Matières

Chapitre I

Introduction générale

I.1 La conception préliminaire des systèmes de production

Des changements importants se sont produits dans les pays industrialisés depuis la seconde guerre mondiale. La production concerne désormais des produits de grande qualité, l'économie d'envergure a progressivement remplacé l'économie d'échelle, les demandes sont devenues imprévisibles, et les entreprises ont à faire face à une concurrence qui se développe à l'échelle mondiale.

Pour s'adapter à ces évolutions, les entreprises utilisent des systèmes très automatisés, flexibles, dont les temps de réglage sont faibles voir inexistants (temps masqués). Le coût de tels systèmes est élevé, tant pour ce qui concerne leur achat que leur fonctionnement, et leur champ d'activité est relativement restreint, ce qui réduit leur cycle de vie. Se lancer dans la production industrielle est devenu une entreprise à haut risque, qui supporte de plus en plus mal les erreurs de conception. C'est la raison pour laquelle les responsables essayent d'améliorer cette activité, et en particulier la conception préliminaire (encore appelée "étude papier").

La conception préliminaire comprend, entre autres, la spécification fonctionnelle, la modélisation et l'évaluation des systèmes. Elle doit aboutir à la définition de la configuration la mieux adaptée à la production prévue pour le système. Plusieurs outils peuvent être utilisés au niveau de la conception préliminaire. Les plus connus sont :

- **Les réseaux de files d'attente** ([Gross et Harris 74], [Buzzacott et Yao 86]), utilisés pour l'évaluation des systèmes en régime permanent. Cependant, s'agissant de systèmes de production, on peut obtenir des résultats très éloignés de la réalité. Ceci est dû aux hypothèses

faites dans la théorie des files d'attente (débits importants, disparition d'entités en cours de processus, hypothèses restrictives sur les lois d'arrivée des produits et sur la distribution des temps de fabrication, etc ...). Nous retiendrons donc que l'utilisation des files d'attente se limite à l'évaluation grossière des performances des systèmes de production.

- **Les approches de type états-transitions** sont également utilisées pour l'évaluation des performances. Ce type d'approche nécessite la définition de tous les états possibles, ainsi que des probabilités de passage entre tout couple d'états. Le problème se ramène ensuite à l'évaluation d'un processus markovien ou semi-markovien. Ce type d'approches est difficile d'utilisation pour l'évaluation des systèmes de production, du fait du nombre important d'états concernés.

- **Les approches de programmation** sont utilisées pour l'évaluation des performances en régime périodique. Elles se ramènent souvent à la résolution d'un programme de programmation linéaire en variables mixtes.

- **La simulation** utilise des *entités* caractérisées par leurs *attributs*. L'*état* du système est un ensemble de valeurs affectées aux attributs. Les autres composantes utilisées en simulation sont les *critères,* dont les valeurs sont fonctions des valeurs des attributs, les *contrôles admissibles* et les *processus* qui permettent de déduire les valeurs des critères du couple (état-contrôle), et les *contraintes* (par exemple les gammes de fabrication et les contraintes de capacité). Le modèle associé à la simulation est un programme qui ne saurait servir de spécification. De nombreux langages de simulation facilitent l'écriture des programmes de simulation. Citons entre autres WITNESS ([Murgiano 90], [Hollocks 91]), SIMAN [Pegden *et al.* 90] et SLAM II [O'Reilly et Ryan 92].

- **Les réseaux de Petri** est un outil graphique et mathématique qui permet la description et l'étude des systèmes de processus concurrents et parallèles évoluant dans le temps de façon discrète, déterministe ou stochastique. Le modèle de base des réseaux de Petri ([Peterson 81], [Murata 89]) correspond à un graphe biparti composé d'un ensemble de places et d'un ensemble de transitions reliées entre elles par des arcs. L'état du système est décrit par la présence d'un certain nombre de jetons dans les places : c'est le marquage du réseau. L'évolution du système se fait par le franchissement des transitions, lesquelles permettent de modéliser les actions du système (par exemple la réalisation d'une opération sur une machine).

Parmi les outils cités ci-dessus, les réseaux de Petri est le seul ensemble d'outils capable de supporter à la fois la spécification fonctionnelle, la modélisation et l'évaluation des systèmes

de production. C'est pour cela qui nous l'avons choisi comme support pour les études que nous avons menées et qui sont présentées dans la suite de ce mémoire.

I.2 Fonctionnement cyclique et non-cyclique

On peut classer les systèmes de production en deux groupes selon la façon dont la production est gérée : les systèmes de production à *fonctionnement cyclique* (répétitif) et les systèmes de production à *fonctionnement non-cyclique* (non-répétitif).

Les systèmes de production à fonctionnement cyclique se caractérisent par le fait que les ratios de production, c'est-à-dire les pourcentages de fabrication des différents produits, sont supposés connus et ne changent pas au cours d'une période suffisamment longue. En d'autres termes, dans ce type de systèmes, on cherche à optimiser un critère (la productivité, par exemple) tout en satisfaisant les ratios demandés. Le fonctionnement optimal d'un tel système est évidemment obtenu par un contrôle périodique.

Une conséquence importante du fait d'avoir un contrôle périodique est la possibilité de modéliser simultanément le système physique et le système de contrôle. Ce modèle est particulièrement utile quand on considère le problème de l'évaluation des performances du système.

Il a été montré qu'un système à fonctionnement cyclique peut être modélisé à l'aide d'une classe particulière de réseaux de Petri, appelés graphes d'événements. Les graphes d'événements possèdent des propriétés fortes qui ont permis l'établissement de résultats analytiques complets concernant leur comportement. Ces résultats rendent possible une analyse détaillée du comportement d'un système modélisé par cette classe de réseaux de Petri, ce qui a mis en évidence des propriétés fortes relatives au comportement de nombreux systèmes, comme par exemple les job-shops, les flow-shops, les systèmes d'assemblage, et les systèmes du type juste-à-temps (KANBAN) (voir en particulier [Hillion 89] et [Laftit 91]).

Les systèmes à fonctionnement non-cyclique doivent s'adapter aux aléas exogènes (variations de la demande dans le temps) et endogènes (modifications imprévues des capacités du fait de pannes). Le contrôle optimal de tels systèmes n'est donc plus cyclique. En conséquence, il ne peut plus être modélisé en même temps que le système physique. Les graphes d'événements n'ont donc plus aucune utilité pour la modélisation des systèmes de production non-cycliques qui, de ce fait, perdent beaucoup au niveau du potentiel d'analyse.

Contrairement au cas cyclique où l'évaluation est réalisée en fonction des résultats analytiques des graphes d'événements, l'évaluation d'un système à fonctionnement non-cyclique passe par la définition préalable d'une planification suivie d'un ordonnancement.

I.3 Positionnement du travail

Dans cette thèse nous participons à la conception préliminaire des systèmes de production à fonctionnement non-cyclique et nous proposons une méthodologie basée sur les réseaux de Petri pour la modélisation, l'analyse et l'évaluation de ce type de système. Contrairement au cas des systèmes cycliques, l'utilisation des réseaux de Petri dans l'étude des systèmes non-cycliques n'a jusqu'ici guère constitué une direction de recherche importante. Ce travail représente donc un premier effort dans le but d'améliorer cette situation. Nous nous plaçons dans le cas où les durées opératoires sont déterministes.

Les propriétés des réseaux de Petri constituent des atouts importants. Elles permettent d'étudier le système considéré de façon très précise dans ses propriétés structurelles et dynamiques. Dans le cas d'un système de production, ces propriétés permettent de dégager des renseignements importants sur l'évolution du système, comme par exemple l'identification des situations de blocage ou d'accroissement non contrôlé des en-cours, la possibilité ou non de revenir à l'état de départ à partir d'un état donné, etc ... Or, il n'existe pas de méthode satisfaisante de vérification des propriétés quand la taille du modèle réseaux de Petri est importante, comme c'est souvent le cas dans la modélisation d'un système de production. Pour résoudre ce problème, deux techniques peuvent être envisagées : soit réduire la taille du modèle afin que les méthodes d'analyse disponibles puissent être appliquées, soit adopter une approche modulaire qui garantisse que le modèle obtenu possède les propriétés désirées. On appelle cette dernière approche de synthèse du modèle.

La réduction de la taille du modèle ([Lee *et al.* 87], [Berthelot 85]) consiste à appliquer des règles de simplification qui permettent d'obtenir un modèle de taille réduite ayant certaines des propriétés du modèle original. Il est important de souligner que le modèle réduit n'est pas équivalent au modèle original, mais qu'il conserve quelques propriétés qualitatives de celui-ci qui dépendent des règles utilisées. Aucun ensemble de règles de réduction n'existe qui puisse garantir pour le modèle réduit la totalité des propriétés du modèle original. Une difficulté majeure dans l'application de cette méthode est l'identification des parties du réseau pour lesquelles les règles de réduction peuvent être utilisées.

Le but principal de l'approche de synthèse est proposer une méthodologie de construction progressive du modèle qui garantit la préservation des propriétés qualitatives souhaitées. Deux

approches de synthèse différentes ont été proposées dans la littérature [Jeng et DiCesari 90] : le raffinement (top-down en anglais) et l'intégration (bottom-up en anglais).

Dans le premier cas, on part d'un modèle initial très simple possédant toutes les propriétés souhaitées et on procède par itérations successives. A chaque étape on raffine le modèle par la substitution d'une place ou d'une transition par un sous-réseau, jusqu'à ce que l'on atteigne le niveau de détails souhaité. Cette substitution ne peut se faire que si le sous-réseau constitue ce qui a été appelé un *bloc bien formé*. Le point faible de cet approche est la définition du modèle initial, ce qui peut être difficile dans le cas des systèmes fortement couplés. Pour plus de détails le lecteur peut consulter [Valette 79], [Zhou *et al.* 89], [Brauer *et al.* 90] et [Suzuki et Murata 83].

Dans le deuxième cas, on considère que le système est composé d'un ensemble de sous-systèmes indépendants qui sont modélisés séparément. Aucune interaction entre les sous-systèmes n'est considérée à ce niveau. Le modèle global est alors obtenu par l'intégration progressive des sous-systèmes en fusionnant leurs places ([Agerwala et Choed-Amphai 78]), transitions ([Villarroel *et al.* 88]) ou chemins communs ([Krogh et Beck 86]). Après chaque étape, il faut vérifier les propriétés du système obtenu par l'intégration, car, et ceci est la difficulté majeure de cette approche, on ne peut pas garantir que celles-ci sont maintenues entre deux étapes successives.

Les travaux présentés dans cette thèse peuvent être placés dans un contexte plus général du développement d'une méthodologie du type "bottom-up" pour la modélisation des systèmes de production non-cycliques ([Proth et Savi 92], [Claver *et al.* 91], [Harlalakis *et al.* 92]). Cette méthodologie part du principe que le système est décomposé en un ensemble de modules (sous-systèmes), et que ces modules possèdent des propriétés qualitatives structurelles (vivacité, réversibilité et existence d'états d'accueil,...) et peuvent rester bornés grâce à un contrôle adéquat, autant de propriétés souhaitables dans le cas des systèmes de production. En outre, les modules considérés dans nos travaux sont tels que leur intégration conserve les propriétés ci-dessus. Nous avons montré que les hypothèses qui sont à la base de ces modules sont peu restrictives pour la modélisation des systèmes de production.

La figure I.1 illustre l'idée d'une telle méthodologie. L'intégration des modules est coordonnée par un système décisionnel qui a comme tâche principale le contrôle de l'évolution du système. Ce système décisionnel est constitué de deux parties : un système décisionnel local associé à chaque module, lequel intervient dans le module afin de que les séquences d'opérations définies au niveau de l'ordonnancement du module soient respectées, et un

système décisionnel global qui est en charge des décisions concernant la synchronisation du fonctionnement des différents modules.

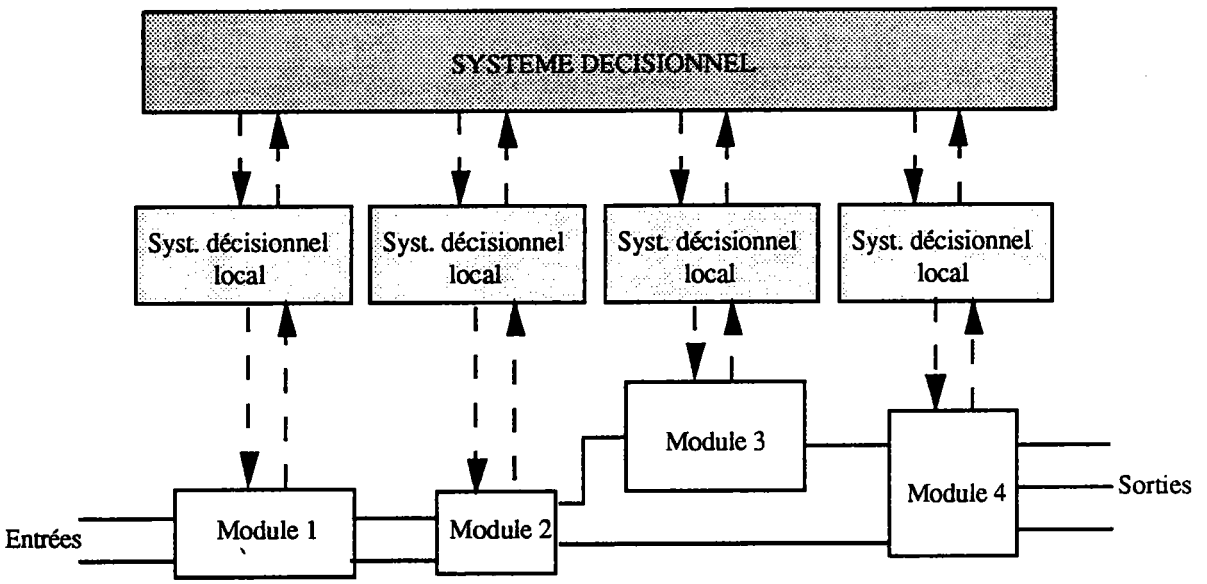


figure I.1 : La méthodologie de modélisation

Cette thèse est consacrée à l'étude de la gestion locale des modules. Ainsi, dans ce qui suit, nous allons utiliser le mot système pour désigner un module. Les études qui nous avons réalisées concernent donc la première partie de la méthodologie que nous venons d'exposer. Le problème de l'intégration est traité dans le cadre d'une autre thèse.

Ces études ont abouti au développement d'une méthode efficace et peu contraignante de modélisation et d'évaluation des systèmes de production à fonctionnement non-cyclique. Dans ce contexte, les contributions principales de la thèse sont les suivantes :

- la méthode développée permet de garantir certaines propriétés qualitatives du modèle (i.e., celles qui sont demandées du point de vue des systèmes de production), moyennant quelques contraintes faibles de réduction du niveau de la flexibilité du système ;

- le fait d'avoir utilisé les réseaux de Petri dans toutes les étapes du travail (modélisation, planification et ordonnancement), permet d'avoir un modèle cohérent qui réduit le possibilités d'erreurs qui peuvent survenir dans le cas où différents modèles sont employés au cours des étapes de conception. Ceci a l'avantage supplémentaire de faciliter la compréhension ;

- nous avons développé de nouveaux algorithmes pour la planification à court-terme et l'ordonnancement qui sont entièrement basés sur la structure des modèles physiques.

- les résultats obtenus ont été intégrés dans un logiciel itératif qui permet d'étudier complètement un système donné. Ce logiciel possède une interface agréable basée sur un système de fenêtrage, et une grande souplesse au niveau de l'utilisation.

I.4 Plan de la thèse

Ce mémoire est organisé en six chapitres. Dans le chapitre 2 nous rappelons les bases des réseaux de Petri. Nous insistons plus particulièrement sur tous les aspects qui sont intéressants du point de vue des systèmes de production. Nous dédions une partie de ce chapitre à une synthèse des principaux résultats publiés dans la littérature concernant les graphes d'événements. Ces résultats nous seront utiles lors de la présentation des systèmes à fonctionnement cyclique dans le chapitre 3. Finalement, dans la dernière partie de ce chapitre, nous introduisons une classe particulière de réseaux de Petri, les réseaux de Petri avec transitions d'entrée et de sortie (RPES), lesquels permettront de modéliser un système de production non-cyclique. Après avoir étudié les propriétés d'une sous-classe des RPES, les RPES sans conflit, nous montrons comment décomposer un RPES en un ensemble de RPES sans conflit, et nous définissons les conditions dans lesquelles l'utilisation de ces RPES sans conflit permettra de garantir les propriétés qualitatives souhaitées pour le système.

Nous abordons dans le chapitre 3 les systèmes à fonctionnement cyclique et nous montrons à l'aide de trois cas de figure, à savoir les job-shop, les systèmes d'assemblage et les systèmes du type juste-à-temps, les résultats obtenus dans le cadre de la modélisation et de l'évaluation des performances de ces systèmes

Le chapitre 4 est consacré à l'étude des systèmes à fonctionnement non-cyclique. Nous considérons d'abord le problème de la planification à court-terme et nous montrons que la solution de ce problème passe par la résolution d'un problème de programmation linéaire en nombres entiers ou d'un problème de programmation quadratique suivant le critère d'optimisation adopté. Nous abordons ensuite le problème de l'ordonnancement et nous présentons les deux algorithmes heuristiques qui ont été mis en place pour résoudre ce problème : le premier est basé sur la méthode du recuit simulé et le second est basé sur la procédure par séparation et évaluation.

Dans le chapitre 5, nous faisons d'abord une série d'études comparatives entre les algorithmes utilisés dans chacune des étapes de l'évaluation du modèle. Ensuite, nous appliquons la méthodologie développée sur une batterie d'exemples de tailles importantes, et nous discutons les résultats obtenus.

Nous concluons avec le dernier chapitre en discutant la portée et les limites des résultats présentés dans cette étude. Nous discutons et suggérons aussi quelques axes de recherche pour la suite de ce travail.

Chapitre II

Les réseaux de Petri

II.1 Introduction

L'utilisation des réseaux de Petri (**RdP**) pour la modélisation, l'analyse et l'évaluation de systèmes, et plus particulièrement de systèmes de production, n'a pas cessé de s'étendre depuis plusieurs années. Ceci est une conséquence directe de deux caractéristiques principales des RdP : d'abord, leur pouvoir de représentation, permettant la modélisation des activités parallèles, concurrentes et synchrones, et l'utilisation d'une représentation graphique simple qui est relativement proche du système modélisé, ensuite leur pouvoir d'analyse, caractérisé par l'existence d'un ensemble de propriétés qui rendent possible une évaluation très détaillée du fonctionnement des systèmes qu'ils modélisent.

Les RdP ont énormément évolué depuis leur création par C. A. Petri [Petri 62]. Plusieurs variantes du modèle original ont été développées en vue de satisfaire des critères spécifiques, comme par exemple l'augmentation du pouvoir de représentation, au prix d'une perte de puissance d'analyse. Ceci est la raison principale qui nous a amenés à choisir les réseaux de Petri élémentaires dans le cadre du travail que nous présentons dans ce mémoire.

Nous proposons dans les sections que suivent une introduction aux RdP et, en particulier, à une classe de RdP qui est à la base des résultats présentés. Nous nous contentons de présenter les aspects qui seront nécessaires à la compréhension des résultats que nous introduisons dans les chapitres suivants. Des descriptions plus détaillées des réseaux de Petri peuvent être trouvées dans [Murata 89],.[Peterson 81], [Brams 83] et [David et Alla 89]. Pour des applications des réseaux de Petri aux systèmes de production, les lecteurs peuvent se reporter à [Silva et Valette 90], [Dicesare *et al.* 93] ou [Valavanis 90].

Dans les sections II.2 à II.5 nous faisons un rappel des réseaux de Petri : la section II.2 présente les notions de base, la section II.3 introduit quelques classes particulières de RdP et les sections II.4 et II.5 sont destinées, respectivement, aux propriétés qualitatives et aux méthodes de vérification de ces propriétés. Dans la section II.6, nous introduisons les réseaux de Petri temporisés et présentons les deux possibilités classiques : associer la temporisation aux places ou aux transitions. La section II.7 est consacrée à la présentation d'une classe particulière de RdP, les graphes d'événements temporisés, lesquels possèdent des propriétés fortes et sont bien adaptés à l'étude des systèmes de production cycliques. Finalement, dans la section II.8, nous introduisons et formalisons une nouvelle classe de RdP, les RPES (Réseaux de Petri avec transitions d'entrée et transitions de sortie) lesquels forment la base de la méthodologie que nous avons développée pour l'étude des systèmes de production non-cycliques. Nous montrons comment décomposer un RdP en RPES sans conflit et nous présentons les conditions qui doivent être vérifiées pour qu'un RdP puisse être décomposé.

II.2 Notions de base

II.2.1 Définition d'un RdP

Un RdP est un graphe orienté constitué de deux types de noeuds: les places (représentées par des cercles) et les transitions (représentées par des barres). Un arc orienté relie toujours deux noeuds de natures différentes. Une transition (resp. place) peut être reliée à une ou plusieurs places (resp. transitions) en entrée ou/et en sortie. A chaque arc est affecté un entier positif qui est son poids (l'absence d'entier correspond à un poids de valeur unitaire). Un RdP où tous les arcs sont pondérés à 1 est appelé un *réseau de Petri ordinaire*.

L'état du système se traduit par la présence d'un nombre entier positif ou nul de jetons (représentés par des points) dans chaque place. Le *marquage* M d'un RdP est la donnée du nombre de jetons dans chaque place. M peut être représenté sous la forme d'un vecteur colonne de dimension égale au nombre de places et dont la $i^{ème}$ composante, $M(p_i)$, est le marquage de la place p_i.

La modification de l'état se fait par le franchissement (tir) de transitions. Une transition t est dite *franchissable* si chacune de ses places d'entrée contient un nombre de jetons supérieur ou égal au poids de l'arc joignant cette place à la transition. Le franchissement d'une transition modifie le marquage de la façon suivante :
a) on retire de chacune de ses places d'entrée un nombre de jetons égal au poids de l'arc joignant cette place à la transition ;

b) on ajoute à chacune de ses places de sortie un nombre de jetons égal au poids de l'arc joignant la transition à la place.

Prenons l'exemple du réseau de Petri présenté dans la figure II.1, lequel est composé de 5 places et 4 transitions. Le marquage M_0 observé est :

$$M_0 = [0,1,0,2,1]^t$$

car p_1 contient zéro jetons, p_2 contient un jeton, p_3 contient zéro jeton, p_4 contient deux jetons et p_5 contient un jeton. Tous les arcs sont pondérés à 1 sauf l'arc (t_1, p_3) pondéré à 3, l'arc (p_3, t_3) pondéré à 2 et l'arc (p_4, t_4) pondéré à 2. Dans ce cas, la seule transition franchissable est t_4 et son franchissement conduit au marquage M_1 présenté dans la figure II.2 ($M_1 = [1,1,0,0,0]^t$).

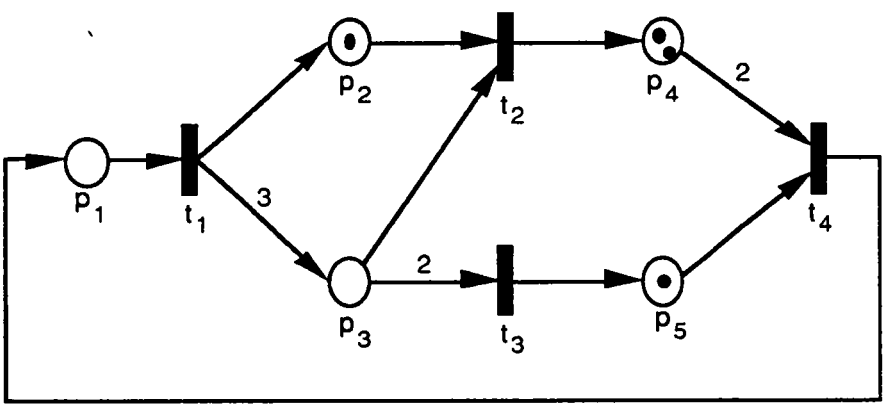


Figure II.1 : Un réseau de Petri marqué

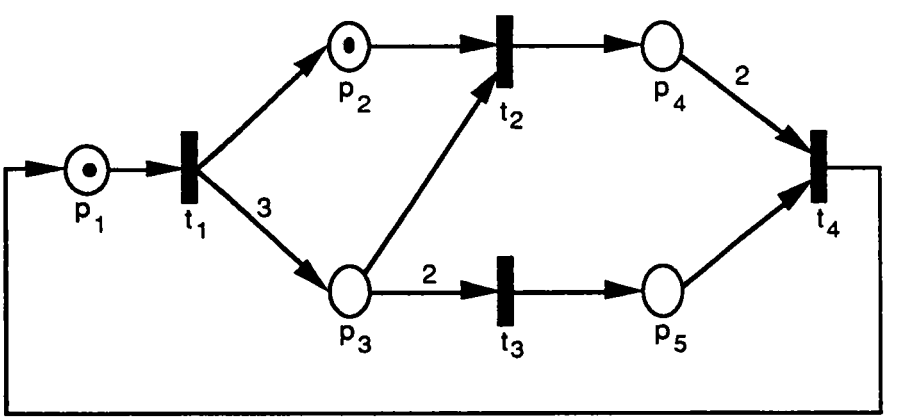


Figure II.2 : Marquage après le franchissement de la transition t_4

Formellement, un réseau de Petri marqué G est un quintuplet :

$$G = \langle P, T, Pré, Post, M_0 \rangle$$

où

- P est un ensemble fini de places et $|P| = n$;
- T est un ensemble fini de transitions et $|T| = m$;
- Pré:PxT $\rightarrow$ N est l'application d'incidence avant, correspondant aux arcs directs liant les places aux transitions. Pré(p,t) > 0 signifie qu'il existe un arc orienté de p vers t et Pré(p,t) est la valuation de cet arc. Pré(p,t) = 0 indique l'absence d'un arc orienté reliant p à t ;
- Post: PxT $\rightarrow$ N est l'application d'incidence arrière, correspondant aux arcs directs liant les transitions aux places. Post(p,t) > 0 signifie qu'il existe un arc orienté de t vers p et Post(p,t) est la valuation de cet arc. Post(p,t) = 0 indique l'absence d'un arc orienté reliant t à p ;
- M_0:P $\rightarrow$ N est le marquage initial.

Les applications Pré et Post peuvent être représentées par des matrices ayant autant de lignes (resp. colonnes) que le nombre de places (resp. transitions). On note Pré(.,t_j) et Post(.,t_j) les colonnes de Pré et Post associées à la transition t_j, respectivement. De même, on note Pré(p_i,.) et Post(p_i,.) les lignes associées à la place p_i.

Si Pré(p_i,t_j) > 0, nous dirons que la place $p_i \in P$ est une place d'entrée de la transition $t_j \in T$. (ou que la transition t_j est une transition de sortie de la place p_i). Si Post(p_i,t_j) > 0 nous dirons que la place p_i est une place de sortie de la transition t_j (ou que la transition t_j est une transition d'entrée de la place p_i). Nous notons :

°t : l'ensemble des places d'entrée de la transition $t \in T$;
t° : l'ensemble des places de sortie de la transition $t \in T$;
°p : l'ensemble des transitions d'entrée de la place $p \in P$;
p° : l'ensemble des transitions de sortie de la place $p \in P$.

Ainsi, la condition qu'une transition t doit vérifier pour qu'elle soit franchissable est :

$$M(p) \geq \text{Pré}(p,t) \qquad \forall p \in {}^\circ t.$$

Cette condition est notée par : M[t >. Le franchissement de cette transition provoque alors le passage à un nouveau marquage M' qui est calculé par l'expression suivante :

$$M'(p) = M(p) - \text{Pré}(p,t) + \text{Post}(p,t), \qquad \forall p \in P$$

On notera ceci M[t > M'.

II.2.2 Matrice d'incidence

La matrice d'incidence d'un RdP est une application $W:PxT \rightarrow Z$ dont les éléments sont définis par :

$$W(p,t) = Post(p,t) - Pré(p,t), \quad \forall p \in P \text{ et } \forall t \in T.$$

W(p,t) indique le changement apporté au marquage de la place p par le franchissement de la transition t.

Pour l'exemple de la figure II.1 les matrices Pré et Post sont les suivantes :

PRE

	t_1	t_2	t_3	t_4
p_1	1	0	0	0
p_2	0	1	0	0
p_3	0	1	2	0
p_4	0	0	0	2
p_5	0	0	0	1

POST

	t_1	t_2	t_3	t_4
p_1	0	0	0	1
p_2	1	0	0	0
p_3	1	0	0	0
p_4	0	1	0	0
p_5	0	0	1	0

et la matrice d'incidence est :

	t_1	t_2	t_3	t_4
p_1	-1	0	0	1
p_2	1	-1	0	0
p_3	1	-1	-2	0
p_4	0	1	0	-2
p_5	0	0	1	-1

II.2.3 Séquence de franchissements et Vecteur caractéristique

Si $M_0[t_1 > M_1, M_1[t_2> M_2,...,M_{n-1}[t_n> M_n$, on dira que $S = <t_1, t_2 ,...,t_n>$ est une *séquence franchissable* pour le marquage M_0 et que son franchissement conduit au marquage M_n. De la même façon on dira que le marquage M_n est accessible à partir du marquage M_0. On notera le franchissement de S par :

$M_0[S > M_n$

Un marquage M est dit accessible à partir de M_0 si et seulement s'il existe une séquence S franchissable à partir de M_0 dont le franchissement conduit au marquage M. Pour un RdP donné, l'ensemble de tous les marquages accessibles à partir d'un marquage initial M_0 est :

$$R(M_0) = \{M \mid \exists S,\ M_0[S > M\}.$$

Dans le cas du réseau de la figure II.1, étant donné le marquage initial $M_0 = [0,1,0,2,1]^t$, $S = \{t_4,t_1,t_2\}$ est une séquence franchissable qui conduit au marquage $M = [0,1,0,1,0]^t$.

Etant donnée une séquence de franchissements S, le *vecteur caractéristique* (ou vecteur de comptage) de S, noté $\underline{S}$, est un vecteur de dimension m dont la $i^{ème}$ composante correspond au nombre de fois que la transition t_i apparaît dans la séquence S. Prenons l'exemple ci-dessus, le vecteur caractéristique associé la séquence $S = \{t_4,t_1,t_2\}$ est $\underline{S} = [1,1,0,1]$.

Un vecteur $\underline{S}$ est dit vecteur caractéristique réalisable s'il lui correspond au moins un séquence de franchissements S à partir d'un marquage M. Un vecteur $\underline{S}$ quelconque n'est pas toujours réalisable. De même, à un vecteur $\underline{S}$ réalisable peut correspondre à différentes séquences de franchissements.

II.2.4 Equation fondamentale

Soit S une séquence de franchissements telle que M[S > M'. Alors, le nouveau marquage M' obtenu par le franchissement de S peut être calculé par l'expression suivante :

$$M' = M + W \cdot \underline{S}$$

Ainsi, avec le franchissement de la séquence $S = \{t_4,t_1,t_2\}$ le réseau de la figure II.1 atteindra le marquage :

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

II.2.5 L'arbre de recouvrement

L'ensemble des marquages accessibles à partir du marquage initial M_0 peut être représenté sous la forme d'un arbre, que nous appelons *arbre des marquages atteignables*. Dans cet arbre, les nœuds représentent les marquages et les arcs les transitions franchies. En partant de M_0, nous représentons tous les nouveaux marquages obtenus par le franchissement de chacune des transitions validées par M_0. On répète cette procédure à partir de chacun de ces marquages pour atteindre des nouveaux marquages, et ainsi de suite. La figure II.3 donne un exemple d'un RdP marqué et de son arbre des marquages atteignables.

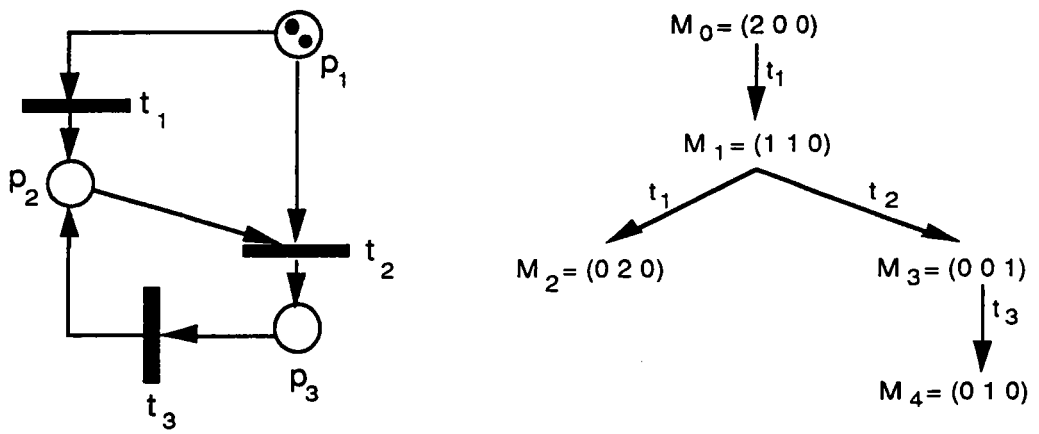


Figure II.3 : Un exemple d'arbre des marquages atteignables

Dans l'exemple ci-dessus, l'arbre obtenu est fini puisque le nombre de marquages atteignables est limité. Mais ce n'est pas toujours le cas. Un réseau avec un nombre infini de marquages atteignables aura un arbre infini. Même dans le cas où le nombre de marquages atteignables est limité l'arbre obtenu peut être infini. Pour résoudre ce problème nous allons construire un autre arbre, nommé *arbre de recouvrement*, lequel sera toujours fini, par construction. L'algorithme pour l'obtention d'un arbre de recouvrement est présenté ci-dessous. Dans cet algorithme le symbole "ω" est utilisé pour représenter un marquage non borné.

Algorithme II.1:

1. Poser le marquage initial M_0 comme la racine de l'arbre.
2. Tant que des marquages nouveaux existent faire :
   2.1. Choisir un nouveau marquage M.
   2.2. Si M a été déjà trouvé sur le chemin conduisant de la racine M_0 jusqu'à M, marquer M par "déjà trouvé". On ne poursuivra pas les recherches à partir de ce nœud.
   2.3. Si aucune transition ne peut être franchie à partir de M, marquer M par

"blocage".

2.4. Pour toute transition t franchissables à partir de M faire :

2.4.1. Calculer le marquage M_t obtenu par franchissement de t.

2.4.2. Si, sur le chemin reliant M_0 à M_t, il existe un marquage M* tel que $M_t(p) \geq M^*(p)$ quelle que soit la place p et si $M_t(p) > M^*(p)$ pour au moins une place p, alors marquer par "ω" tous les éléments de M_t correspondant aux places p telles que $M_t(p) > M^*(p)$.

2.4.3. Conserver M_t comme nouveau sommet de l'arbre (i.e. nouveau marquage) et établir l'arc (M,M_t) marqué t.

Un exemple d'application de cet algorithme est présenté dans la figure II.4.

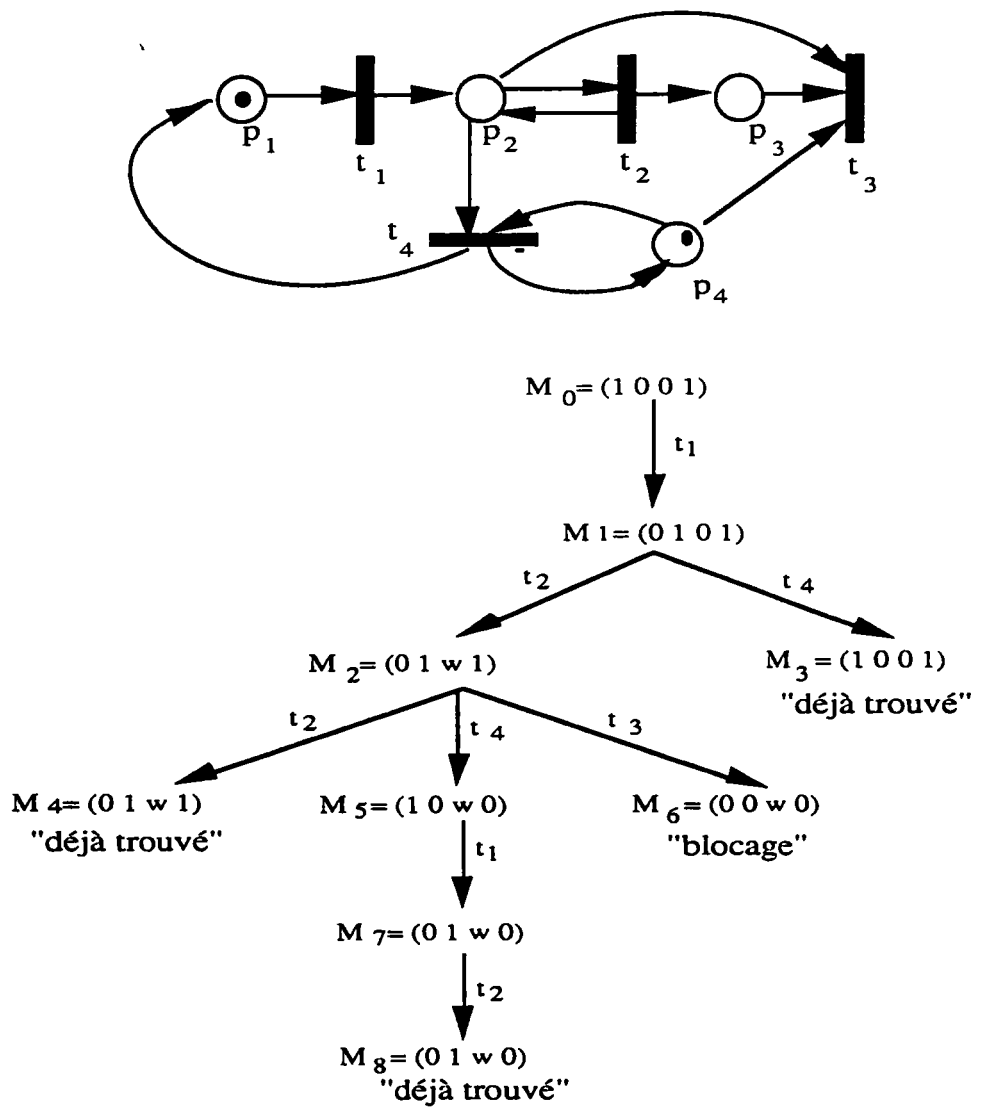


Figure II.4 : Un exemple d'arbre de recouvrement

Remarque :

L'utilisation du symbole "ω" dans cet algorithme provoque une perte d'information du fait que l'arbre ne contient pas tous les marquages qu'il est possible d'atteindre à partir de M_0. Une conséquence directe de ceci est que deux RdP différents peuvent avoir le même arbre de recouvrement (voir l'exemple présenté dans [Peterson 81]).

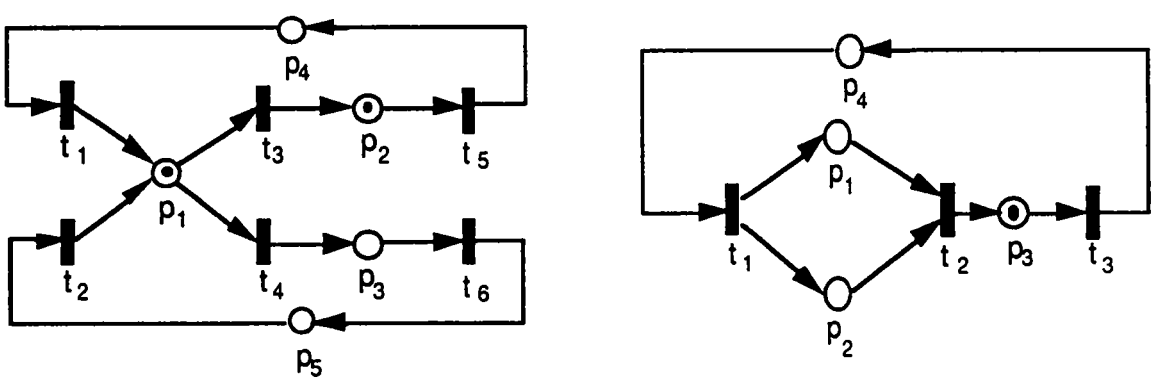
L'arbre de recouvrement est l'outil de base pour la vérification des propriétés d'un RdP qui sont définies dans la section II.4. On y reviendra dans la section II.5.

II.3 Sous-classes des Réseaux de Petri

Cette section est consacrée à une description succincte de quelques réseaux de Petri à structure particulière. D'autres sous-classes et des extensions des RdP, peuvent être trouvées dans la bibliographie fournie au début de ce chapitre.

II.3.1 Machine à états et graphes d'événements

Une *machine à états* est un RdP dans lequel chaque transition a exactement une place d'entrée et une place de sortie. Un *graphe d'événements* correspond au dual d'une machine à états : c'est un RdP dans lequel chaque place a exactement une transition d'entrée et une transition de sortie. Un exemple de chacune de ces deux sous-classes de RdP est présenté dans la figure II.5 ci-dessous.



(a) : machine à états (b) : graphe d'événements

Figure II.5 : Exemples d'une machine à états et d'un graphe d'événements

Comme on peut le constater dans les exemples de la figure ci-dessus, un réseau du type machine à états permet la représentation de décisions (voir place p_1 dans la figure II.5.a) mais ne permet pas la représentation de la synchronisation des activités parallèles. Un graphe

d'événements permet par contre de représenter la concurrence (voir place t_2 dans la figure II.5.b) mais non les décisions.

II.3.2 RdP sans conflit

Une place p ayant au moins deux transitions de sortie constitue ce qu'on appelle un *conflit structurel* (aussi appelé décision ou choix). On représentera ceci par un doublet formé d'une place et de l'ensemble de ses transitions de sortie $<p_1,\{t_1, t_2, ...\}>$.

Un conflit représente la possibilité d'une exclusion mutuelle entre différents événements du système. Un conflit structurel devient un *conflit effectif* quand le marquage est tel que le nombre de marques dans la place p est inférieur à la somme des marquages des arcs conduisant aux transitions de sortie de cette place. Ceci implique donc la nécessité d'un choix des transitions à franchir. Dans la figure II.5.a le doublet $<p_1,\{t_3,t_4\}>$ est un conflit effectif.

Un *RdP sans conflit*, comme l'indique son nom, est un RdP dans lequel il n'y a pas de conflit structurel.

II.3.3 RdP pur

Une circuit élémentaire constitué d'une seule transition et d'une seule place est appelée une *boucle élémentaire* ou plus simplement boucle. Une telle boucle se caractérise par le fait que la transition possède une place d'entrée qui est également une place de sortie de cette transition.

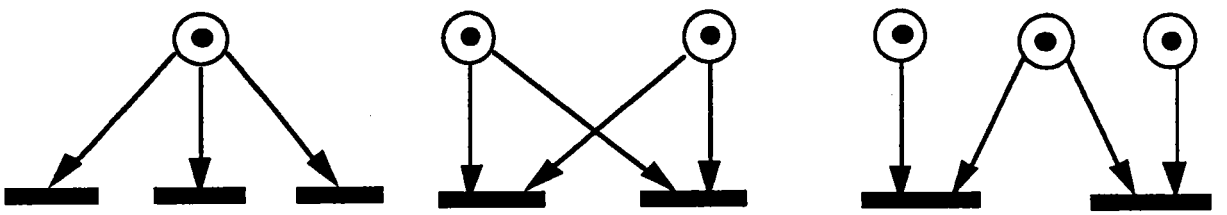
Un RdP est *pur* s'il ne contient pas de boucle élémentaire

Une propriété intéressante des RdP est qu'il est toujours possible de transformer un réseau impur en un réseau pur en ajoutant dans chaque boucle élémentaire une transition et une place.

II.3.4 RdP à choix libre

On dira qu'un RdP est *à choix libre* si pour tout conflit $<p_1,\{t_1, t_2, ...\}>$, p_1 est la seule place d'entrée de l'ensemble de transitions $\{t_1, t_2, ...\}$. Il s'agit donc d'un cas où la présence d'une marque dans la place p_1 autorise le franchissement de toutes ses transitions de sortie (le choix de la transition à franchir reste libre). Cette définition peut être généralisée à la classe des réseaux à choix libre étendu de la façon suivante : un RdP est *à choix libre étendu* si pour tout conflit $<p_1,\{t_1, t_2, ...\}>$ toutes les transitions $\{t_1, t_2, ...\}$ ont le même ensemble de places

d'entrée. Les différentes structures qui caractérisent un réseau à choix libre, à choix libre étendu, et à choix non libre sont illustrées dans la figure II.6.



(a) : Choix libre (b): Choix libre étendu (c) : Choix non libre

Figure II.6 : Réseau à choix libre, réseau à choix libre étendu et réseau à choix non libre

II.4 Propriétés qualitatives

II.4.1 Vivacité

Un RdP G est dit *vivant* pour un marquage initial M_0 si, pour tout marquage accessible $M \in R(M_0)$, il est possible de trouver une séquence de franchissements S qui permet de franchir n'importe quelle transition de G en partant de M. Un blocage correspond à un marquage où aucune transition n'est validée. La propriété de vivacité assure donc le non-blocage.

II.4.2 Réseau Borné

Un RdP est *k-borné* pour un marquage initial M_0 si, quel soit le marquage $M \in R(M_0)$ et quelle que soit la place $p \in P$, $M(p) \leq k < \infty$, où k est un entier naturel. Un RdP est dit *sauf* s'il est 1-borné.

Du point de vue des systèmes de production, cette propriété garantit qu'il n'y aura pas d'accumulation d'en-cours dans le système.

II.4.3 Consistance et réversibilité

Un RdP est dit *consistant* s'il existe un marquage initial M_0 et une séquence de franchissements S contenant au moins une fois chaque transition, tel que $M_0[S > M_0$.

Un RdP est *réversible* pour un marquage initial M_0 si, quel que soit le marquage accessible $M \in R(M_0)$, il existe une séquence de franchissements S tel que $M[S > M_0$. En

d'autres termes, dans un RdP réversible, il est toujours possible de revenir au marquage initial.

II.4.4 Persistance

Un RdP est *persistant* pour un marquage initial M_0 si, quel que soit le marquage accessible $M \in R(M_0)$ et quel que soit le couple de transitions validées pour ce marquage, le franchissement d'une des deux transitions n'empêche pas le franchissement de l'autre. Un RdP persistant ne nécessite pas que soient prises des décisions pour la résolution des conflits, car l'ordre de franchissement ne conduira pas à annuler une possibilité de franchissement. Pour cette raison, un RdP persistant est aussi appelé un RdP à *décision libre*. Un RdP sans conflit est toujours persistant.

II.4.6 Invariants

Les invariants permettent de caractériser le réseau vis-à-vis de certaines propriétés des marquages accessibles et des transitions franchissables. Les notions d'invariants sont importantes car ils permettent d'identifier les parties du réseau pour lesquelles il y a une conservation du nombre de jetons ou pour lesquelles les séquences de franchissements des transitions conduisent à nouveau au marquage de départ.

II.4.6.1 p-invariants

Soit $B = [b_1, b_2, \ldots b_n]^t$, $n = |P|$, $b_i \geq 0$, un vecteur de pondération des places P d'un RdP G. B est un *p-invariant* si et seulement si:

$$B^t W = 0$$

où W est la matrice d'incidence.

Le vecteur B est aussi appelé un P-semi-flot. L'ensemble de places dont le poids est non nul dans B, noté P(B), est appelé support de B. En effet, si on multiplie l'équation fondamentale $M = M_0 + W \cdot \underline{S}$ par B^t à gauche, on obtient :

$$B^t \cdot M = B^t \cdot M_0 + B^t W \cdot \underline{S}$$

Mais comme, par définition, $B^t \cdot W = 0$, on en déduit que $B^t \cdot M = B^t \cdot M_0$, quel que soit le marquage accessible $M \in R(M_0)$. Donc, si B est un p-invariant, ceci implique que la somme

des marquages de l'ensemble de places P(B), pondéré par les composantes du vecteur B, est constant.

II.4.6.2 t-invariants

Soit S une séquence de franchissements à partir du marquage initial M_0, telle que :

$W \underline{S} = 0$

où :

$\underline{S}$ est le vecteur caractéristique de S.

W est la matrice d'incidence.

Le vecteur $\underline{S}$ est appelé un *t-invariant* ou un T-semi-flot. Nous désignons T(S) le support de S, c'est-à-dire l'ensemble des transitions qui apparaissent dans la séquence S. La séquence S est une séquence répétitive puisque $M_0[S > M_0$, d'après l'équation fondamentale.

Il est important de souligner que le calcul algébrique de la solution de $W X = 0$ ne dépend pas du marquage. Ainsi, on peut obtenir des solutions X qui ne correspondent pas nécessairement à un t-invariant, car aucune séquence de franchissements ne possède un vecteur caractéristique $\underline{S}$ telle que $\underline{S} = X$.

II.4.6.3 Invariants minimaux

Notons Q l'ensemble des invariants (p-invariants ou t-invariants) associés à un RdP et notons J l'ensemble de leurs supports. Un invariant $B \in Q$ est dit minimal si et seulement si il n'existe pas d'autre invariant $F \in Q$ tel que $F \leq B$. Un support $S \in J$ est dit minimal si et seulement si il ne contient pas d'autres supports que $\emptyset$ et lui même. Il est important de noter que le support d'un invariant minimal n'est pas forcement un support minimal. Pour illustrer ceci prenons le réseau présenté dans la figure II.7 [Murata 89].

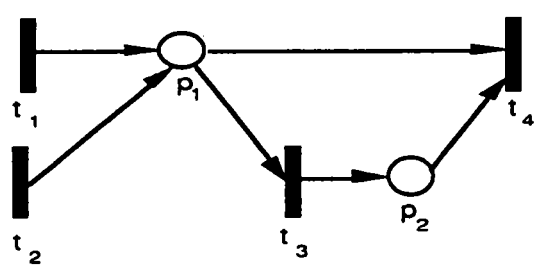


Figure II.7 : Réseaux avec 3 t-invariants minimaux et 2 t-invariants à support minimaux

Ce réseau possède trois t-invariants minimaux : $b_1 = (1\ 1\ 1\ 1)^t$, $b_2 = (2\ 0\ 1\ 1)^t$ et $b_3 = (0\ 2\ 1\ 1)^t$. Mais seulement b_2 et b_3 sont à support minimaux car le support de b_1, $\{t_1, t_2, t_3, t_4\}$ contient le support de b2, $\{t_1, t_3, t_4\}$. Dans ce cas b_1 peut être exprimé comme une combinaison linéaire de b_2 et b_3 ($b_1 = (b_2+b_3)/2$).

Les invariants minimaux dont les supports sont minimaux sont appelés invariants à supports minimaux. Tous les invariants d'un RdP peuvent être obtenus comme une combinaison linéaire des invariants à support minimaux [Memmi et Roucairol 80]

II.5 Méthodes d'analyse

Dans cette section nous faisons quelques commentaires à propos des méthodes disponibles pour la vérification des propriétés d'un RdP. Pour une description plus détaillée, le lecteur pourra consulter les références fournies au début de ce chapitre ou celles présentées tout au long de cette section.

a) Arbre de recouvrement

L'arbre de recouvrement est la technique d'analyse des propriétés d'un RdP la plus utilisée. L'inspection directe de l'arbre de recouvrement permet de mettre en évidence certaines propriétés d'un RdP. Si on examine un arbre construit selon l'algorithme présenté dans la section II.2.5, on peut dire, entre autres, que :

a) le RdP est sauf si et seulement si tous les marquages ne contiennent que des 0 et des 1.

b) Une transition t est morte (i.e. jamais franchie) si aucun des arcs de l'arbre de recouvrement n'est marqué t.

c) Le RdP est borné si et seulement si le symbole "ω" n'apparaît dans aucun des marquages correspondant à un nœud de l'arbre.

L'inconvénient principal de cette technique réside dans le fait qu'elle est une méthode de recherche exhaustive et, par conséquent, limitée à des réseaux de petite taille.

b) Algèbre linéaire

Comme nous l'avons souligné dans les paragraphes précédents, un réseau de Petri peut être représenté sous forme matricielle. Cette écriture permet d'utiliser des résultats de l'algèbre linéaire pour obtenir certains résultats sur les réseaux de Petri. Par exemple, pour vérifier si un réseau est vivant, consistant et borné nous utilisons les relations données dans la suite. Dans ce

relations C est la matrice d'incidence, m est le nombre de transitions et n est le nombre de places du réseau. Nous considérons aussi qu'un vecteur x est supérieur à zéro si et seulement si toutes ses composantes sont supérieures à zéro.

- vivacité : $\exists\, x \in R^m, x > 0$, tel que $C \cdot x \geq 0$
- consistance : $\exists\, x \in R^m, x > 0$, tel que $C \cdot x = 0$
- réseau borné : $\exists\, y \in R^n, y > 0$, tel que $y^{tr} \cdot C \leq 0$

Cependant, dans le cas général deux problèmes majeurs se posent pour la résolution des équations associées à une certaine propriété : i) le fait que l'on est contraint de trouver des solutions entières non nulles, et ii) la possibilité d'obtenir des solutions non significatives pour la propriété recherchée, car certaines relations donnent des conditions qui sont des conditions nécessaires mais non suffisantes.

Pour plus de détails concernant l'utilisation de l'algèbre linéaire dans la vérification des propriétés d'un RdP le lecteur peut consulter [Memmi et Roucairol 80], [Lautenbach 86], [Alaiwan et Toudic 85] et [Colom et Silva 91]

c) Réduction du réseau

La réduction ne constitue pas une méthode de vérification des propriétés, mais une technique qui fournit un modèle de taille réduite, lequel préserve certaines propriétés qualitatives du réseau original. Ceci rend l'analyse par les méthodes précédentes plus facile.

Berthelot [Berthelot 83] propose six simplifications élémentaires qui ne modifient pas les propriétés de vivacité, ni le fait que le réseau soit borné ou sauf. [David et Alla 89] présentent des règles de réduction qui préservent les invariants.

II.6 Réseaux de Petri temporisés

Dans les RdP que nous avons présentés jusqu'ici nous considérons que tous les franchissements sont exécutés immédiatement. Aucune durée n'est associée à un franchissement. Cependant le fonctionnement de certains systèmes dépend du temps. Par exemple, dans le cas d'un système de production, le modèle doit refléter la durée des opérations, des transports, des changements d''outils, etc. Pour pouvoir modéliser de tels systèmes nous allons utiliser les RdP temporisés. Deux types de modèles existent : celui dans lequel les temporisations sont associées aux places, et celui dans lequel les temporisations sont

associées aux transitions.

II.6.1 Temporisations associées aux places

Dans ce type de modèles une temporisation θ, éventuellement nulle, est associée à chaque place P. Dès qu'une marque arrive dans la place P, elle reste indisponible pour un temps θ, à la fin duquel elle devient disponible et peut être utilisée dans le franchissement des transitions de sortie de cette place.

A un instant τ quelconque le marquage M d'un RdP avec temporisation associée aux places est l'union du marquage M^d, correspondent aux marques disponibles, et du marquage M^i, correspondent aux marques indisponibles. Une transition est validée pour un marquage $M=M^dUM^i$ si elle est validée pour le marquage M^d. Comme dans le cas des RdP non-temporisés, le franchissement d'une transition t a une durée nulle. Cependant, les conséquences du franchissement de t sont un peu différentes dans le cas des RdP temporisés : celui-ci provoque l'enlèvement des marques disponibles de chacune des places d'entrée de t et le dépôt des marques indisponibles dans chacune des places de sortie de t. Ces dernières deviendront disponibles après l'écoulement du temps correspondant à la temporisation de chaque place de sortie et pourront, à ce moment là, être utilisées dans la validation des transitions.

II.6.2 Temporisation associée aux transitions

Une autre forme de temporisation consiste en associer une durée positive ou nulle pour le franchissement de chaque transition. On peut montrer que ce modèle est équivalent au modèle précédent, c'est-à-dire qu'il est possible de passer de l'un à l'autre.

Si θ est la temporisation liée à une transition validée t et τ_0 l'instant où son franchissement débute. Alors le franchissement de t se terminera à l'instant $\tau = \tau_0 + \theta$. Les instants τ_0 et τ sont appelés, respectivement, début de franchissement de t et fin de franchissement de t. La seule action réalisée au début de franchissement d'une transition t consiste à réserver les marques nécessaires à son franchissement. La transition t est effectivement franchie quand on atteint l'instant de fin de franchissement, et ceci se traduit par l'enlèvement des marques réservées dans les places d'entrée et le dépôt de marques non réservées dans les places de sortie de t.

Comme précédemment, on dira que le marquage M est composé de marques réservées et de marques non réservées. Une transition est validée pour un marquage M si et seulement si elle est validée pour le marquage correspondant aux marques non réservées.

II.7 Graphes d'événements

Dans le cadre de l'étude des systèmes de production à fonctionnement cyclique que nous allons présenter dans le chapitre III, nous introduisons les principaux résultats concernant les graphes d'événements (GE) temporisés. Ces résultats montrent que les GE possèdent des propriétés fortes qui facilitent l'analyse de leur comportement et qui, par conséquent, sont très utiles pour l'évaluation des systèmes qu'ils modélisent.

Nous considérons les GE dans lequel les temporisations, déterministes ou stochastiques, sont associées aux transitions. *Une autre hypothèse concerne la non-préemption* : aucun franchissement d'une transition ne peut commencer avant la fin du franchissement précédent. En tenant en compte ces deux hypothèses, nous présentons d'abord quelques définitions et résultats qui seront nécessaires par la suite. Ensuite nous considérerons les GE déterministes, puis les GE stochastiques.

II.7.1 Définitions et résultats préliminaires

Un *circuit élémentaire* est un chemin direct ayant comme arrivée le sommet (place ou transition) de départ, et dans lequel chaque sommet n'apparaît qu'une seule fois dans le chemin. Un graphe d'événements est dit *fortement connexe* si deux sommets quelconques sont reliés par un chemin direct. Le graphe d'événements fortement connexe présenté dans la figure II.8 possède deux circuits élémentaires :

$\gamma_1 = \langle p_1, t_2, p_3, t_3, p_4, t_4, p_2, t_1 \rangle$
$\gamma_2 = \langle p_5, t_2, p_3, t_3, p_4, t_4, p_6, t_5 \rangle$

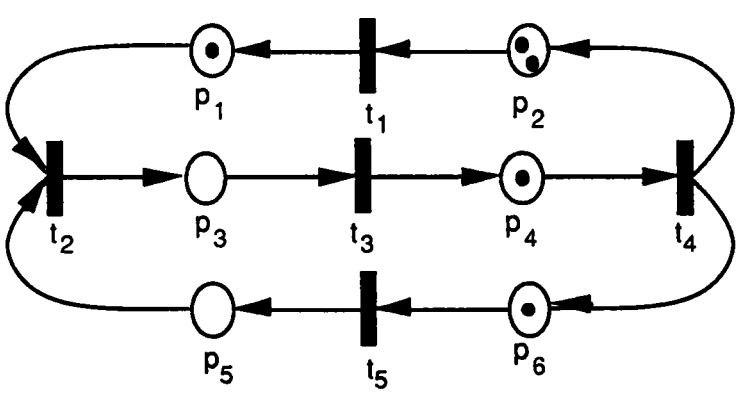


Figure II.8 : Un graphe d'événements fortement connexe

Le deux résultats présentés ci-dessous sont très importants, puisqu'ils permettent de vérifier si un graphe d'événements est vivant et borné. Ces résultats sont dûs à [Commoner *et al.* 71]

Résultat II.1

Dans un graphe d'événements, le nombre de jetons dans tout circuit élémentaire est invariant quelle que soit la séquence de franchissement des transitions.

Résultat II.2

Un graphe d'événements est vivant si et seulement si tout circuit élémentaire contient au moins un jeton.

D'après le résultat II.1, il est évident qu'il existe un p-invariant associé à chaque circuit élémentaire. En plus, on peut montrer que si on associe une pondération égale à 1 à chaque place dans un circuit élémentaire et une pondération égale à 0 aux autres places, alors le vecteur de pondération résultant correspond à un p-invariant minimal [Hillion 86]. Un GE fortement connexe est borné pourvu que son marquage initial soit borné. Ceci est une conséquence directe du résultat II.1 et du fait que dans un GE fortement connexe toute place appartient au moins à un circuit élémentaire.

On peut aussi montrer qu'à un GE fortement connexe possède un t-invariant dont toutes les composantes sont égales à 1. En d'autres termes, ceci signifie que si, à partir d'un marquage initial, chaque transition est franchie exactement une fois, on revient au marquage initial.

II.7.2 Graphes d'événements déterministes

Les résultats qui sont présentés dans cette section considèrent que nous sommes dans le contexte des GE fortement connexes dont chaque transition t_i est temporisée à θ_i. Soit Γ l'ensemble de tous les circuits élémentaires du GE et $\gamma \in \Gamma$ un circuit élémentaire quelconque. Une définition très importante est celle du *temps de cycle* d'un circuit élémentaire. Pour un circuit élémentaire γ, le temps de cycle est :

$$C(\gamma) = \mu(\gamma) / M(\gamma)$$

où $\mu(\gamma)$ est la somme des temps de franchissement des transitions du circuit élémentaire γ et $M(\gamma)$ est le nombre total de jetons présents dans les places appartenant à ce circuit. Notons $C^* = \max_{\gamma \in \Gamma} C(\gamma)$ le plus grand temps de cycle. Tout circuit élémentaire tel que $C(\gamma) = C^*$ est

appelé *circuit critique*. Il est important de noter que ce sont ces circuits qui réglent la vitesse de fonctionnement du réseau, i.e, le temps de cycle du réseau est C* [Ramamoorthy et Ho 80].

En ce qui concerne la vitesse de fonctionnement, les études des GE temporisés se sont portées sur deux modes de franchissement des transitions : le *fonctionnement au plus tôt* (**FTP**) dans lequel les transitions sont franchies dès qu'elles sont franchissables, et le *fonctionnement périodique* (**FPE**) qui correspond au cas où les transitions sont franchies de façon périodique.

Dans le cas de fonctionnement au plus tôt, Chretienne [Chretienne 83] a montré qu'un régime k-périodique est atteint après un nombre fini de franchissements des transitions. Le terme k-périodique est dû au fait que l'état du graphe redevient identique à lui-même après k temps de cycle C*. Ceci est donné par le résultat suivant :

Résultat II.3

Le fonctionnement au plus tôt d'un GE temporisé est k-périodique avec une fréquence moyenne de franchissement des transitions égale à 1/C. Autrement dit, il existe n_0 entier positif tel que :*

$$S_t(n+k) = S_t(n) + K \cdot C^*, \qquad \forall n \geq n_0, \quad \forall t \in T$$

où $S_t(i)$ désigne la date de début du i-me franchissement de la transition t.

Les remarques suivantes concernent le fonctionnement au plus tôt. Si tous les circuits critiques contiennent un seul jeton, alors le fonctionnement est simplement périodique (k=1). Lorsqu'il y a un seul circuit critique, le facteur de périodicité k est égal au nombre de jetons dans ce circuit. Enfin, le fonctionnement au plus tôt garantit les performances maximales du système.

Dans le cas de fonctionnement périodique, [Ramchandani 74] et [Ramamoorthy et Ho 80] ont montré qu'il est toujours possible de réaliser un régime de fonctionnement périodique assurant les mêmes performances que celle du fonctionnement au plus tôt, i.e., les performances maximales. D'après leurs résultats, la condition nécessaire et suffisante pour qu'on puisse atteindre un régime périodique de période π est que :

$$\pi \geq C^*.$$

Le mode de fonctionnement périodique est complètement défini par la donnée d'une période π et du temps de début du premier franchissement $S_t(1)$ de la transition t pour tout $t \in T$. Nous avons alors la relation suivante :

$$S_t(n) = S_t(1) + (n-1) \cdot \pi \qquad \forall t \in T \text{ et } n > 0$$

Le mode FPE est réalisable si et seulement si la relation suivante est vérifiée :

$$S_{\cdot p}(1) + \theta_{\cdot p} \leq S_{p\cdot}(1) + \pi . \tilde{M}_0(p), \quad \forall p \in P$$

où $\theta_{\cdot p}$ est la temporisation de la transition d'entrée de la place p et $M_0(p)$ est le marquage initial de cette place.

Les résultats présentés ci-dessus introduisent une méthode analytique simple, basée sur le calcul des chemins critiques, d'évaluation des performances d'un système modélisé par un graphe d'événements temporisé. Par la suite, nous allons étudier un problème de grande importance pour l'analyse des systèmes de production, lequel consiste à faire en sorte que le temps de cycle devienne inférieur ou égal à un temps de cycle donné à priori avec un minimum de jetons. Le lecteur pourra trouver une description plus détaillée de ce problème dans [Laftit *et al.* 92].

Comme nous l'avons souligné dans la section précédente, le problème qui se pose ici consiste en trouver un marquage initial qui garantit l'obtention d'un temps de cycle donné et qui, en même temps, minimise une combinaison linéaire invariante du nombre de jetons dans les places. Avant de donner une définition plus formelle de ce problème, nous avons besoin de la définition suivante :

Définition II.1

Soit α un réel positif. Le marquage M_0 est dit réalisable pour α si le temps de cycle $C(M_0)$ obtenu avec M_0 en régime FPE est inférieur ou égal à α.

Le problème à résoudre peut alors être formulé de la façon suivante :

$$\text{Trouver} \quad M_0^* \in E_\alpha \quad \text{tel que}$$
$$f_z(M_0^*) = \min_{M_0 \in E_\alpha} f_z(M_0)$$

où :

- E_α est l'ensemble des marquages réalisables pour α ;
- $f_Z(M) = Z^t \cdot M$, et Z est un p-invariant.

Dans la suite, ce problème sera noté $\mathcal{P}(\alpha, \text{FPE})$ si le mode de fonctionnement est périodique, ou par $\mathcal{P}(\alpha, \text{FPT})$ s'il s'agit de fonctionnement au plus tôt. Le but recherché est de

trouver des algorithmes pour résoudre le problème $\mathcal{P}(\alpha, FPT)$. Avant de décrire les algorithmes qui permettent de résoudre ce problème, nous présentons quelques résultats intermédiaires qui sont dûs à [Laftit *et al.* 92].

Résultat II.4

Le trois propositions suivantes sont équivalentes pour un graphe d'événements fortement connexe :

(i) M_0 *est réalisable pour* α *;*

(ii) $\mu(\gamma) \; / \; M_0(\gamma) \leq \alpha, \quad \forall \gamma \in \Gamma$ *;*

(iii) il existe un régime FPE tel que : $S_{\bullet p}(1) + \theta_{\bullet p} \leq S_{p\bullet}(1) + \alpha . M_0(p), \quad \forall p \in P$

Les conditions (ii) et (iii) du résultat II.4 s'appliquent dans le cas d'un régime FPE, tandis que dans le cas FPT seule la condition (ii) est vérifiée : la condition (iii) n'a aucune signification lorsque le régime est FPT

Résultat II.5

Soit θ_t *la temporisation de la transition t. Alors pour tout* $\alpha \geq \max_{t \in T} \theta_t$ *le marquage* $M_0 = <1,1,...,1>$ *qui consiste à mettre un jeton dans chaque place, est un marquage réalisable pour le problème* $\mathcal{P}(\alpha, FPE)$.

Résultat II.6

Si M_0^* *fait partie d'une solution optimale de* $\mathcal{P}(\alpha, FPE)$, *alors le marquage* M_0^* *est une solution optimale de* $\mathcal{P}(\alpha, FPT)$.

Résultat II.7

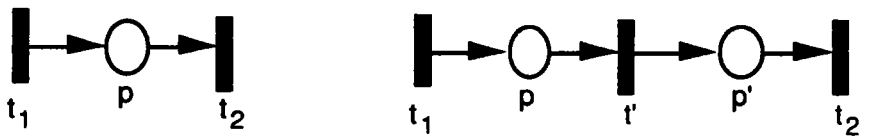
Il existe une solution optimale $\left\{ M_0^*, \quad S_{p\bullet}(1), \quad p \in P \right\}$ *du problème* $\mathcal{P}(\alpha, FPE)$ *telle que :*

(i) $S_{\bullet p}(1) \in \left[-\theta_{\bullet p}, \alpha - \theta_{\bullet p} \right], \quad \forall p \in P$

(ii) $M_0^*(p) \leq 2, \quad \forall p \in P$

Le Résultat II.7 indique qu'il existe une solution optimale telle que $M_0^*(p) \leq 2, \quad \forall p \in P$. Cependant, il est toujours possible de transformer un GE fortement connexe en un graphe équivalent ayant même la solution optimale et tel que la solution optimale de ce dernier vérifie : $M_0^*(p) \leq 1, \quad \forall p \in P$. Comme illustré dans la figure II.9, cette transformation consiste à remplacer toute place p par un triplet (p,t',p') dans lequel la transition t' est à temporisation

nulle. On obtient le Résultat II.8.



avant transformation après transformation

Figure II.9 : Transformation du graphe

Résultat II.8

Le problème $\mathcal{P}(\alpha, FPE)$ peut toujours être transformé de telle façon que la solution optimale $\left\{M_0^*,\ S_{p^*}(1),\ p \in P\right\}$ *vérifie :*

$$M_0^*(p) \leq 1, \quad \forall p \in P$$

Nous présentons maintenant trois algorithmes qui permettent de résoudre le problème $\mathcal{P}(\alpha, FPT)$. Nous nous contentons ici d'une description succincte du fonctionnement de chaque algorithme. Le lecteur intéressé pourra se reporter aux références citées pour une description détaillée. Le premier utilise un approche du type programmation linéaire. Le deuxième est appelé algorithme d'ajustement et le dernier est appelé algorithme d'optimisation convexe.

A- Algorithme basé sur la programmation linéaire ([Hillion et Proth 88])

Dans ce cas, la solution du problème $\mathcal{P}(\alpha, FPT)$ est déterminée par la résolution du problème de programmation linéaire suivant :

$$\text{Min } f_z(M_0)$$

sous les contraintes :

$\mu(\gamma) / M_0(\gamma) \leq \alpha, \ \forall \gamma \in \Gamma$, où Γ est l'ensemble de circuits élémentaires ;
$M_0(p) \in \{0,1\}, \forall p \in P$.

Ceci correspond à un problème de programmation linéaire du type 0-1. La difficulté majeure de cet algorithme est la nécessité de calculer tous les circuits élémentaires.

B - Algorithme d'ajustement ([Laftit *et al.* 92])

Cet algorithme, comme le précédent, résout directement le problème $\mathcal{P}(\alpha, \text{FPT})$. La présentation de cet algorithme nécessite quelques définitions et résultats supplémentaires.

D'après la partie (ii) du Résultat II.4, nous savons qu'un marquage M appartient à une solution admissible pour $\mathcal{P}(\alpha, \text{FPT})$ si et seulement si $\mu(\gamma) / M(\gamma) \leq \alpha$, pour tout circuit élémentaire γ. Nous pouvons récrire cette relation comme suit :

$$M(\gamma) - \mu(\gamma)/\alpha \geq 0.$$

Définissons alors $\omega(M,p,\alpha)$ comme :

$$\omega(M,p,\alpha) = \min_{\gamma \in \Gamma_p} \{M(\gamma) - \mu(\gamma)/\alpha\}$$

où Γ_p est l'ensemble de circuits élémentaires contenant p. Ainsi, la condition précédente peut s'écrire : un marquage M appartient à une solution admissible pour $\mathcal{P}(\alpha, \text{FPT})$ si et seulement si $\omega(M,p,\alpha) \geq 0$ pour tout $p \in P$. A partir de cela, nous avons le résultat suivant.

Résultat II.9

Soit $\mathcal{M}(\alpha)$ l'ensemble de marquages M tels que $\omega(M,p,\alpha) \in [0,1)$, $\forall p \in P$. Du fait que cette condition satisfait $\omega(M,p,\alpha) \geq 0$, on peut en déduire que tout $M \in \mathcal{M}(\alpha)$ correspond à une solution réalisable du problème $\mathcal{P}(\alpha, \text{FPT})$. Les deux propositions suivantes sont vérifiées :

(i) Soit $M \in \mathcal{M}(\alpha)$ et soit $M_1 \leq M$ et $M_1 \neq M$ (i.e. $M_1(p) \leq M(p)$ et pour au moins un $p \in P$, $M_1(p) < M(p)$). Alors M_1 n'est pas réalisable et, partant, ne correspond pas à une solution admissible du problème $\mathcal{P}(\alpha, \text{FPT})$.

(ii) $\mathcal{M}(\alpha)$ contient au moins une solution optimale du problème $\mathcal{P}(\alpha, \text{FPT})$.

L'algorithme d'ajustement est une heuristique basée principalement sur le Résultat II.9. A partir d'un marquage initial M réalisable (comme par exemple un jeton par place), il réduit itérativement d'un jeton le marquage total du GE jusqu'à trouver un marquage appartenant à $\mathcal{M}(\alpha)$. Le choix de la place où on enlève le jeton à chaque itération est effectué dans l'ensemble $F(M) = \{p / p \in P ; \omega(M,p,\alpha) \geq 1\}$. Il est important de souligner que $\omega(M,p,\alpha)$ représente, d'une certaine façon, le degré de liberté associé à la place p, car la probabilité d'enlever un jeton de la place p est d'autant plus grande que la valeur de $\omega(M,p,\alpha)$ est grande. En conséquence, on peut définir le degré de liberté total du système comme :

$$g(M) = \sum_{p \in P} \omega(M,p,\alpha)$$

Le critère qu'on adopte pour choisir une place, parmi les places appartenant à l'ensemble F(M), est le suivant : on choisira la place qui réduit au maximum la valeur de la fonction objectif et qui réduit au minimum le degré de liberté total du système. Pour permettre ce choix, nous introduisons la définition suivante. Soit M^1 le marquage obtenu après avoir enlevé un jeton de la place p, et soit Δ_p la quantité définie par :.

$$\Delta_p = \frac{[g(M) - g(M^1)]}{Z_p},$$

où Z_p est la composante correspondant à la place p dans le p-invariant Z (rappelons que ce p-invariant est utilisé dans la fonction objectif). On voit bien dans cette relation que la quantité Δ_p est proportionnelle à la variation du degré de liberté total, et inversement proportionnelle à la pondération de la place p, donc à la variation de la fonction objectif. Etant donné que ceci correspond bien au critère fixé antérieurement, nous allons choisir la place p* qui vérifie :

$$\Delta_{p^*} = \underset{p \in F(M)}{MIN}\ \Delta_p$$

Dans le cas où $M(p^*) = 0$, l'algorithme cherche un marquage $M_1 \in R(M)$ tel que $M_1(p^*) \geq 1$ (l'existence de ce marquage est assuré par le Résultat II.9) et il poursuit le processus à partir de M_1. Cet algorithme peut donc être résumé comme suit.

Algorithme II.2 :

1 - Générer un marquage initial réalisable.

2 - Si $\omega(M,p,\alpha) < 1$ pour tout $p \in P$, alors arrêter le calcul. Le marquage M est la solution du problème.

3 - Si $\omega(M,p,\alpha) \geq 1$ pour au moins une place $p \in P$, alors calculer p* telle que :

$$\Delta_{p^*} = \underset{p \in F(M)}{MIN}\ \Delta_p$$

4 - Si $M(p^*) \geq 1$ faire :

$$\begin{cases} M(p) = M(p), & \forall p \in P,\ p \neq p^* \\ M(p^*) = M(p^*) - 1. \end{cases}$$

Sinon, calculer le marquage M^1 accessible à partir de M, tel que $M^1(p^*) \geq 1$, et faire :

$$\begin{cases} M(p) = M^1(p) & \forall p \in P,\ p \neq p^* \\ M(p^*) = M^1(p^*) - 1. \end{cases}$$

5 - Aller en 2.

Un remarque importante concerne le calcul de $\omega(M,p,\alpha)$. Afin d'éviter la recherche de tous les circuits élémentaires contenant p, les auteurs montrent qu'il est possible d'obtenir la valeur de $\omega(M,p,\alpha)$ par le calcul du chemin minimal dans un graphe direct construit à partir du graphe d'événements.

Algorithme d'optimisation convexe ([Laftit *et al.* 92])

Cet algorithme fourni la solution optimale du problème $\mathcal{P}(\alpha, FPE)$, de laquelle on déduit celle du problème $\mathcal{P}(\alpha, FPT)$. Le modèle considéré ici est celui du GE transformé (voir figure II.9), pour lequel il existe un marquage optimal où chaque place contient au plus un jeton. Soit un marquage M, notons par $x_p = 1 - M(p)$ et $y_p = 1 - M(p')$ les complémentaires à 1 de M(p) et M(p') respectivement. M(p') étant le marquage associé aux places fictives créées dans la transformation. Le problème $\mathcal{P}(\alpha, FPE)$ peut alors s'écrire :

$$\text{Max} \sum_{p \in P} Z_p (x_p + y_p)$$

s.c.

$$S_{\bullet p}(1) + \theta_{\bullet p} - S_{p\bullet}(1) + \alpha(x_p + y_p) \leq 2\alpha, \quad \forall p \in P$$

$$x_p, y_p \in \{0,1\}, \forall p \in P$$

où Z_p est le coefficient correspondant à p dans la fonction objectif f_z du problème $\mathcal{P}(\alpha, FPE)$.

Ce problème est un problème de programmation linéaire du type mixte, dans lequel les variables x_p et y_p sont des entiers et les variables $S_{\bullet p}(1)$ et $S_{p\bullet}(1)$ sont des réels, pour tout $p \in P$. Nous nous réferons à ce problème comme le problème Q dans la suite.

Nous définissons ensuite les problèmes $\mathcal{P}_i$ (i = 1,2,...) comme suit :

$$\text{Max} \sum_{p \in P} Z_p (x_p^i + y_p^i)$$

s.c.

$$S_{\bullet p}(1) + \theta_{\bullet p} - S_{p\bullet}(1) + \alpha(x_p + y_p) \leq 2\alpha, \quad \forall p \in P$$

$$x_p, y_p \in [0,1], \forall p \in P$$

Les problèmes $\mathcal{P}_i$ sont des problèmes d'optimisation convexe avec variables réelles. Un algorithme pour résoudre ce type de problème est présenté dans [Benson 85]. Nous présentons maintenant le résultat qui constitue la base de l'algorithme d'optimisation convexe.

Résultat II.10

Soit $\{x^*_{p,i}, y^*_{p,i}, S^*_{\bullet p,i}\}_{p\in P}$ *une solution optimale du problème* P_i. *Si la condition suivante est vérifiée :*

$$\sum_{p\in P\setminus P_1} Z_p(x^*_{p,i})^i + \sum_{p\in P\setminus P_2} Z_p(y^*_{p,i})^i < 1$$

où $P1 = \{p / p \in P \text{ et } x^*_{p,i} = 1\}$ *et* $P2 = \{p / p \in P \text{ et } y^*_{p,i} = 1\}$, *alors* $\{\lfloor x^*_{p,i}\rfloor, \lfloor y^*_{p,i}\rfloor, S^*_{\bullet p,i}\}_{p\in P}$ *est une solution optimale de Q.*

$\lfloor\bullet\rfloor$ est l'entier immédiatement inférieur ou égal à $\bullet$.

Le processus d'obtention de la solution du problème $\mathcal{P}(\alpha, FPT)$ peut alors être résumé par les étapes suivantes :

1- Faire i = 2 ;
2- Calculer la solution optimale $S(\mathcal{P}_i)$ du problème $\mathcal{P}_i$;
3- Si l'inégalité présentée dans le Résultat II.10 n'est pas vérifiée, faire i+1 et aller en 2 ; sinon aller en 4 ;
4- Obtenir la solution optimale S(Q) du problème Q à partir de $S(\mathcal{P}_i)$ en faisant :

$$S(Q) = \{\lfloor x^*_{p,i}\rfloor, \lfloor y^*_{p,i}\rfloor, S^*_{\bullet p,i}\}_{p\in P}$$

5- La solution optimale $M^*_0(p)$ du problème $\mathcal{P}(\alpha, FPT)$ est alors :

$$M^*_0(p) \equiv 2 - \lfloor x^*_{p,i}\rfloor - \lfloor y^*_{p,i}\rfloor, \quad \forall p \in P$$

II.7.3 Graphes d'événements stochastiques

Nous nous plaçons maintenant dans le cas où les temporisations des transitions sont stochastiques, c'est-à-dire qu'à chaque transition $t \in T$ est associée une variable aléatoire $X_t \in \mathbb{R}^+$ qui définit la durée de franchissement de cette transition. Nous admettons également que le GE est fortement connexe. Soit X^k_t, k = 1,2,3... la séquence des durées de franchissement d'une transition $t \in T$. [Baceli *et al* 89] a montré que, si les séquences des durées de franchissement sont indépendantes pour tout $t \in T$, et si chaque séquence est formée par des variables aléatoires ergodiques et stationnaires, alors il existe une constante positive π, telle que :

$$\lim_{k\to\infty} \frac{S_t(k)}{k} = \pi \quad ps \qquad \forall t \in T \text{ et pour tout marquage initial } M \in R(M_0)$$

où St(k) est la fin du k-me franchissement de la transition t. La constante π est le temps de cycle moyen du GE.

L'expression ci-dessus montre que le temps de cycle moyen converge presque sûrement toujours vers la même valeur si on part du même marquage initial. On remarque aussi que la valeur de π ne dépend que du nombre initial de jetons dans chaque circuit élémentaire, i.e, $\pi(M_0) = \pi(M)$, pour tout marquage initial $M \in R(M_0)$.

Cependant la détermination du temps de cycle moyen π demeure très difficile, car aucune méthode analytique générale n'est disponible. On se contente alors de déterminer des bornes inférieure et supérieure, comme celles qui ont été établies par [Proth et Xie 93] et que nous présentons dans la suite.

Notons par m_t la valeur moyenne de X_t^k, et par s_t l'écart-type de X_t^k, i.e., $m_t = E\left[X_t^k\right]$ and $s_t^2 = E\left[\left(X_t^k - m_t\right)^2\right]$. Les bornes sont données par le Résultat II.11.

Résultat II.11

$$\underline{\pi} \le \pi(M_0) \le \overline{\pi}$$

où :

$$\underline{\pi} = \max_{\gamma\in\Gamma} E\left[\max\left\{\frac{\mu[\gamma \setminus \{t^*(\gamma)\}] + m_{t^*}(\gamma)}{M_0(\gamma)}, m_{t^*}(\gamma)\right\}\right]$$

$$\overline{\pi} = E\left[\max_{r\in R} \mu(r)\right]$$

et :

- *$t^*(\gamma)$ est la transition du circuit élémentaire γ ayant le plus grand temps de franchissement moyen , i.e.* $m_{t^*(\gamma)} = \max_{t\in\gamma} m_t$
- $\mu[\gamma \setminus \{t^*(\gamma)\}]$ *est la somme des temps de franchissement de toutes les transitions, sauf*

$t^*(\gamma)$, appartenant à γ, i.e, $\mu[\gamma \setminus \{t^*(\gamma)\}] = \sum_{t \in \gamma, t \neq t^*(\gamma)} X_t^k$

- *$M_0(\gamma)$ est le nombre initial de jetons dans γ.*
- *R est l'ensemble des chemins directs, tels que : (i) le chemin commence et fini par des places marquées, et (ii) aucune autre place du chemin n'est marquée.*
- *$\mu(r)$ est la somme des variables aléatoires associées aux transitions de r.*

Le problème d'atteignabilité d'un temps de cycle moyen C* dans un graphe d'événements stochastique a aussi été considéré dans [Laftit 91]. Il a montré qu'il est toujours possible d'atteindre C* avec un nombre fini de jetons si la condition $C^* > C_0 = \max_{t \in T} m_t$ est vérifiée, et il a proposé un algorithme heuristique qui permet de trouver un marquage initial M_0 qui minimise une combinaison linéaire p-invariante et qui permet d'atteindre le temps de cycle moyen spécifié. Cet algorithme fonctionne à deux temps :

i) Dans une première phase, il cherche la solution optimale du problème déterministe obtenu en attribuant à chaque transition une temporisation égale à la valeur moyenne de la variable aléatoire associée à cette transition. La solution du problème déterministe est obtenue par l'utilisation d'une des trois méthodes présentées dans la section II.7.2.1, et elle fournit une borne inférieure du temps de cycle moyen dans le cas stochastique. Cette phase correspond donc à la recherche d'une solution initiale.

ii) La deuxième phase est un processus itératif : à chaque itération un jeton est ajouté dans une place et le nouveau temps de cycle moyen du GE est évalué par simulation. Si ce nouveau temps de cycle moyen est inférieur ou égal à C*, l'algorithme s'arrête, sinon il passe à l'itération suivante. Le choix de la place est réalisé comme suit. On choisit d'abord le circuit élémentaire ayant le plus grand temps de cycle moyen, et on sélectionne, parmi les places de ce circuit, l'ensemble de places ayant le plus petit coefficient dans le p-invariant. Notons cet ensemble $\mathcal{P}$. Ensuite, on sélecionne tous les circuits élémentaires ayant un temps de cycle moyen grand et qui ont au moins une place en commun avec $\mathcal{P}$. On introduit le jeton dans une place appartenant à cette intersection.

Nous notons Γ l'ensemble de circuits élémentaires du graphe d'événements. Cet algorithme peut donc être formulé comme suit.

Algorithme II.3 :

Première phase

1 - Calcul du marquage initial optimal W_0 du problème dans lequel on assimile la

temporisation des transitions à la valeur moyenne des variables aléatoires associées à ces transitions. Ce problème consiste à atteindre le temps de cycle $C_0 = \max_{t \in T} m_t$ avec la minimisation de la fonction objectif $f_z(W_0) = Z^t \cdot W_0$ (Z est un p-invariant). Soit π_0 le temps de cycle obtenu.

2 - Utiliser les variables aléatoires en question pour générer les durées de franchissement des transitions, et simuler le système afin d'obtenir le cycle moyen Q_0 correspondent au marquage initial W_0.

3 - Si $Q_0 < C^*$, fin.

Deuxième phase

4 - Soit $\gamma_0 \in \Gamma$ le circuit élémentaire ayant le plus grand temps de cycle moyen et P_1 l'ensemble de places appartenant à γ_0 et ayant le plus petit coefficient dans le p-invariant Z. Soit $\gamma_1 \in \Gamma - \{\gamma_0\}$ le circuit élémentaire ayant le plus grand temps de cycle moyen et au moins une place en commun avec P_1, $\gamma_2 \in \Gamma - \{\gamma_0, \gamma_1\}$ le circuit élémentaire ayant le plus grand temps de cycle moyen et au moins une place en commun avec $P_1 \cap \gamma_1$, et ainsi de suite jusqu'à atteindre un circuit γ_q tel que :

$$P_1 \cap \gamma_1 \cap \gamma_2 \cap \ldots \cap \gamma_q = \emptyset.$$

5 - Ajouter un jeton dans la place appartenant à $P_1 \cap \gamma_1 \cap \gamma_2 \cap \ldots \cap \gamma_{q-1}$. Nous continuons d'appeler W_0 le nouveau marquage obtenu.

6- Simuler le système afin de déterminer le cycle moyen Q_0 correspondent au marquage W_0.

7 - Si $Q_0 < C^*$, W_0 est la solution du problème ; sinon aller dans l'étape 4.

La difficulté majeure de cet algorithme réside dans le fait qu'il nécessite le calcul de tous les circuits élémentaires du GE, ce qui peut réduire considérablement sa performance dans le cas d'un GE où le nombre de circuits est important.

II.8 RdP avec des transitions d'entrée et des transitions de sortie (RPES)

II.8.1 Définition d'un RPES

Comme nous l'avons souligné antérieurement, les graphes d'événements permettent de modéliser et évaluer le comportement des systèmes de production à fonctionnement cyclique. Notre but étant d'étudier les systèmes de production non-cycliques, nous définissons maintenant une nouvelle classe de RdP qui nous permettra de modéliser et évaluer de tels systèmes.

Dans le chapitre I, nous avons vu que la modélisation d'un système de production à fonctionnement non-cyclique doit représenter : i) l'arrivée et la sortie de produits du système et, ii) les décisions relatives aux choix d'alternatives quand plusieurs alternatives se présentent. Pour permettre cela, nous allons utiliser des transitions d'entrée, des transitions de sortie, et des places de contrôle.

Une *transition d'entrée* (ou transition source) est une transition qui ne possède que des places de sortie. Ce type de transition permet de modéliser l'introduction de produits dans le système, car son franchissement fait apparaître de nouveaux jetons dans chacune de ses places de sortie. Il est clair que ces transitions nécessitent un mécanisme de contrôle pour éviter qu'elles soient franchies un nombre illimité de fois. Ce mécanisme sera implanté à l'aide des places de contrôle qui nous définissons dans la suite.

Une *transition de sortie* (ou transition puits) se caractérise par le fait de ne posséder que des places en entrée. Donc le franchissement d'une transition de sortie permet de modéliser le fait qu'un produit quitte le système. Comme dans le cas des transitions d'entrée, nous allons associer une place de contrôle à chacune de transitions de sortie afin de contrôler le nombre de produits qui sortent du système.

Les *places de contrôle* [Krogh 87], dorénavant représentées par deux cercles concentriques, permettent d'intervenir dans le fonctionnement du réseau à partir de l'extérieur. Par une manipulation adéquate du marquage de ces places on peut , par exemple, résoudre des situations de conflit et éviter des marquages conduisant à des blocages ([Holloway et Krogh 90], [Ushio 90], [Ichikawa et Hiraishi 88]). Soit P_c l'ensemble de places de contrôle associées à un RdP. Une place de contrôle $p \in P_c$ a les caractéristiques suivantes :

(i) elle ne possède aucune transition d'entrée et une seule transition de sortie, i.e.,

$$^\bullet p = \varnothing \text{ et } |p^\bullet| = 1 ;$$

(ii) le marquage M(p) appartient à l'ensemble {0, 1} ;

(iii) le marquage ne peut être actualisé qu'à partir de l'extérieur et, en conséquence, il ne change pas avec le franchissement de sa transition de sortie.

Dans un RdP avec des places de contrôle, on distingue deux types de transitions : les transitions contrôlées et les transitions non-contrôlées. Une transition contrôlée n'est validée que si elle est validée par le marquage de ses places d'entrée et par le marquage de la place de contrôle associée. Dans le RdP contrôlé de la figure II.10, les deux places de contrôle c_1 et c_2 permettent de résoudre le conflit existant entre les transitions t_2 et t_3. Il est clair que l'on doit avoir au plus un jeton dans une des places c_1 ou c_2. La seule transition validée dans la figure est la transition t_2, et son franchissement provoque l'enlèvement du jeton de la place p_1 et l'arrivée

d'un jeton dans la place p_2. Le marquage de la place de contrôle c_1 reste inaltéré. Tant qu'aucune décision externe n'est prise pour changer le marquage des places de contrôle, le jeton arrivant dans la place p_1 sera utilisé pour le franchissement de t_2.

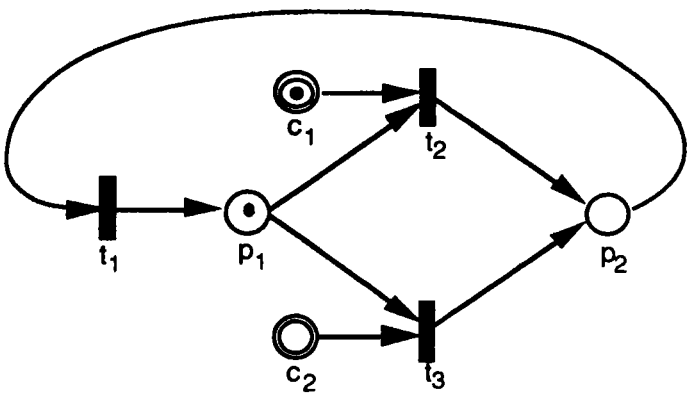


figure II.10 : Un RdP contrôlé

Nous pouvons maintenant définir une classe particulière des RdP, que nous appelons *RPES* (réseaux de Petri avec des transitions d'entrée et de sortie), laquelle sera utilisée pour la modélisation d'un système de production à fonctionnement non-cyclique.

Définition II.2

Un RPES est un RdP ordinaire vérifiant les conditions suivantes :

(i) il existe au moins une transition source et une transition puits ;

(ii) une place de contrôle est associée à chaque transition source, à chaque transition puits, et à toute transition dont le franchissement peut nécessiter la levée d'un conflit.

(iii) il n'y a ni places sources (places sans transition d'entrée) ni places puits (places sans transition de sortie) ;

Il est important de souligner ici que nous déterminons les marquages des places de contrôle (i.e. les décisions concernant la résolution des conflits et les décisions concernant l'entrée et la sortie de produits du système) à partir des résultats fournis par l'ordonnancement. Dans un souci de simplification, nous n'allons plus représenter les places de contrôle dans les RPES utilisés dans la suite.

La figure II.11 montre un exemple d'un RPES contenant deux transitions source (t_1 et t_2) et trois transitions puits (t_{12}, t_{13} et t_{14}) Dans cet exemple toutes les transitions sont contrôlées, à l'exception des transitions t_6 et t_{11}.

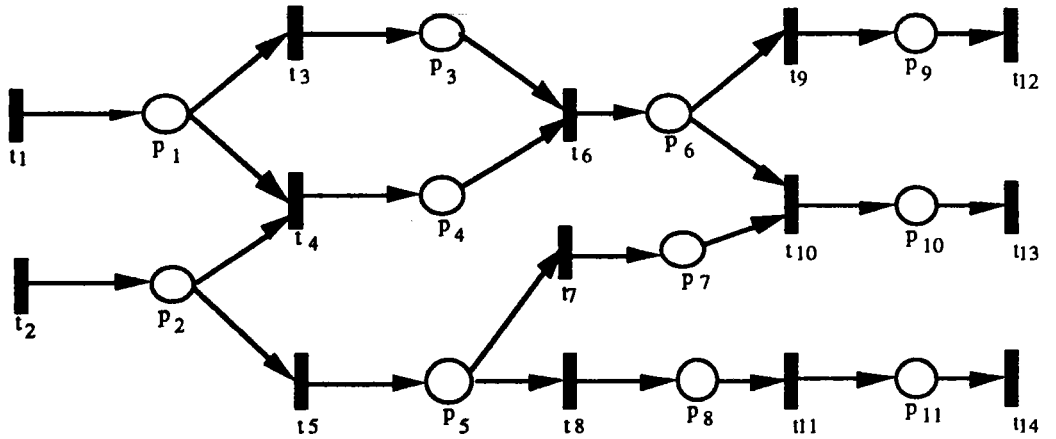


figure II.11 : Un exemple d'un RPES

Le modèle que nous venons de décrire nous permettra de modéliser le fonctionnement d'un système non-cyclique. L'approche que nous développons pour évaluer ce type de système est basée sur :

(i) la décomposition du modèle RPES en un ensemble de RPES sans conflit ;

(ii) l'utilisation des t-invariants de ces RPES sans conflit pour résoudre le problème de planification à court terme et d'ordonnancement.

Avant de présenter ces deux points en détails, nous allons définir les RPES sans conflit et présenter leurs propriétés.

II.8.2 RPES sans conflit

Nous définissons un RPES sans conflit comme suit.

Définition II.3

Un RPES sans conflit est un RPES dans lequel chaque place ne contient qu'une seule transition de sortie. En d'autres termes, le réseau est structurellement sans conflit, i.e.,

$$^{\bullet}t_1 \cap {}^{\bullet}t_2 = \varnothing, \quad \forall\, t_1, t_2 \in T, \text{ où } T \text{ est l'ensemble de transitions du RPES.}$$

En termes de structure, cette sous-classe de réseaux de Petri peut être caractérisée selon le diagramme de Venn de la figure II.12 ci-dessous. Dans cette figure le symbole RdP désigne un réseaux de Petri général, CL désigne un réseaux de Petri à choix libre, et RPES-sc désigne un RPES sans conflit. Comme indiqué dans cette figure, les RPES sans conflit sont un cas particulier des réseaux de Petri à choix libre.

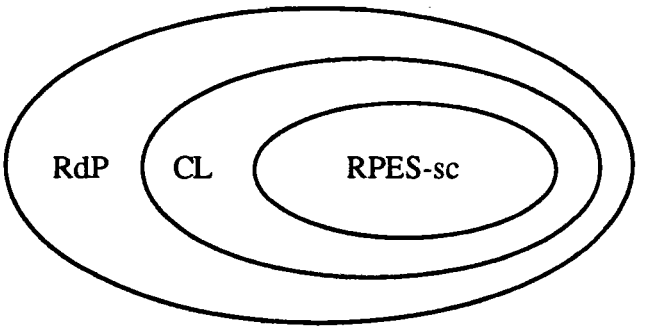


figure II.12 : Diagramme de Venn

Du point de vue des systèmes de production, trois propriétés qualitatives du modèle RdP sont particulièrement importantes :
- un RdP *vivant* garantit que toutes les opérations pour lesquelles le système a été défini peuvent être atteintes à partir de n'importe quel état du système ;
- un modèle RdP *borné* garantit qu'il n'y aura pas accumulation des en-cours dans le système. Donc, à partir de l'étude de cette propriété on pourra, par exemple, définir des tailles plus raisonnables des stocks tampons du système ;
- un RdP *réversible* assure que, quel que soit l'état du système, il est toujours possible de revenir à l'état de départ. Ceci est indispensable pour la manutention, pour des ajustements des outils ou pour des changements de production.

Avant de considérer ces propriétés dans le cas des RPES sans conflit, nous présentons d'abord la définition de verrou et de trappe et, ensuite, deux résultats qui seront nécessaires pour les démonstrations qui suivent.

Définition II.4

Un sous-ensemble de places P' d'un RdP constitue un verrou si toutes les transitions ayant une place de sortie appartenant à P' ont aussi une place d'entrée appartenant à P'. En d'autres termes, l'ensemble de transitions de sortie de P' contient l'ensemble de transitions d'entrée de P' ($^{\circ}P' \subseteq P'^{\circ}$).

Définition II.5

Une trappe est un sous-ensemble de places P' d'un RdP dans lequel l'ensemble de transitions d'entrée de P' contient l'ensemble de transitions de sortie de P'($P'^{\circ} \subseteq {}^{\circ}P'$).

Dans l'exemple de la figure II.13, le sous-ensemble de places $<p_1, p_2>$ est un verrou et le sous-ensemble de places $<p_3, p_4>$ est une trappe. Il est facile de remarquer que si la transition t_3 est franchie, le verrou ne sera plus jamais marqué tandis que la trappe, par contre,

sera marquée dans tous les marquages qui suivent. Ceci caractérise un verrou et une trappe.

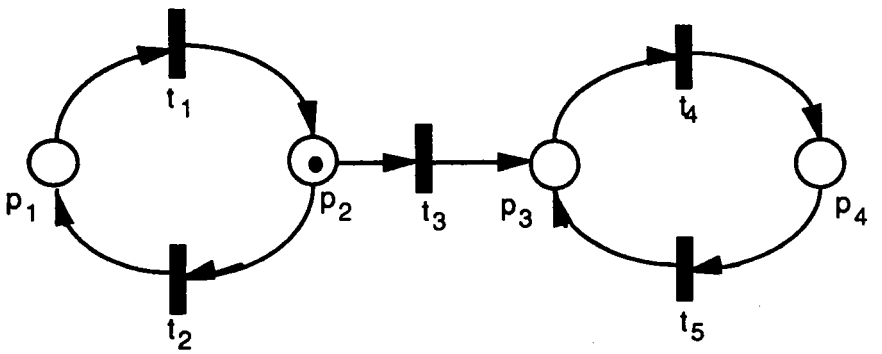


figure II.13 : Exemple de verrou et de trappe

Résultat II.12 [Murata 89]

Un RdP à choix libre marqué est vivant si et seulement si chaque verrou contient une trappe marquée.

Résultat II.13 [Murata 89]

Dans un RdP N où l'ensemble des places de chaque circuit élémentaire est une trappe, un marquage M est atteignable à partir du marquage initial M_0 si et seulement si il existe un vecteur entier non négatif x tel que :

(i) x est une solution de $M = M_0 + C \cdot x$;

(ii) le sous-réseau N_x obtenu à partir de N par la suppression de toutes les transitions pour lesquelles $x(t) = 0$, ne contient pas de verrou non marqué.

Nous pouvons maintenant étudier les propriétés des RPES sans conflit, et plus spécifiquement celles qui nous intéressent du point de vue des systèmes de production que nous avons présentés antérieurement. Comme nous allons voir dans la suite, le fait d'avoir un RPES sans conflit consistant est une condition nécessaire pour qu'on puisse garantir les propriétés recherchées. Ainsi, nous présentons le résultat suivant qui donne une condition suffisante pour qu'un RPES sans conflit ne soit pas consistant.

Résultat II.14

Soit N un RPES sans conflit et soit Γ l'ensemble de tous les circuits élémentaires de N. Si pour au moins un circuit élémentaire $\gamma \in \Gamma$, il existe une place $p \in \gamma$ telle que p possède plus d'une transition d'entrée, alors N n'est pas consistant.

Preuve :

Supposons que N soit consistant et qu'il existe un circuit élémentaire γ et une place $p \in \gamma$

telle que p possède plus d'une transition d'entrée.

D'après la définition de consistance, il existe un marquage initial M_0 et une séquence de franchissement des transitions σ où chaque transition franchit au moins une fois, telle que $M_0[\sigma > M_0$.

Comme N est sans conflit, $\forall p \in \gamma \Rightarrow p^\circ \in \gamma$. Ceci implique que le nombre de jetons dans γ ne peut pas être réduit. Maintenant, si $\exists p \in \gamma$ telle que ${}^\circ p - \gamma \neq \emptyset$, alors le franchissement de toute transition $t \in p - \gamma$ ira augmenter le nombre de jetons dans γ. La transition t apparaît au moins une fois dans la séquence σ et, en conséquence, le franchissement de σ ira augmenter le nombre de jetons dans γ. Pourtant on ne pourra pas retrouver le marquage initial M_0 et N n'est pas consistant.

Q.E.D.

Le résultat suivant concerne la propriété de vivacité d'un RPES sans conflit.

Résultat II.15

Si un RPES sans conflit est consistant et si chaque circuit élémentaire contient au moins un jeton, alors il est vivant.

Preuve :

Nous montrons d'abord que, dans un RPES sans conflit, tout verrou contient un circuit élémentaire et, ensuite, que ce circuit correspond à une trappe. D'après le résultat II.12, le RPES est vivant.

Soit P' un verrou du réseau et soit $p_i \in P'$ une place quelconque du verrou. D'après la définition de RPES sans conflit, p_i possède une transition de sortie et au moins une transition d'entrée. Soit t une transition d'entrée de la place p_i. D'après la définition de verrou, nous savons que $\exists p_j \in {}^\circ t$ telle que $p_j \in P'$. Ceci implique que la transition t n'est pas une transition d'entrée du RPES. Si la place p_j est identique à la place p_i, nous avons déjà un circuit (en l'occurrence une boucle élémentaire) et nous arretôns les calculs. Sinon, considérons une transition d'entrée t' de la place p_j. Pour la même raison que dans le cas précédent, nous pouvons également écrire pour cette transition : $\exists p_k \in {}^\circ t'$ telle que $p_k \in P'$. Comme la transition t, t' n'est pas non plus une transition d'entrée du RPES. Si la place p_k est identique à une des places p_i ou p_j nous avons un circuit élémentaire, et nous arretôns les calculs. Dans le cas contraire, on continue à partir de la place p_k et ainsi de suite. Puisque le nombre de places est fini, ce processus conduit forcément à une place que nous avons déjà rencontrée, ce qui signifie qu'on obtiendra un circuit élémentaire.

A partir du résultat II.14, nous savons que chaque place appartenant à un circuit élémentaire d'un RPES sans conflit consistant possède une seule transition d'entrée et une seule transition de sortie. Dans un tel circuit, l'ensemble de transitions d'entrée des places du circuit est égal à l'ensemble de transitions de sortie de ces mêmes places. Ceci correspond donc à la définition d'une trappe.

Q.E.D.

Le résultat suivant montre qu'un RPES sans conflit n'est pas structurellement borné. Cependant, nous allons voir dans la section suivante qu'il est possible de contrôler le franchissement de ses transitions de façon à ce qu'il reste toujours borné. Rappelons qu'n RdP est structurellement borné s'il est borné pour tout marquage initial M_0 fini.

Résultat II.16

Un RPES sans conflit n'est pas structurellement borné

Preuve :

Ce résultat est immédiat étant donné la structure d'un RPES sans conflit.

Q.E.D.

Le résultat suivant permet de définir des conditions dans lesquelles un RPES sans conflit est réversible.

Résultat II.17

Si un RPES sans conflit est consistant et si chaque circuit élémentaire contient au moins un jeton, alors pour tout marquage initial M_0 et pour tout $M \in R(M_0)$ nous avons $M_0 \in R(M)$ (i.e. le réseau est réversible).

Preuve :

Pour tout marquage $M \in R(M_0)$ les conditions suivantes sont vérifiées :

(a) Il existe une séquence finie σ telle que $M_0[\sigma > M$. Nous pouvons donc écrire $M = M_0 + C \cdot \underline{\sigma}$, où $\underline{\sigma}$ est le vecteur caractéristique de σ ;

(b) Etant donné que le réseau est consistant, alors il existe un vecteur x, $x_i > 0$, i = 1,...,n où n est le nombre de transitions, tel que $C \cdot x = 0$. Donc, pour tout entier k positif nous avons $C \cdot k \cdot x = 0$;

(c) A partir de (a) et de (b) nous avons :

$$M = M_0 + C \cdot k \cdot x + C \cdot \underline{\sigma}$$

ou,

$$M_0 = M + C \cdot (k \cdot x - \underline{\sigma})$$

Or nous pouvons choisir k de telle sorte que $X = (k \cdot x - \underline{\sigma}) > 0$ et $X_i > 0$. D'où,

$$M_0 = M + C \cdot X$$

(d) Dans la preuve du résultat II.15 nous avons vu que dans un RPES sans conflit N consistant, chaque verrou contient un circuit élémentaire, et que celui-ci correspond à une trappe (chaque trappe est marquée d'après les hypothèses de départ et, par conséquent, chaque verrou l'est aussi). En (c) nous avons défini un vecteur X ayant de composantes strictement positives qui est solution de $M_0 = M + C \cdot X$. Ainsi, à partir du résultat II.13 nous pouvons conclure que le marquage M_0 est atteignable depuis le marquage M (le réseau N_x défini dans la condition (ii) de ce résultat II.13 est, dans notre cas, identique à N).

Q.E.D.

Les résultats que nous venons de présenter permettent d'établir les conditions dans lesquelles un RPES sans conflit est vivant et réversible. En plus, du fait de la consistance du réseau, ces résultats imposent la présence d'au moins un jeton dans chaque circuit élémentaire. Il est clair que si le réseau considéré ne possède pas de circuit élémentaire, alors on peut récrire les résultats II.15 et II.17 comme suit.

Résultat II.18

Un RPES sans conflit consistant dans lequel il n'y a pas de circuit élémentaire est vivant et réversible.

Dans la section qui suit, nous allons étudier les caractéristiques que doivent vérifier les modèles RPES d'un système de production pour qu'on puisse les utiliser dans la résolution des problèmes de planification à court terme et d'ordonnancement. En particulier, nous définissons la notion de RPES décomposables, et nous montrons que les RPES décomposables ont toutes les propriétés qualitatives souhaitées pour les systèmes de production.

II.8.3 RPES décomposables

La notion de décomposabilité est le point clé de la démarche que nous utilisons pour

résoudre le problème de planification à court terme. L'introduction de cette notion sera réalisée de façon progressive : dans un premier temps, nous considérons la décomposabilité en termes de RPES sans conflit et consistants. Nous présentons les conditions que doivent vérifier les RPES décomposables et nous étudions leurs propriétés qualitatives. Ensuite, nous montrons que les résultats obtenus sont aussi valables si on considère l'ensemble de sous-réseaux obtenus à partir des composantes positives des t-invariants à support minimal des RPES sans conflit. Enfin, nous discutons le problème de la flexibilité opératoire du système en considérant le temps nécessaire pour résoudre le problème de planification..

La contenu du paragraphe précédent indique que nous allons extraire d'un modèle RPES tous les sous-réseaux qui représentent des RPES sans conflit. La définition suivante donne les conditions que doivent vérifier chacun de ces sous-réseaux pour qu'ils puissent être considérés comme des RPES sans conflit.extraits d'un RPES.

Définition II.6

Soit N un RPES. Un sous-réseau NC extrait de N est un RPES sans conflit si :

(i) NC est un RPES sans conflit connexe ;

(ii) les transitions sources (resp. puits) de NC sont des transitions sources (resp. puits) de N ;

(iii) toute place (resp. transition) de NC est une place (resp. transition) de N ;

(iv) pour toute transition $t \in NC$, l'ensemble des places d'entrée (resp. de sortie) de t dans NC est identique à l'ensemble des places d'entrée (resp. de sortie) de t dans N.

Nous introduisons maintenant la définition d'un RPES décomposable en termes de RPES sans conflit. Il est important de souligner ici que dans la suite nous ne considérons que les RPES dans lequel il n'y a pas de circuits élémentaires.

Définition II.7

Soit N un RPES dans lequel il n'y a pas de boucles et soit $\{NC_1, NC_2, \ldots, NC_r\}$ l'ensemble de RPES sans conflit extraits de N. Si les conditions suivantes sont vérifiées :

(i) chaque RPES sans conflit NC_i, $i = 1,\ldots,r$ est consistant ,

(ii) l'ensemble des RPES sans conflit couvrent N, i.e., $N = \bigcup_{i=1}^{r} NC_i$,

alors N est un RPES décomposable.

Un RPES NC_i sans conflit et consistant extrait de N contient les transitions correspondant aux composantes positives d'un t-invariant de N, et seulement celles-là.(ceci est

une conséquence du fait qu'il est consistant). Donc, la définition de décomposabilité peut aussi être exprimée en termes de ces t-invariants. Nous allons considérer les propriétés qualitatives des RPES décomposables, à commencer par la propriété de consistance.

Résultat II.19

Un RPES décomposable N est consistant.

Preuve :

Soit C la matrice d'incidence de N et soit NC_1, NC_2,..., NC_r l'ensemble de RPES sans conflit consistants extraits de N.

Le fait de que NC_i (i = 1,...,r) est consistant implique en qu'il existe un vecteur

$$x^i = \left[x_1^i, x_2^i, \ldots, x_q^i\right]^{Tr}$$

tel que :

$$x_k^i > 0 \text{ si } t_k \in NC_i,\ x_k^i = 0 \text{ si } t_k \notin NC_i, \text{ et } Cx^i = 0.$$

Soit le vecteur x^* définit de la façon suivante :

$$x^* = \left[x_1^*, x_2^*, \ldots, x_q^*\right]^{Tr}$$

tel que :

$$x_k^* = \sum_{i=1}^{r} x_k^i, \text{ pour } k = 1, 2, \ldots, q.$$

$x_k^i > 0$ pour tout k = 1, 2,...,q, car toute transition t_k appartient au moins à un RPES sans conflit, et pourtant un des x_k^i (k=1,2,...,q) est strictement positif pour au moins un $i \in \{1, 2, \ldots, q\}$. Donc, nous pouvons écrire :

$$Cx^* = C\sum_{i=1}^{r} x^i = \sum_{i=1}^{r} Cx^i = 0$$

Ceci implique que N est consistant.

Q.E.D.

Dans la suite nous nous intéressons aux propriétés de vivacité et réversibilité des RPES décomposables. Proth *et al.* [Proth *et al.* 94] ont récemment étudié les RPES sans boucles et ont généralisé les résultats obtenus auparavant dans le cadre des RPES décomposables. En particulier, ils ont démontré qu'un RPES sans boucles est toujours vivant et que, si en plus le réseau est consistant, alors il est aussi réversible. Ainsi, à partir de ces résultats et en prenant en compte le résultat II.19, on a le résultat suivant :

Résultat II.20

Un RPES décomposable N est vivant et réversible.

Il ne nous reste plus qu'à considérer le problème qui consiste à faire en sorte que le réseau reste borné. Comme nous l'avons souligné antérieurement, la présence des transitions d'entrée dans le modèle RPES permet l'introduction d'un nombre illimité de jetons dans le réseau et, en conséquence, un RPES n'est pas structurellement borné. Néanmoins, si on accepte de restreindre le fonctionnement du RPES de telle sorte que celui-ci consiste à activer des ensembles de transitions correspondant aux composantes positives de t-invariants, alors on pourra garantir que le réseau sera toujours borné. En d'autres termes, le franchissement d'une transition t un certain nombre de fois est lié au franchissement de toutes les transitions relatives à un t-invariant contenant la transition t en tenant compte des composantes du t-invariant. En clair, cela signifie que l'apport total de jetons dans le réseau par cette séquence est nul et qu'on impose que le RPES revienne à son état de départ après chaque cycle de franchissements. Le nombre d'activations de chaque ensemble de transitions sera déterminé de façon à satisfaire le nombre de franchissements des transitions de sortie.

Ce qui vient d'être dit est résumé dans le résultat suivant.

Résultat II.21

Un RPES décomposable N dont le fonctionnement consiste à activer des ensembles de transitions correspondant à des t-invariants (c'est-à-dire le support d'un t-invariant), est maintenu borné. Le nombre d'activations des supports de t-invariants est fini si le nombre de franchissements des transitions de sortie est fini.

Preuve :

Soit NC_1, NC_2,..., NC_r l'ensemble de RPES sans conflit consistants qui couvrent N et soit X_i le t-invariant associé à chaque RPES sans conflit NC_i, i = 1, ..., r. Soit encore k1, ..., ks le nombre de fois que chaque transition de sortie $t_{o,1}$, ..., $t_{o,s}$ doit être franchie. Définissons n^i_j comme le composant du t-invariant X_i lequel correspond à la transition $t_{o,j}$. Au

moins un des n^i_j , i = 1, ..., r, est strictement positif pour tout j ∈ {1, ..., s}, et au moins un des n^i_j, j = 1, ..., s, est strictement positif pour tout i ∈ {1, ..., r}. Pourtant, le problème de programmation linéaire suivant :

$$Min \sum_{i=1}^{r} y_i$$

s. c.

$$\sum_{i=1}^{r} y_i . n^i_j \geq k_j, \quad j = 1,2,...,s$$

$$y_i \in \{0,1,...,\}, \quad i = 1,2,...,r$$

a une solution finie. L'entier y_i correspond au nombre de fois que chaque ensemble de transitions appartenant au support de chaque t-invariant X_i doit être franchi, de façon à franchir les transitions de sortie le nombre de fois souhaités. Pourtant, le marquage M(p) de toute place $p \in t^\circ_v$, où t°_v est la transition correspondante au $v^{éme}$ composant du t-invariant X_i, est borné par :

$$M_0(p) + \sum_{i=1}^{r} y_i . x^i_v, \qquad \text{où } x^i_v \text{ est } v^{éme} \text{ composant de } X_i$$

Cette relation est valable pour tout p ∈ P.

Q.E.D.

Il est important de souligner ici que du fait que chaque RPES sans conflit est consistant, nous savons qu'il est couvert par un ou plusieurs t-invariants à support minimal. Donc, on peut considérer la décomposition de ces RPES sans conflit en un sous ensemble de réseaux selon les composantes positives de leurs t-invariants à support minimal. Pour ce faire, ces sous-réseaux doivent vérifier des conditions équivalentes à celles données dans la définition II.6 et, en plus, leur union doit couvrir le RPES sans conflit duquel ils ont été extraits. Il est clair que ces sous-réseaux correspondent aussi à des RPES sans conflit et, en conséquence, que les résultats que nous avons présentés antérieurement restent valables si on décompose le modèle RPES dans cet ensemble de sous-réseaux.

Comme nous allons le montrer dans le chapitre 4, le problème que nous serons amenés à résoudre dans la planification à court terme consistera à fabriquer, dans chaque période

élémentaire, autant de produits de chaque type que les demandes définies pour ces types de produits. En même temps, nous essayerons de trouver une solution qui minimise un certain critère de fonctionnement du système. Nous rappelons que dans notre cas, le nombre de produits de chaque type fabriqués au cours d'une période élémentaire est la donnée du nombre total de franchissements des transitions de sortie du modèle RPES de ce produit. La démarche de résolution de ce problème est basée sur la combinaison des t-invariants dont les supports représentent les sous-réseaux consistants extraits du modèle RPES. D'après les résultats que nous venons de présenter, cette démarche garantira que le modèle aura toutes les propriétés qualitatives intéressantes du point de vue des systèmes de production.

En effet, la façon de décomposer le modèle (ce qui revient à définir les t-invariants qu'on utilisera) est étroitement liée au degré de flexibilité opératoire que nous imposons au système. Ainsi, si on décompose le modèle de telle sorte que les transitions des sous-réseaux correspondent au support des t-invariants à support minimal, on obtient le plus grand degré de flexibilité du système. Par contre, ceci peut augmenter beaucoup la complexité des calculs de la solution du problème de planification à court terme, car le nombre de combinaisons des sous-réseaux peut être plus élevé. Cependant, dans certains cas, on peut se contenter d'avoir des résultats moins précis contre un gain de temps au niveau du calcul de la solution du problème considéré. On utilisera donc, dans ce cas, non plus les sous-réseaux dont les transitions représentent le support d'un t-invariant à support minimal mais d'autres sous-réseaux obtenus par combinaison linéaire des premiers. Il ne faut pas oublier que l'ensemble de sous-réseaux utilisés doit couvrir le réseau initial.

Dans les exemples que nous présentons dans les chapitres IV et V nous ne considérons que la décomposition en termes de sous-réseaux correspondant à des t-invariants à support minimal. Comme nous allons le montrer, le processus d'obtention des sous-réseaux qui nous intéressent est à deux temps : d'abord on décompose le modèle en un ensemble de RPES maximaux sans conflit (i.e. un RPES sans conflit que n'est pas contenu dans d'autres RPES sans conflit), et ensuite on décompose chaque RPES maximal sans conflit en des sous-réseaux correspondantes à des t-invariants à support minimal.

II.9 Conclusion

Nous pouvons séparer ce chapitre en trois parties principales. Dans la première partie nous avons fait un rappel sur les réseaux de Petri et, en particulier, nous avons insisté sur tous les aspects qui sont nécessaires pour la compréhension du travail que nous présentons dans ce mémoire. La deuxième partie a été consacrée à l'étude d'une classe particulière de réseaux de Petri : les graphes d'événements. Nous avons présenté les principaux résultats concernant

l'évaluation des performances et l'optimisation du marquage des graphes d'événements temporisés, tant en ce qui concerne le cas d'une temporisation déterministe que le cas d'une temporisation stochastique. Dans la troisième partie du chapitre nous avons défini et étudié les RPES (les réseaux de Petri avec des transitions sources et des transitions puits), que nous allons utiliser pour modéliser et évaluer les systèmes de production non-cycliques.

Chapitre III

Systèmes de production cycliques

III.1 Introduction

Ce chapitre est dédié à l'étude des systèmes de production cycliques (à fonctionnement répétitif). En d'autres termes, nous considérons les systèmes dans lesquels la fabrication des différents produits se fait suivant des ratios donnés. Comme nous l'avons souligné antérieurement, cette particularité des systèmes cycliques conduit à un contrôle optimal qui est aussi périodique et qui, en conséquence, peut être intégré au modèle du système. De même, le fait d'avoir des ratios de production qui ne varient pas d'une période à la période suivante permet de modéliser ces systèmes à l'aide des graphes d'événements, dont les propriétés ont été présentées dans le chapitre précédent. Nous utiliserons ces résultats dans l'étude des différents exemples de systèmes cycliques considérés dans ce chapitre.

Trois exemples de systèmes de production sont considérés dans cette étude : les ateliers flexibles (en anglais "job-shops"), les systèmes d'assemblage et les systèmes du type juste-à-temps (Kanban). Après avoir introduit le problème de la modélisation de ces systèmes, nous considérons le problème de l'optimisation de leurs performances. Ce dernier problème consiste à obtenir un temps de cycle donné tout en minimisant le nombre de ressources de transport utilisées.

Les résultats que nous allons présenter dans les sections qui suivent ont été proposés par Hillion [Hillion 89] et Laftit [Laftit 91]. Le lecteur pourra aussi trouver une bonne partie de ces résultats dans [Hillion et Proth 89] et [Laftit *et al.* 92].

Ce chapitre est organisé en trois sections. Dans la deuxième section nous définissons les systèmes considérés et nous montrons comment ils peuvent être modélisés à l'aide des graphes

d'événements. Dans la troisième section nous présentons le problème de l'optimisation des performances de ces systèmes. La résolution de ce problème est illustrée à l'aide de deux exemples numériques qui concernent le cas des ateliers flexibles.

III.2 Définition et modélisation de quelques systèmes de production

Dans cette section nous considérons le problème de la modélisation de quelques systèmes de production fonctionnant sur une base cyclique à l'aide des graphes d'événements. Après une brève définition des caractéristiques et des hypothèses utilisées pour chaque type de système considéré, nous décrivons les différentes étapes de la modélisation. Les modèles auxquels nous aboutissons correspondent à des graphes d'événements fortement connexes, ce qui permet l'utilisation des résultats décrits dans le chapitre II.

III.2.1 Modélisation d'un atelier flexible (job-shop)

Le premier exemple que nous considérons correspond au cas d'un atelier flexible. Dans un atelier flexible nous disposons d'un certain nombre de machines qui sont utilisées pour la fabrication de différents types de produits. La fabrication de chaque type de produit est réalisée par le passage de ce produit sur certaines des machines, suivant un ordre établi. On dira que le passage d'un produit sur une machine définit une *opération élémentaire*, et que l'ordre de passage du produit dans les différentes machines définit le *routage* du produit. Si de plus on associe à chaque machine le temps nécessaire pour réaliser l'opération sur le produit, on obtient la *gamme de fabrication* du produit. Enfin, pour compléter la définition d'un atelier flexible il faut définir l'ordre dans lequel les différents produits passent sur chacune des machines, ce qu'on appelle dans la suite les *séquences d'entrée* dans les machines.

Nous pouvons donc définir un atelier flexible comme étant composé :

- d'un ensemble de machines, soit $M = \{M_1, M_2,\ldots,M_r\}$;
- de l'ensemble des types de produits à fabriquer, soit $P = \{P_1, P_2,\ldots,P_q\}$;
- de la gamme de fabrication γ_i du produit P_i, $i = 1, 2,\ldots,q$. La gamme de fabrication du produit P_i est donnée par :

$$P_i : M_{j_1}(\theta_{i,j_1}), M_{j_2}(\theta_{i,j_2}),\ldots,M_{j_k}(\theta_{i,j_k})$$

où la séquence $\{M_{j1}, M_{j2},\ldots,M_{jk}\}$ correspond au routage de P_i et $\theta_{i,j}$ correspond à la durée du passage du produit P_i sur la machine M_j ;

- de la séquence d'entrée $S_j = \{P_{i1}, P_{i2},\ldots,P_{is}\}$ dans chacune des machines utilisées.

Une remarque importante est que le routage de chaque produit est supposé unique et fixé à priori. Donc, chaque opération définie par ce routage ne peut être réalisée que par une et une seule machine. Cependant, comme l'a souligné Hillion [Hillion 89], cette hypothèse n'est pas restrictive en soi car dans le cas où plusieurs routages sont possibles, on doit d'abord trouver une politique que fixe les pourcentages de chaque type de produit affectés aux routages possibles. Une fois cette répartition réalisée, il suffit de définir les produits.par son routage et non plus par ses caractéristiques physiques.

L'hypothèse d'une fabrication répétitive impose qu'on spécifie les ratios de production souhaités pour chaque type de produit fabriqué dans l'atelier. A partir de cela on peut introduire la définition de cycle élémentaire de production. Un *cycle élémentaire* correspond au temps nécessaire pour produire les quantités minimales de chaque produit qui respectent les ratios. Pour satisfaire la production totale, il suffit alors de relancer un nombre déterminé de fois le cycle élémentaire de production.

Il est important de souligner que les ratios de production doivent être pris en compte au moment de la définition des séquences d'entrée dans les machines. Pour illustrer ceci nous allons considérer l'exemple d'un atelier flexible composé de trois machines, M_1, M_2 et M_3, et qui peut produire trois types de produits, P_1, P_2 et P_3. Les gammes de fabrication des produits sont les suivantes :

P_1 : $M_1(3)$, $M_2(1)$, $M_3(2)$,
P_2 : $M_1(1)$, $M_2(4)$,
P_3 : $M_1(2)$, $M_3(3)$.

Supposons que les ratios de production souhaités sont les suivants :

25% de produits du type P_1,
50% de produits du type P_2,
25% de produits du type P_3,

ce qui signifie qu'on doit produire deux fois plus de produits de type P_2 que de produits de type P_1 et P_3. Ainsi, un ensemble possible pour les séquences d'entrée dans les trois machines, M_1, M_2 et M_3, est :

S_1 : (P_1, P_2, P_2, P_3),
S_2 : (P_2, P_2, P_1),
S_3 : (P_1, P_3).

Comme on peut le voir, un produit n'apparaît dans une séquence que si la machine concernée fait partie du routage de ce produit. On constate aussi que les ratios fixés sont respectés, car la proportion de P_2 est deux fois plus grande que celle de P_1 et de P_3 dans toutes les séquences où le produit P_2 apparaît. Une dernière remarque est que toute séquence obtenue par permutation des éléments dans une des séquences précédentes, est aussi une séquence d'entrée valable pour l'exemple considéré.

Nous allons utiliser cet exemple d'atelier flexible tout au long de cette section. Mais avant de montrer comment le modéliser à l'aide des graphes d'événements, nous introduisons deux hypothèses qui sont retenues par la suite. La première est que le nombre de ressources de transport est limité et que chaque ressource de transport ne peut transporter qu'un seul type de produit. En d'autres termes, ceci signifie que la ressource de transport suit le produit tout au long de son routage et que, dès que celle-ci est libéré, elle est immédiatement réutilisée et vient se placer en attente devant la première machine sur laquelle le produit doit passer. La deuxième hypothèse concerne les temps de transport d'une machine à l'autre et les temps de changements d'outils dans les machines. Nous supposons que ces temps sont négligeables par rapport aux temps opératoires ; ils sont donc fixés à zéro. Le lecteur pourra consulter [Hillion 89] pour une discussion sous la manière d'introduire ces temps dans le modèle, et sur les conséquences que ceci entraîne au niveau de l'analyse des performances.

La modèle de l'atelier flexible est construit en deux étapes : d'abord on modélise les gammes de fabrication et, ensuite, on ajoute les circuits de commande.

Nous partons d'une séquence S qui exprime les ratios de production souhaités. Pour l'exemple considéré précédemment (25% de P_1, 50% de P_2 et 25% de P_3), la séquence S peut être la suivante :

$S = (P_1, P_2, P_2, P_3)$.

Nous représentons autant de gammes de fabrication que d'éléments dans cette séquence (dans ce cas nous allons en avoir 4 circuits : 1 pour P_1, 2 pour P_2 et 1 pour P_3). Ce modèle est illustré par la figure III.1 ci-dessous.

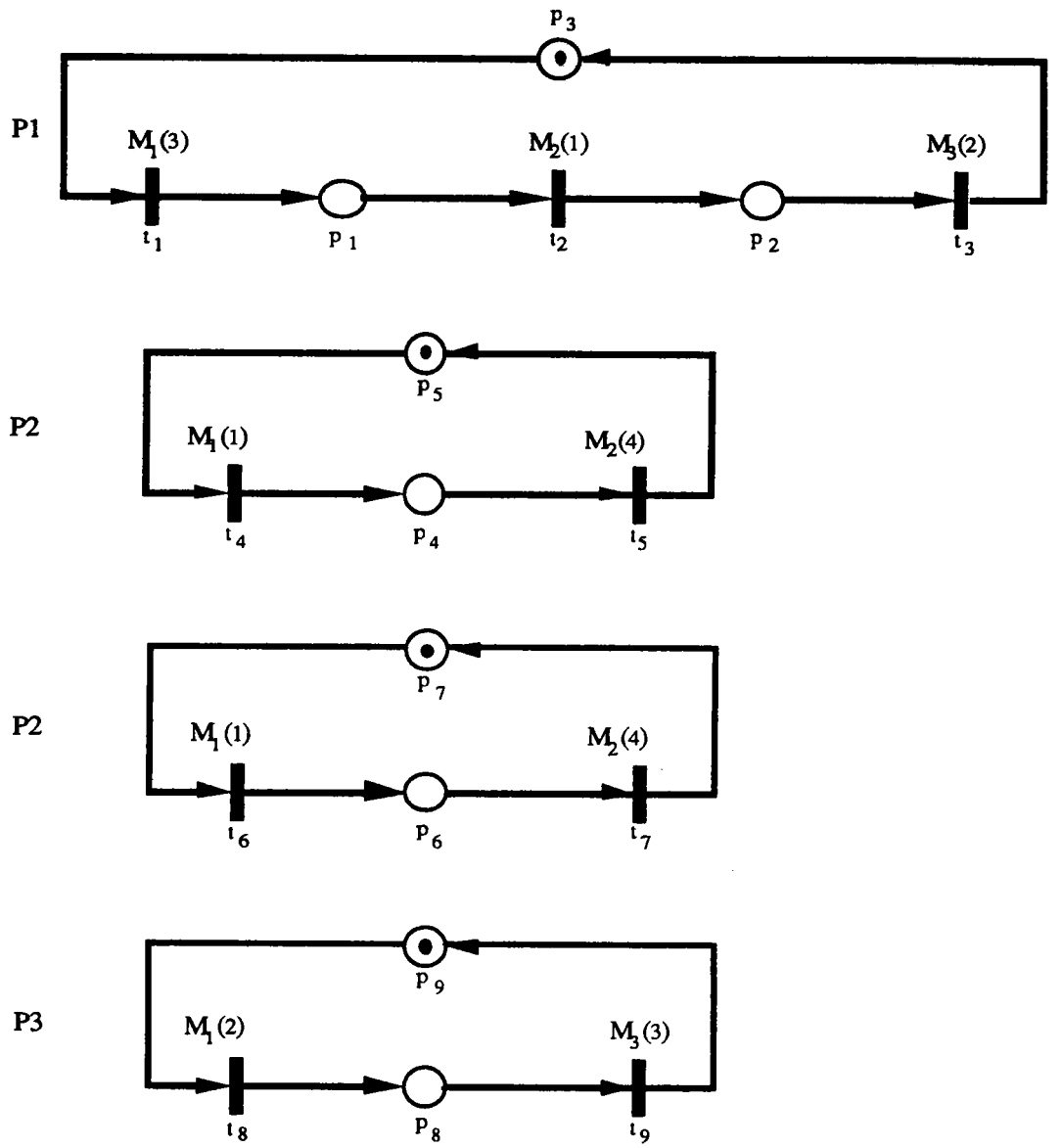


figure III.1 : Représentation des gammes de fabrication

Comme nous pouvons le constater dans cette figure, nous avons introduit une boucle pour chacune des gammes. Cette boucle indique qu'un nouveau produit sera lancé en production dès qu'un produit sera terminé. Le jeton qui figure dans une des places de bouclage (p_3, p_5, p_7, p_9) représente donc le fait que la ressource de transport est chargée avec un nouveau produit, et qu'elle attend la disponibilité de la machine qui doit réaliser la première opération de la gamme de fabrication du produit. Les jetons dans les autres places représentent les en-cours de l'atelier (i.e. les produits plus les ressources de transport). Les circuits ainsi obtenus sont appelés *circuits de fabrication.*

Du fait qu'un circuit élémentaire dans un graphe d'événements est un p-invariant garantit que le nombre de jetons dans les circuits de fabrication ne change pas. Donc chaque circuit de

fabrication modélise bien le fait qu'un nombre limité de ressources de transport (représenté par le nombre total de jetons qui circulent dans chaque circuit) est associé à chaque type de produit. Le taux de production de chacun de ces circuits ne dépend alors que du nombre de ressources de transport utilisées, et du temps total nécessaire à la réalisation de l'ensemble des opérations définies par la gamme de production du produit. Rappelons que le taux de production f (i.e. le nombre de produits fabriqués par unité de temps) d'un circuit élémentaire est défini par :

$$f = 1 / C$$

où C est le temps de cycle du circuit. Ceci est le taux maximal qui est obtenu en régime permanent. On sait que ce régime est atteint après un nombre fini de périodes. Soit n le nombre de circuits de fabrication, le ratio de production r_i, i = 1, 2,...,n, du produit P_i est donné par l'expression suivante :

$$r_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$$

Evidement, si chaque circuit de fabrication évolue librement, on aura très peu de chance d'obtenir les ratios de production souhaités. En plus, ces ratios dépendent du taux de production de chaque circuit et, donc, si celui-ci change (par exemple par l'augmentation du nombre de ressources de transport utilisées) on obtiendra de nouveaux ratios.

Dans le modèle tel qu'il a été défini, rien n'empêche que deux transitions qui représentent des opérations exécutées sur la même machine soient franchies en même temps, ce qui représenterait la fabrication simultanée de deux produits. Ceci n'est pas acceptable.

Pour résoudre ces deux problèmes nous allons synchroniser les circuits de fabrication à l'aide des *circuits de commande*. Ceci est réalisé de la façon suivante :

(i) on crée un circuit de commande pour chaque machine. Ce circuit passe pour toutes les transitions correspondant à des opérations exécutées sur la même machine et ne contient qu'un seul jeton. Ceci empêche le franchissement simultané de deux transitions associées à la même machine ;

(ii) pour chaque circuit de commande, l'ordre de passage dans les transitions doit respecter la séquence d'entrée définie pour la machine à laquelle correspond le circuit. Ceci permet d'obtenir les ratios souhaités pour chaque produit.

Le modèle complet de l'atelier est donné par la figure III.2.

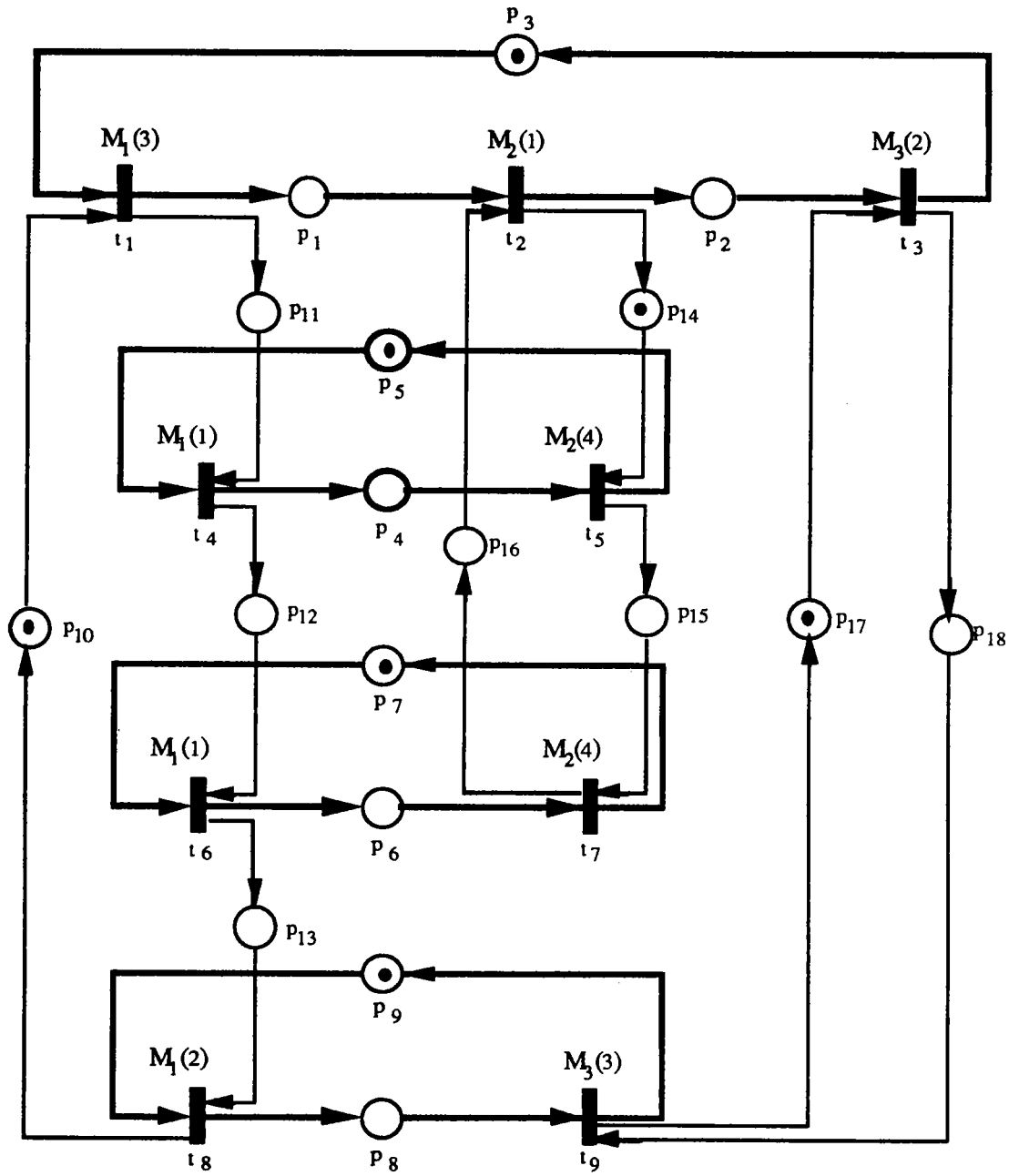


figure III.2 : Le modèle complet de l'atelier flexible

A titre d'illustration regardons comment a été construit le circuit de commande pour la machine M_1 dans le modèle présenté. La séquence d'entrée de M_1 avait été définie comme :

S_1 : (P_1, P_2, P_2, P_3).

Le circuit de commande doit passer par les transitions représentant les opération exécutées par M_1 successivement dans les gammes de P_1, P_2, P_2, et P_3. Ceci correspond au passage dans l'ordre par t_1, t_4, t_6, t_8 (ou par t_1, t_6, t_4, t_8, car les transitions t4 et t6 correspondent au même

type de produit). La position du jeton dans ce circuit indique que la machine M_1 sera initialement allouée à l'opération réalisée sur le produit P_1. Les circuits de commande de M_2 et M_3 ont été obtenus de façon identique en considérant les séquences d'entrée S_2 : (P_2, P_2, P_1) et S_3 : (P_1, P_3), respectivement.

Par l'introduction des circuits de commandes on a fait apparaître un autre type de circuit dans le modèle. Ces circuits sont les *circuits mixtes* qui sont des circuits formés en partie par de chemins appartenant à des circuits de fabrication, en partie par de chemins appartenant à des circuits de contrôle. Les circuits ci-dessous sont des exemples de circuits mixtes du réseau de la figure III.2 :

$$\gamma_1 = <t_1, p_{11}, t_4, p_4, t_5, p_{15}, t_7, p_{16}, t_2, p_2, t_3, p_3, t_1> ;$$

$$\gamma_2 = <t_1, p_1, t_2, p_2, t_3, p_{18}, t_9, p_9, t_8, p_{10}, t_1>.$$

Il est important de souligner que le détermination des circuits élémentaires ne pose aucun problème, qu'ils soient des circuits de fabrication, de commande ou mixtes, car ces circuits correspondent à des p-invariants minimaux. Il suffit donc d'utiliser une méthode de recherche de ces p-invariants pour pouvoir extraire les différents circuits.

Pour clore cette section on remarque que le modèle de l'atelier flexible que nous avons construit correspond bien à un graphe d'événements fortement connexe. On peut donc utiliser les résultats présentés dans le chapitre II pour l'analyse et l'optimisation des performances de l'atelier.

III.2.2 Modélisation d'un système d'assemblage

Dans un système d'assemblage le processus de fabrication est organisé en étapes, lesquelles regroupent une ou plusieurs opérations élémentaires pouvant être effectuées en parallèle. Ces opérations représentent soit des opérations d'assemblage (ou sous-assemblage), soit des opérations de transformation des composants. Ce processus peut donc être représenté sous la forme d'arbre, comme dans la figure III.3. Cette figure donne les gammes d'assemblage de deux produits, P_1 et P_2. La gamme du produit P_1 comporte une seule opération d'assemblage, laquelle utilise un composant du type C_1 et deux composants du type C_2. Cette opération d'assemblage est effectuée par la machine M_4. La gamme du produit P_2 comporte deux opérations d'assemblage : la première utilise un composant du type C_3 et un composant du type C_4, la deuxième utilise un composant du type C_5 et un composant du type C_6. Ces deux opérations d'assemblage sont aussi effectuées sur la machine M_4.

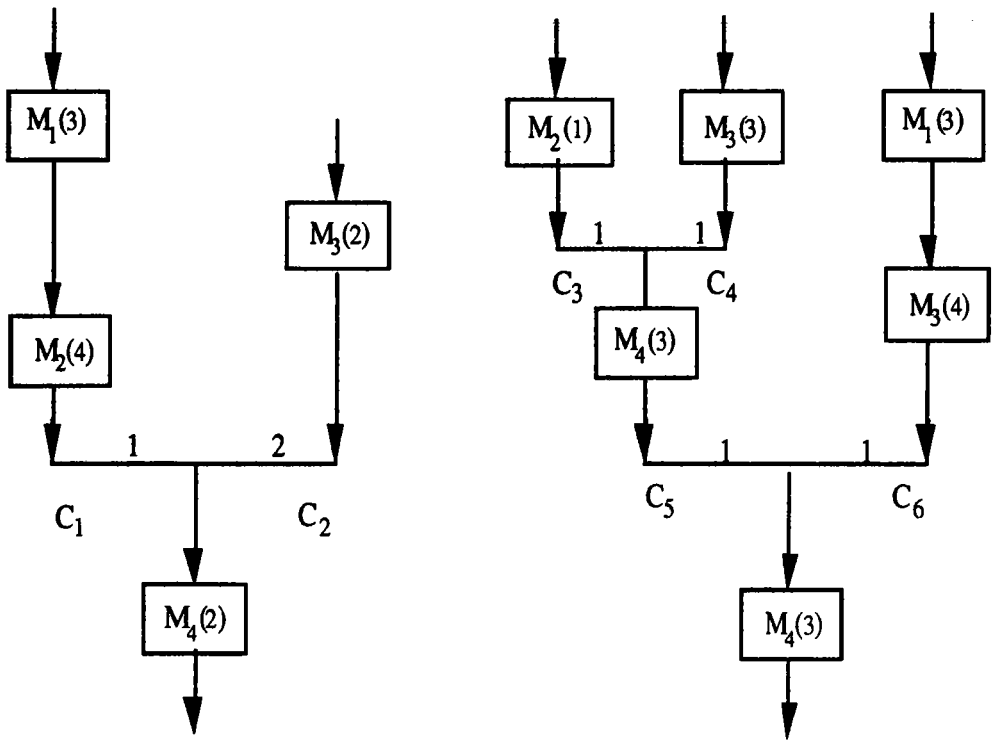


figure III.3 : Gammes d'assemblage de deux produits P_1 et P_2

Dans la suite nous supposons que les composants intervenant dans un même montage sont lancés en fabrication simultanément. Le modèle d'un produit est obtenu en répétant les séquences d'opérations correspondant à un composant autant de fois que le nombre de composants utilisés. La figure III.4 est le modèle correspondant à la gamme de produits de type P_1. Comme on peut le constater dans cette figure, la séquence d'opérations qui transforme les composants de base en composants C_2 est répétée deux fois. On notera, aussi, l'introduction d'une transition artificielle (t1) dont le but est de synchroniser le lancement de tous les composants de base nécessaires à l'assemblage du produit P_1.

Pour modéliser le fait que nous avons affaire à une production répétitive, on ajoute à ce modèle un chemin reliant la transition t_6 à la transition t_1. Cela permet de relancer la production d'un nouveau produit aussitôt que le précédent est terminé (i.e un composant C_1 et deux composants C_2).

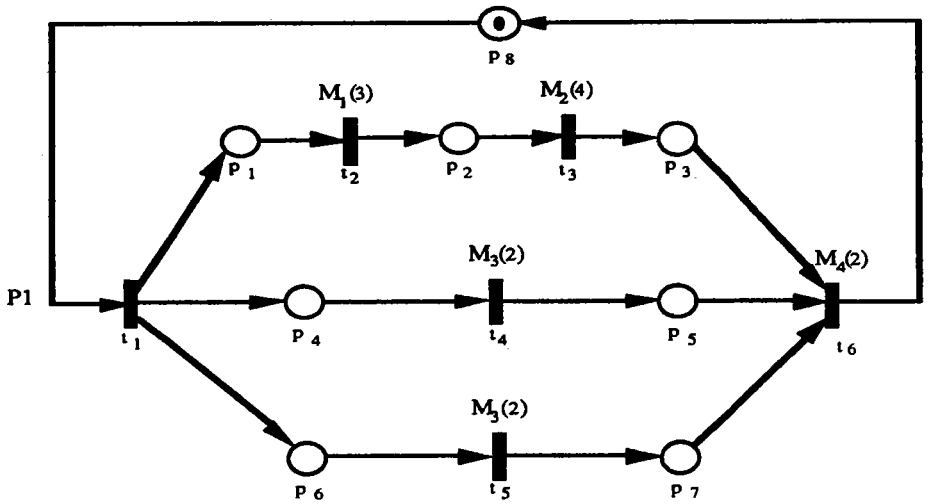


figure III.4 : Modèle de la gamme de P_1

Le modèle correspondant à la gamme de produits du type P_2 est présenté dans la figure III.5. Dans ce modèle le chemin t_{11}, p_{15}, t_8 permet de relancer l'assemblage d'un nouveau composant C_5 dès qu'un composant C_5 est terminé. De plus, le chemin t_{11}, p_{15}, t_8 permet de relancer un nouveau produit P_2. Les transitions t_7 et t_8 sont des transitions artificielles qui permettent le lancement simultané des composants nécessaires au sous-assemblage de C_5 et à l'assemblage de P_2, respectivement.

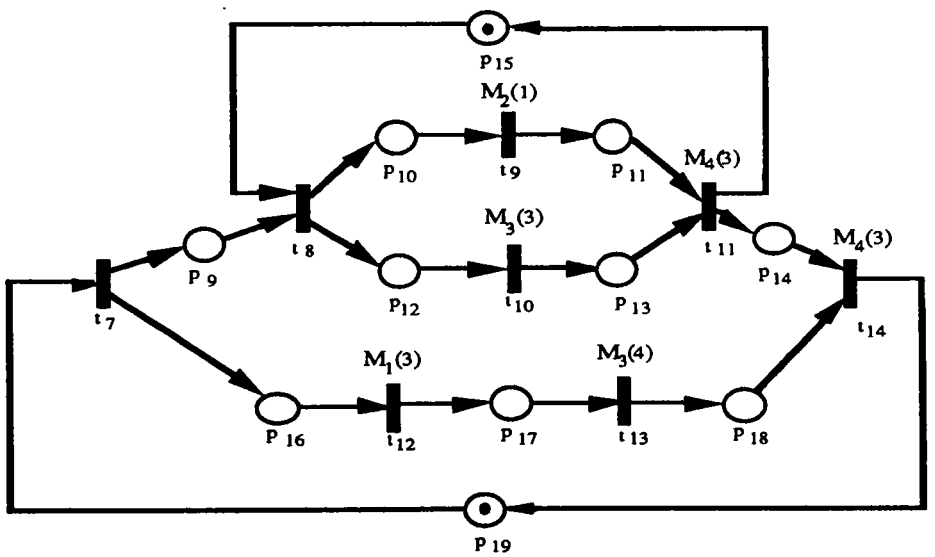


figure III.5 : Modèle de la gamme de P_2

Le processus que nous utilisons pour créer le modèle complet est le même que celui que nous avons employé dans le cas des ateliers flexibles : d'abord on définit le nombre de modèles de chaque gamme à utiliser suivant les ratios de production spécifiés. Ainsi, si le ratios de production sont 1/3 pour P_1 et 2/3 pour P_2, nous représentons une fois le modèle de P_1 et deux fois le modèle de P_2. Ensuite, nous ajoutons les circuits de commande (un circuit pour chaque

machine) pour éviter l'utilisation simultanée d'une machine pour l'exécution de plusieurs opérations. Le modèle complet est donné dans la figure III.6. Nous supposons qu'à chaque produit correspond une ratio de 50% .

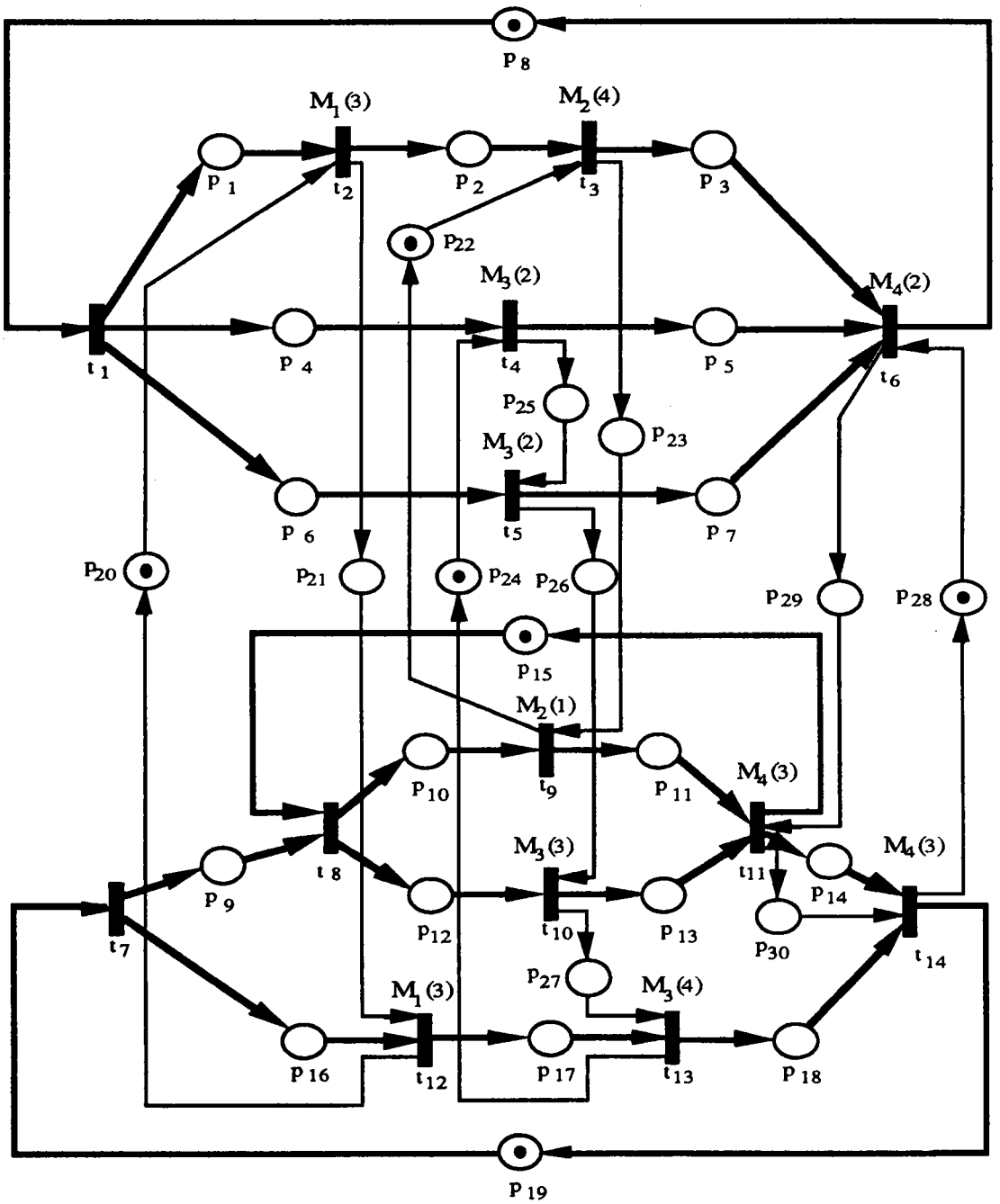


figure III.6 : Modèle complet du système d'assemblage

Les circuits de commande sont définis de telle sorte que chacun d'eux contienne toutes les transitions représentant des opérations effectuées par la même machine. Nous aboutissons encore à un graphe d'événements fortement connexe auquel s'appliquent tous les résultats du chapitre II.

III.2.3 Modélisation d'un système du type Kanban

On considère ici un système de fabrication composé de stations comportant des aires de stockage sur lesquelles s'effectuent les transformations. Les stations sont définies librement et peuvent être constituées, par exemple, d'une seule machine, d'un groupe de machines en série ou d'une cellule de fabrication.

A chaque station est affectée un ou plusieurs *Kanban*, lesquels correspondent à des étiquettes portant la référence d'un type de produit et une quantité à fabriquer. Un Kanban libre indique que le lot de produits correspondant peut être introduit dans la station. Lorsque le lot est introduit dans la station, le Kanban correspondant lui est attaché et, en conséquence, il n'est plus libre. Ce n'est qu'après que le lot aura quitté la station, que le Kanban pourra devenir libre à nouveau. La libération du Kanban se fera à l'instant où le lot aura été demandé par la station suivante. Ce processus est illustré par la figure III.7.

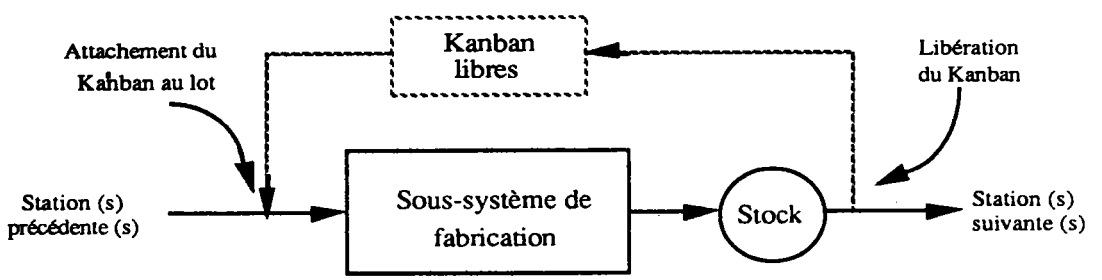


figure III.7 : Modèle d'une station

Lorsque plusieurs Kanbans sont libres en même temps dans la même station, l'ordre d'introduction des produits correspondants se fait, parmi ceux qui sont disponibles dans les stations précédentes, suivant une priorité donnée. Ceci étant, le modèle de base d'une station est celui représenté par le réseaux de Petri donné dans la figure III.8. Dans ce modèle, la transition w_1 représente l'opération réalisée dans la station. Les jetons situés dans les places p_i, $q_{i,1}$ et $q_{i,2}$ représentent les états possibles des Kanbans : ceux de la place p_i représentent les Kanban libres, ceux de la place $q_{i,1}$ représentent les Kanbans associés à de produits qui n'ont pas encore été traités, et ceux de la place $q_{i,2}$ représentent les Kanbans associés à de produits finis qui attendent le passage à la station suivante (ceci sera autorisé par la présence de jetons dans la place p_{i+1}).

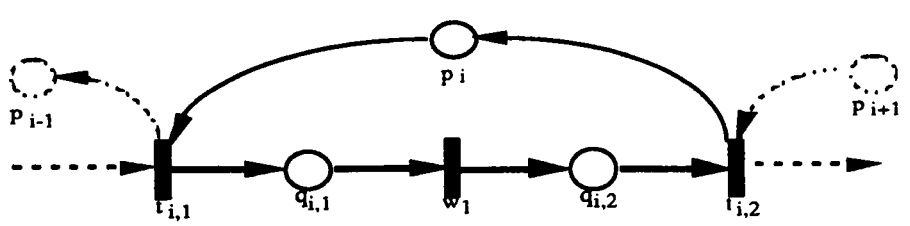


figure III.8 : Modèle d'une station

Nous pouvons maintenant considérer la modélisation d'un système du type Kanban. Ceci est fait à l'aide de l'exemple suivant : considérons un système composé de trois stations en série. Chaque station est constituée d'une seule machine. Les machines sont notées respectivement M_1, M_2 et M_3, et ne réalisent qu'une seule opération sur le produit. Ce système fabrique deux types de produits, notés P_1 et P_2, qui passent successivement dans les trois stations. Les gammes de fabrication de ces deux produits sont données par :

$P_1 : M_1(1), M_2(2), M_3(2)$; $P_2 : M_1(1), M_2(4), M_3(3)$.

Supposons que les ratios de production spécifiés correspondent à une production de 50% de chaque type de produit, et que les séquences d'entrée des machines soient identiques et correspondent à la fabrication d'un produit de type P1 et d'un produit de type P2, i.e, $S_1 = S_2 = S_3 = (P_1, P_2)$. Le modèle de ce système est donné dans la figure III.9. Dans ce modèle, on note la présence des circuits de commande lesquels sont utilisés, comme dans le cas précédents, pour éviter qu'une station soit utilisée au même temps pour effectuer deux opérations différentes. On remarque aussi que ce modèle correspond à un graphe d'événements fortement connexe.

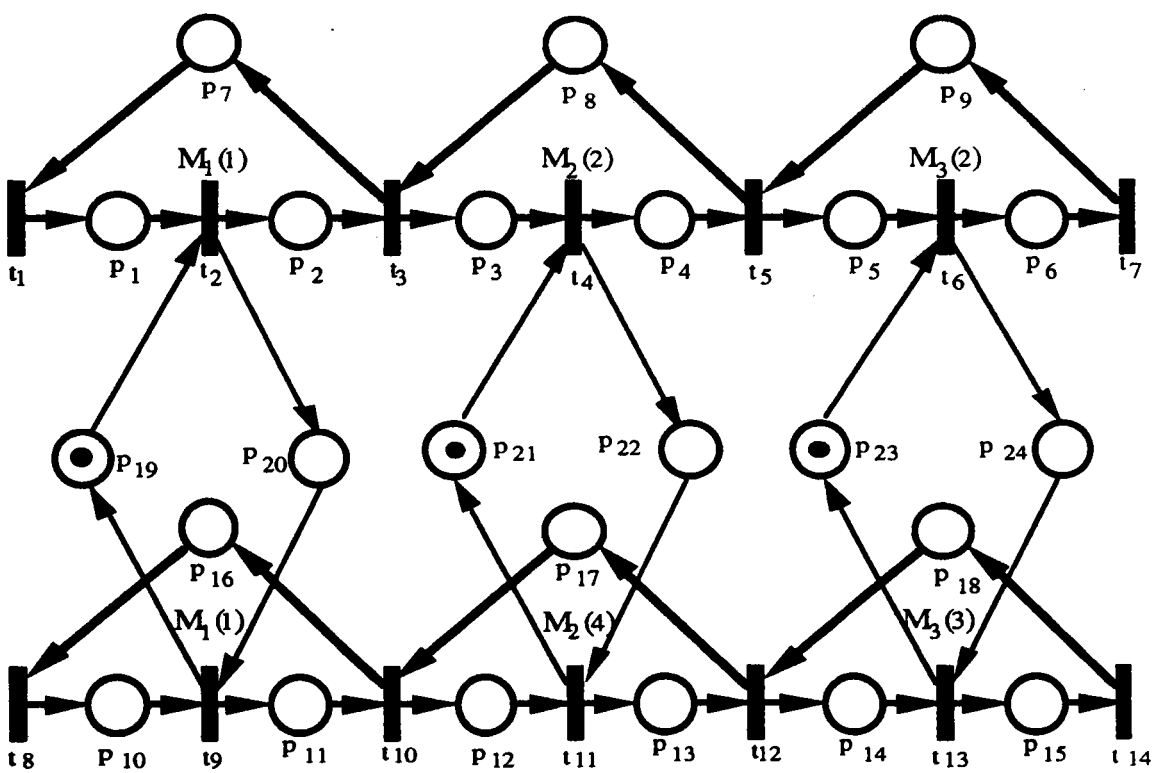


figure III.9 : Modélisation d'un système du type Kanban

III.3 Evaluation des systèmes

Cette section est consacrée à l'évaluation des systèmes que nous avons modélisés précédemment. Le problème de base que nous considérons ici est celui de l'obtention d'un temps de cycle donné avec le minimum de jetons. Les modèles obtenus étant des graphes d'événements fortement connexes, on pourra se servir des algorithmes présentés dans le chapitre II pour résoudre ce problème.

Par ailleurs nous n'allons considérer que les ateliers flexibles, car l'évaluation des systèmes d'assemblage et des systèmes du type Kanban est réalisé suivant la même démarche. Le lecteur intéressé pourra consulter les références fournies au début du chapitre pour des détails concernant l'évaluation de ces deux types de systèmes. Nous présentons le problème de l'évaluation des ateliers flexibles à l'aide de deux exemples : un pour le cas où le temps opératoires sont déterministes, l'autre pour le cas où ces temps sont aléatoires. Ces deux exemples sont issus de [Laftit 91].

III.3.1 Cas déterministe

Nous considérons ici le cas d'un atelier flexible où le temps opératoires sont déterministes, lequel est modélisé de la façon présentée dans la section III.2.1.

Il a été montré par Hillion et Proth [Hillion et Proth 89] que, dans ce cas, il est toujours possible de faire en sorte que la machine la plus chargée travaille sans arrêt, i.e, qu'il est possible de réduire le temps de cycle du modèle à celui de la machine la plus chargée. Il est clair que ce temps de cycle est plus petit temps que l'on peut obtenir. Il correspond donc à la productivité maximale.

L'objectif qu'on se fixe ici est d'atteindre la productivité maximale avec un minimum de ressources de transport (modélisés par les jetons dans les circuits de fabrication). Le critère que nous chercher à minimiser s'écrit donc :

$$f_z(M_0) = z^t \cdot M_0 = \sum_{p \in P\text{-}P_c} M_0(p)$$

où P est l'ensemble de places du graphe, P_c est l'ensemble de places appartenant à de circuits de commande et z est un p-invariant. Etant donné le modèle utilisé pour représenter l'atelier flexible, le vecteur p-invariant z est défini dans ce cas par :.

$$\begin{cases} z_i = 1 & \text{si } p_i \notin P_c \\ z_i = 0 & \text{si } p_i \in P_c \end{cases}$$

Soit C* le temps de cycle de la machine goulot. Nous savons que C* est réalisable si la deuxième (ou troisième) condition du résultat II.4 est vérifié. Nous savons aussi que chaque circuit de commande ne doit contenir qu'un seul jeton. Donc, le problème à résoudre peut être défini comme suit :

$$P1 \begin{cases} Min\ f_z(M_0) = z^t \cdot M_0 = \sum_{p \in P\text{-}P_c} M_0(p) \\ \text{s.c.} \\ \quad \dfrac{\mu(\gamma)}{M_0(\gamma)} \le C^* \qquad \forall \gamma \in \Gamma \\ \quad M_0(\gamma) = 1 \qquad \forall \gamma \in \Gamma_c \end{cases}$$

où :

Γ est l'ensemble des circuits élémentaires du graphe ;

Γ_c est l'ensemble des circuits de commande du graphe

Ce problème est un problème de type $\mathcal{P}(\alpha, \text{FPT})$ présenté dans le chapitre II (section II.4), augmenté de la contrainte qu'impose un seul jeton par circuit de commande. Pour résoudre ce problème on peut utiliser, par exemple, l'algorithme d'ajustement décrit dans le chapitre II (voir [Laftit 91] pour l'application de l'algorithme d'optimisation convexe). Il suffit pour cela de partir d'une solution initiale dans laquelle chaque circuit de commande contient un seul jeton, en faisant en sorte que cette contrainte reste respectée pendant l'exécution de l'algorithme.

Pour illustrer ce qui vient d'être présenté, nous considérons un atelier flexible composé de quatre machines M_1, M_2, M_3, M_4, lequel est en mesure de fabriquer trois produits P_1, P_2 et P_3 dont les gammes sont :

P_1 : $M_1(1)$, $M_2(1)$, $M_3(3)$, $M_4(3)$,
P_2 : $M_1(1)$, $M_4(1)$, $M_3(2)$,
P_3 : $M_1(1)$, $M_2(2)$, $M_4(1)$.

Les ratios de production souhaités pour P_1, P_2, et P_3 sont, respectivement, 25%, 25%, 50%. Les séquences d'entrée dans les quatre machines sont données par :

S_1 : (P_1, P_2, P_3, P_3),
S_2 : (P_3, P_1, P_3),
S_3 : (P_2, P_1),
S_4 : (P_1, P_2, P_3, P_3).

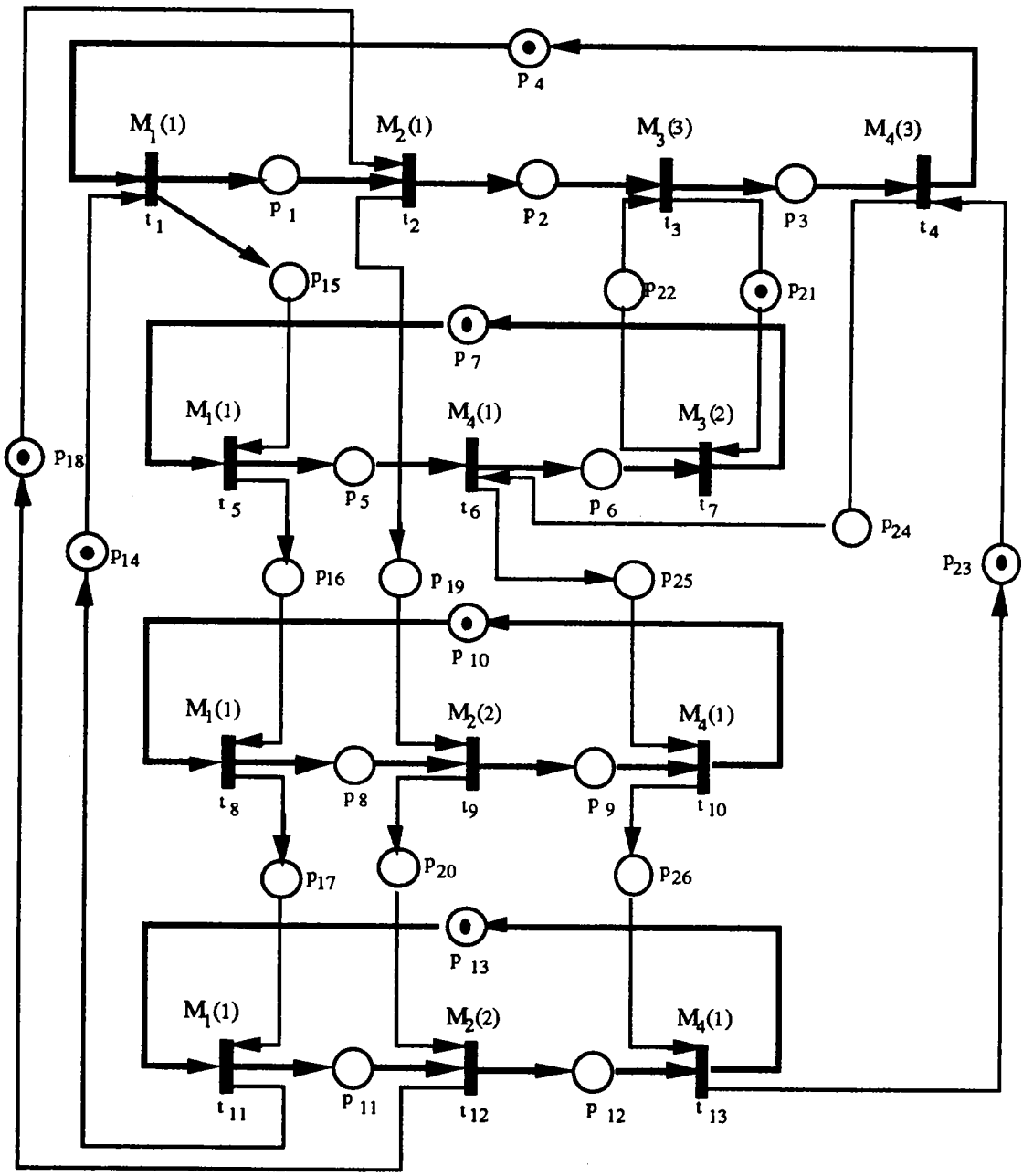


figure III.10 : Modèle dont les temps opératoires sont déterministes

Le modèle est présenté dans la figure III.10. Les temps de cycle des circuits de commande associés à M_1, M_2, M_3, M_4, sont respectivement égaux à 4, 5, 5 et 6 unités de temps. Donc la machine M_4 est la machine la plus chargée, et le temps de cycle minimum qu'on peut obtenir est $C^* = 6$ unités de temps.

Nous utilisons l'algorithme d'ajustement pour résoudre ce problème (voir section II.7.2.1 pour la description de cet algorithme). Pour cela nous partons d'un marquage initial tel que : (i) toutes les places appartenant aux circuits de fabrication sont marquées avec un jeton (places p_1 à p_{13}), et (ii) une seule place de chaque circuit de commande est marquée avec un jeton (les places choisies sont p_{14}, p_{18}, p_{21} et p_{23}). Alors le critère à minimiser s'écrit :

$$f_z(M_0) = z^t \cdot M_0 = \sum_{i=1}^{13} M_0(p_i)$$

où z est un p-invariant défini par : $\begin{cases} z_i = 1 & \text{si} \quad i = 1,2,\ldots,13 \\ z_i = 0 & \text{si} \quad i = 14,15,\ldots,26 \end{cases}$

Compte tenu du fait que dans l'algorithme d'ajustement une marque ne peut que diminuer et que les places de commande ne sont pas incluses dans le critère, on est assuré que la contrainte liée à la présence d'un seul jeton dans les circuits de commande est satisfaite. Les résultats de l'application de cet algorithme sont fournis dans la table III.1. Comme on peut le constater, un jeton est supprimé à chaque itération jusqu'à arriver au marquage minimal.

Itérations	**p***	**Places** 00000000011111111112222222 12345678901234567890123456	**Nombre total de jetons**
0		11111111111111100010010101000	17
1	7	11111101111111100010010101000	16
2	1	01111101111111100010010101000	15
3	11	01111101110111000100101000	14
4	8	01111100110111000100101000	13
5	5	01111000110111000100101000	12
6	1	01100010110110100100101000	11
7	11	01011001100101000100010100	10

table III.1 : Résultats de l'algorithme d'ajustement

Le résultat final indique que pour obtenir le temps de cycle voulu (C* = 6), le marquage initial doit contenir dix jetons dont quatre sont des jetons des circuits de commande. En regardant le marquage initial des places appartenant aux circuits de fabrication (p_1 à p_{13}), on déduit que les six ressources de transports sont distribuées de la façon suivante : deux sont affectées à P_1, une est affectée à P_2, et trois sont affectées à P_3.

III.3.2 Cas stochastique

Nous considérons maintenant le cas d'un atelier flexible où les temps de fabrication sont aléatoires. Cette hypothèse ne change en rien le modèle. Cependant, les durées des opérations ne sont plus fixées comme dans le cas précédent, mais obtenues à partir des valeurs que peuvent prendre les variables aléatoires associée à ces opérations. En conséquence, la notion de circuit critique (i.e. le circuit ayant le plus grand temps de cycle) n'a plus de sens, car on ne connait pas à priori la longueur du cycle de chaque circuit élémentaire.

Donc le problème n'est plus de saturer la machine la plus chargée mais simplement de trouver un bon compromis entre le niveau des en-cours et la productivité du système. En d'autres termes, cela signifie que nous cherchons à atteindre un temps de cycle moyen du système avec un nombre minimal de jetons. Ce problème est résolu en utilisant l'algorithme que nous avons présenté dans la section II.7.3 du chapitre II. Les deux étapes de base de cet algorithme sont : (i) trouver une solution initiale en résolvant un problème déterministe et, (ii) à partir de cette solution initiale, augmenter progressivement le nombre de jetons dans les places appartenant aux circuits de fabrication de façon à améliorer (réduire) le temps de cycle moyen du système.

Nous illustrons maintenant l'application de cet algorithme dans un exemple donné. L'exemple que nous utilisons pour cela est celui d'un atelier flexible composé de trois machines, notées M_1, M_2, M_3. Cet atelier fabrique trois types de produits notés P_1, P_2 et P_3, dont les gammes de fabrication sont :

$P_1 : M_1(X_1), M_2(X_2), M_3(X_3),$
$P_2 : M_1(X_4), M_3(X_5),$
$P_3 : M_2(X_6), M_1(X_7), M_3(X_8).$

Les variables entre parenthèses sont les variables aléatoires qui définissent les temps opératoires. Nous supposons que les produits sont fabriqués dans les mêmes proportions, et que les séquences d'entrée des machines sont données par :

$S_1 : (P_1, P_2, P_3,),$
$S_2 : (P_1, P_3),$
$S_3 : (P_1, P_2, P_3),$

Le modèle de l'atelier est donné par le réseau de Petri présenté dans la figure III.11. Le critère que nous cherchons à minimiser est le même que celui du cas déterministe, c'est-à-dire

on veut réduire le nombre de ressources de transport utilisées. Le critère s'écrit donc :

$$f_z(M_0) = z^t \cdot M_0 = \sum_{i=1}^{8} M_0(p_i)$$

où le vecteur p-invariant z est égal à (1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0).

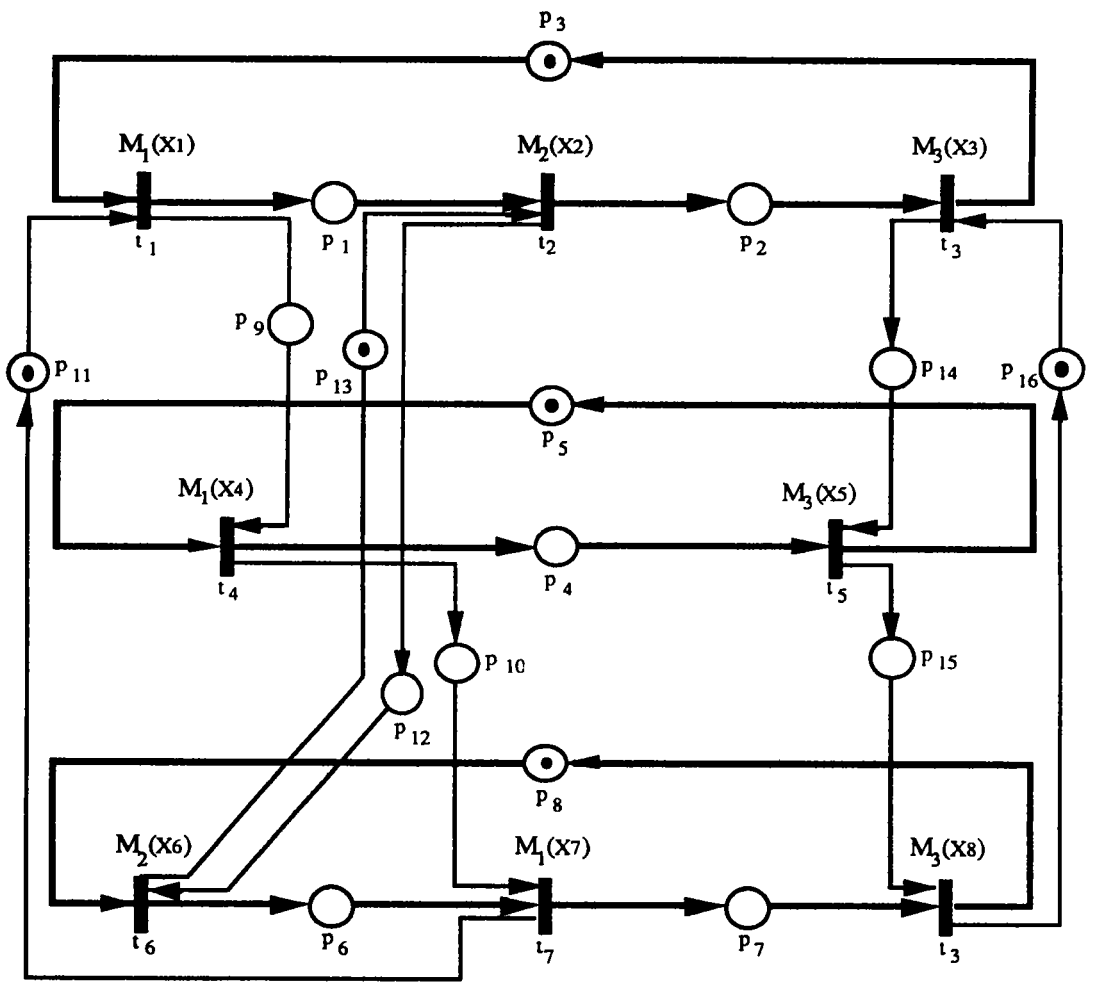


figure III.11 : Modèle d'un atelier dont les temps opératoires sont aléatoires

Comme résultat de la première phase de l'algorithme (i.e la solution du problème déterministe), nous obtenons une solution initiale pour laquelle le temps de cycle est C* = 12 unités de temps. Le marquage associé à cette solution correspondant à un jeton dans les places p_3, p_5, p_8, p_{11}, p_{13} et p_{16}. Les autres places ne sont pas marquées. Avec ce marquage, le temps de cycle moyen du problème stochastique est de 14,43 unités de temps et la valeur du critère (i.e le nombre de jetons dans p_1 à p_8) est 3.

Nous allons supposer, par exemple, que le temps de cycle moyen que nous voulons atteindre est égal à 12,1 unités de temps. Ceci sera l'objectif fixé pour l'exécution de la deuxième phase de l'algorithme. Du fait qu'un circuit de commande ne peut contenir qu'un seul jeton, seules les places des circuits de fabrication sont considérées au moment de choisir une place où ajouter un jeton. Les résultas des itérations successives de l'algorithme ainsi que les bornes inférieure et supérieure sont présentés dans la table III.2.

Itérations	Ajouter un jeton en	Borne inférieure	Cycle moyen	Borne supérieure	Valeur du critère
état initial		14,26	14,43	17,50	3
1	P_1	12,69	12,85	15,94	4
2	P_4	12,46	12,68	15,48	5
3	P_7	12,12	12,28	14,25	6
4	P_8	12,05	12,08	13,01	7

table III.2 : Résultats de la deuxième phase de l'algorithme

Donc, à partir du nouveau marquage du réseau on peut conclure que pour obtenir le temps de cycle moyen spécifié, nous devons utiliser sept ressources de transport : deux sont affectées à P_1, deux sont affectées à P_2, et trois sont affectées à P_3.

III.4 Conclusion

Dans ce chapitre nous avons considéré les problèmes de modélisation et d'optimisation des performances de quelques systèmes de production fonctionnant sous l'hypothèse d'une production répétitive (cyclique). En particulier, nous avons montré comment modéliser les ateliers flexibles, les systèmes d'assemblage et les systèmes du type Kanban. Les modèles auxquels nous avons abouti sont des graphes d'événements fortement connexes, ce qui a permis l'utilisation des résultats et des algorithmes développés dans le chapitre II pour l'évaluation de ces modèles. Le problème considéré dans l'évaluation est celui de l'obtention d'un temps de cycle spécifié avec l'utilisation d'un nombre minimal de jetons dans les circuits de fabrication. La résolution de ce problème a été illustrée à l'aide de deux exemples d'ateliers flexibles : l'un pour le cas déterministe, l'autre pour le cas stochastique. Dans le chapitre qui suit nous allons considérer la modélisation et l'évaluation des systèmes de production non-cycliques.

Chapitre IV

Systèmes de production non-cycliques

IV.1 Introduction

Dans ce chapitre nous étudions les systèmes de production non-cycliques à l'aide des réseaux de Petri. Cette étude passe par la modélisation du système, puis par l'évaluation des performances du modèle obtenu. Comme nous l'avons souligné dans le chapitre II, on a affaire ici à des systèmes où les ratios de production des différents types de produits varient avec le temps. En conséquence, la modélisation par des graphes d'événements telle que nous l'avons développée dans le chapitre précédent ne peut plus être appliquée. Ceci nous amène donc à utiliser une nouvelle classe de réseaux de Petri qui est plus générale que les graphes d'événements. L'introduction de contraintes dans l'évolution du modèle obtenu permettra de garantir que le modèle présente toutes les propriétés intéressantes du point de vue des systèmes de production.

Pour modéliser les systèmes non-cycliques nous utilisons les RPES (Réseaux de Petri avec des transitions d'Entrée et des transitions de Sortie) que nous avons étudiés dans le chapitre II. Dans un premier temps le modèle que nous considérons est celui dans lequel on associe un RPES à la gamme de fabrication de chaque type de produit. Ce modèle est utilisé au niveau de la planification au cours de laquelle aucune contrainte concernant l'utilisation des ressources (machines) n'est considérée. Par la suite, nous enrichirons le modèle de base par l'inclusion d'une place pour chaque ressource, afin d'éviter l'utilisation simultanée de la même ressource pour plusieurs opérations (cette place est une place d'entrée et de sortie de chaque transition représentant l'opération qui utilise cette ressource). On obtiendra ainsi le modèle que nous utilisons pour le calcul de l'ordonnancement des opérations à effectuer au cours de la première période élémentaire telles qu'elles ont été déterminées au niveau de la planification.

L'évaluation du système est un processus itératif dans lequel on alterne :

1 - la résolution d'un problème de planification soumis aux seuls contraintes de capacité. De plus, on cherche la solution qui optimise un certain critère de fonctionnement. Ce problème est résolu, dans notre cas, par la programmation linéaire lors que le critère consiste à minimiser les coûts de stockage et de rupture. Le résultat de cette étape correspond à ce qu'on appelle le *plan de production*, lequel spécifie le nombre de fois que chaque opération doit être réalisée durant chaque période élémentaire.

2 - la résolution d'un problème d'ordonnancement qui réalise le plan de production fixé dans l'étape précédente. Nous disposons pour cela de deux algorithmes : le premier fournit la solution exacte, mais est limité à des problèmes de petite taille, et le deuxième est une heuristique qui ne garantit pas la solution optimale. Nous nous contentons d'appliquer ces algorithmes afin de trouver une solution réalisable, c'est-à-dire une solution pour laquelle la durée totale de production dans chaque période est inférieure au égale à la durée de la période.

Dans le premier paragraphe nous montrons, essentiellement à l'aide d'un exemple, comment modéliser un système de production non-cyclique. Le problème de la décomposition du modèle est aussi considéré dans cette partie. Dans le deuxième paragraphe, le problème de planification est spécifié et mis sous la forme d'un problème de programmation linéaire en nombres entiers si le critère à optimiser correspond à la minimisation des coûts de stockage et de rupture, ou sous la forme d'un problème de programmation quadratique si le critère consiste à équilibrer les charges des machines. Le troisième paragraphe introduit le problème d'ordonnancement et présente les deux algorithmes dont nous disposons pour résoudre ce problème. Une étude comparative permettra de déterminer la meilleure utilisation de chacun d'eux. Enfin, nous présentons un exemple numérique.

IV.2 Modélisation d'un système de production non-cyclique

IV.2.1 Définition du modèle

L'approche que nous utilisons pour modéliser un système de production non-cyclique est semblable à celle que nous avons utilisée pour la modélisation des systèmes cycliques : on modélise d'abord les gammes de fabrication de chaque produit et on ajoute ensuite un mécanisme pour empêcher l'utilisation simultanée d'une même ressource pour effectuer plusieurs opérations.

Néanmoins, quelques différences importantes entre les deux types de modèles sont à

noter. La première est que les gammes de fabrication qu'on considère ici décrivent tous les routages qui peuvent être utilisés dans la fabrication d'un produit, contrairement au cas cyclique où il n'y avait qu'un seul routage possible. Cela entraîne la présence dans les modèles réseaux de Petri correspondants à ces gammes, de places ayant plusieurs transitions d'entrée et plusieurs transitions de sortie. En conséquence, ces gammes ne peuvent plus être modélisées par des graphes d'événements et nous devons utiliser les réseaux de Petri généraux (en l'occurrence les RPES). Une autre différence est que les modèles correspondant aux gammes de fabrication des produits ne sont représentés qu'une seule fois dans le modèle global du système. Rappelons que dans le cas cyclique chacun de ces modèles peut être représenté plus d'une fois en fonction du ratio de fabrication de chaque produit. Enfin, nous modélisons le lancement de la fabrication d'un produit et la sortie d'un produit fini ou semi-fini du système, par des transitions d'entrée et des transitions de sortie, respectivement. On n'a donc plus besoin de connecter les transitions représentant la dernière et la première opération de la gamme de fabrication par l'intermédiaire d'une place de bouclage, comme cela a été fait dans le cas cyclique pour modéliser le lancement de la fabrication d'un nouveau produit.

Nous ferons aussi l'hypothèse que les opérations sont non-préemptives. De plus, nous introduisons une boucle pour chaque transition avec un jeton dans la place appartenant à cette boucle, afin d'empêcher le franchissement d'une transition si celle-ci est en cours de franchissement. Nous ne représentons pas ces boucles pour ne pas surchager les modèles.

Les boucles élémentaires sont les seuls circuits existant dans ces modèles. Ces boucles sont de deux types : celles décrites ci-dessus et celles obtenues par l'inclusion des places ressources que nous allons définir dans la suite. Cependant, nous savons que toute place marquée appartenant à une boucle élémentaire n'est qu'une place implicite ([DiCesari *et al.* 93], [Murata 89]), i.e. une place dont la suppression ne change pas le comportement du modèle. Donc, du point de vue de la vérification des propriétés, les modèles que nous obtenons peuvent toujours être considérés comme étant des réseaux de Petri sans circuits.

Nous illustrons le processus de modélisation d'un système non-cyclique à l'aide d'un petit exemple. Le système considéré est composé de quatre machines (M_1, M_2, M_3 et M_4) et est en mesure de fabriquer deux types de produits (P_1 et P_2), dont les gammes de fabrication sont représentées dans la figure IV.1. Dans cette figure, chaque rectangle représente une opération et contient la liste des machines qui peuvent effectuer cette opération. Par exemple, la première opération de la fabrication du produit P_1 peut être effectuée soit par la machine M_2 soit par la machine M_1.

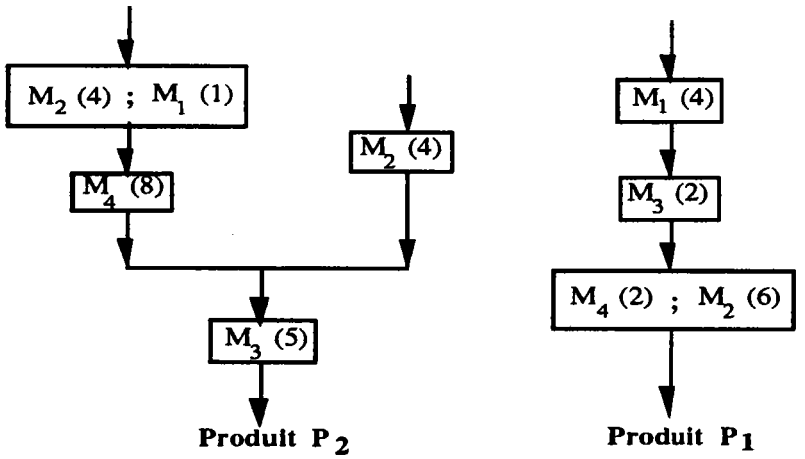


figure IV.1 : Gammes de fabrication des produits P_1 et P_2

La première étape de la modélisation consiste donc à représenter la gamme de fabrication de chaque produit par un RPES. Le modèle obtenu est celui donné par la figure IV.2. Dans ce modèle chaque transition représente une opération élémentaire effectuée sur une machine donnée, à l'exception des transitions d'entrée (t_0, t_4 et t_8) et de sortie (t_7, t_{12} et t_{14}), lesquelles n'utilisent aucune machine et sont temporisées à zéro. Ces transitions ne font que représenter le lancement de la fabrication des produits ou composants et la sortie des produits du système, respectivement.

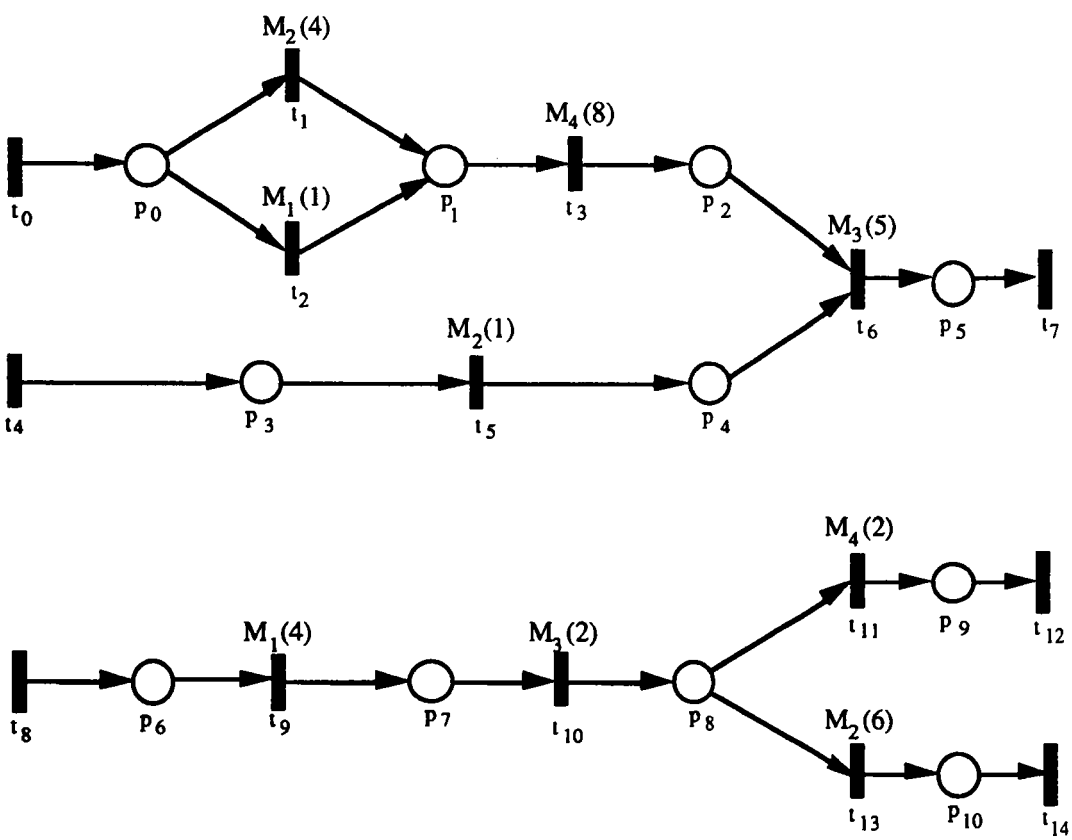


figure IV.2 : Modèle partiel du système

Comme nous l'avons déjà souligné dans l'introduction de ce chapitre, le modèle de la figure IV.2 sera utilisé pour la détermination du plan de production à court terme. En effet, pour la détermination de ce plan, on présuppose que les ressources nécessaires à la réalisation de chacune des opérations élémentaires définies dans le modèle sont disponibles à tout instant. Ce modèle représente bien cela, car aucune contrainte n'existe dans le modèle qui interdise l'utilisation simultanée d'une ressource donnée pour exécuter plus d'une opération.

Une conséquence de la non prise en compte des conflits d'utilisation des ressources dans la détermination du plan de production, est que celui-ci peut ne pas être réalisable au niveau de l'ordonnancement (i.e. la durée totale de fabrication des produits peut être supérieure à la durée maximale permise pour cela, c'est-à-dire à la durée totale de l'horizon de planification). La vérification de la faisabilité du plan correspond à l'étape suivante dans la démarche utilisée pour l'évaluation du système qui a été modélisé. Pour permettre cela nous allons d'abord doter le modèle précédent d'un mécanisme qui supprime les conflits relatifs à l'utilisation de ces ressources. Nous soulignons qu'il ne s'agit plus de définir des circuits de commande comme c'était le cas des systèmes cycliques, car on ne cherche pas à fixer les ratios de fabrication des produits. Il s'agit seulement de prévenir l'utilisation simultanée d'une même ressource pour plusieurs opérations. Donc nous ajoutons dans le modèle une place pour chaque ressource. Cette place ne contient qu'un seul jeton et elle est à la fois place d'entrée et place de sortie de toutes les transitions qui représentent une opération effectuée par la ressource. Par la suite nous appellons cette place place ressource.

Le modèle complet obtenu par l'inclusion des places ressources dans le modèle présenté dans la figure IV.2 est schématisé dans la figure IV.3. On notera que l'introduction des places ressources a créé un certain nombre de boucles élémentaires. Chacune de ces boucles est formée par une place ressource et par une des transitions d'entrée/sortie de cette place.

Comme nous avons vu dans la section II.8.3 du chapitre II, les modèles utilisés pour la planification sont ceux qui peuvent être décomposés en un ensemble de RPES sans conflits. Les résultats que nous avons établis montrent que le fait d'avoir un modèle décomposable dans lequel il n'y a pas de circuits, permet de garantir que le modèle a toutes les propriétés qualitatives intéressantes du point de vue des systèmes de production. Il ne nous reste donc qu'à vérifier si un modèle est décomposable. Ceci fait l'objet de la section suivante.

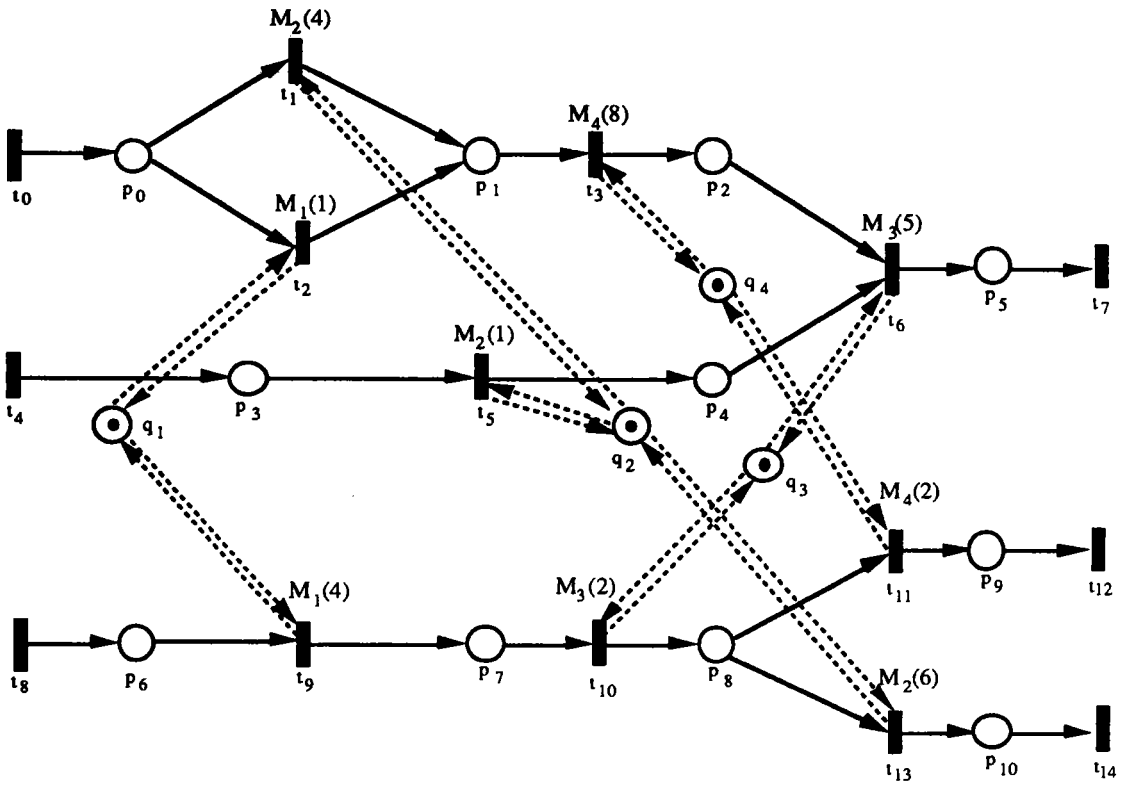


figure IV.3 : Modèle complet

IV.2.2 Décomposition du modèle

La vérification de la décomposabilité du modèle est un processus itératif dans lequel on considère successivement chacun des RPES qui forment le modèle (rappelons que dans le modèle que nous avons défini antérieurement on modélise la gamme de fabrication de chaque produit par un RPES). A chaque itération on considère un seul RPES, et on réalise les opérations suivantes :

(i) on extrait tous les sous-réseaux qui sont des RPES sans conflit de ce RPES ;

(ii) on élimine tous les RPES sans conflit qui ne sont pas consistants ;

iii) si l'ensemble des RPES sans conflit qui n'ont pas été éliminés dans l'étape précédente couvre le RPES, alors on passe au RPES suivant. Dans le cas contraire, on dira que le modèle n'est pas décomposable.

Si tous les RPES du modèle sont couverts par des RPES sans conflit, alors le modèle est décomposable. Par la suite, les RPES sans conflits que nous considérons sont ceux qu'on appelle les RPES maximaux sans conflits, dont la définition est donnée ci-dessous.

Définition IV.1

Soit NC un RPES sans conflit extrait d'un RPES N. Nous dirons que NC est maximal si, quel que soit le RPES sans conflit NC extrait de N, tel que NC* ≠ N, NC* ne contient pas*

NC.

Nous proposons maintenant un algorithme qui permet d'extraire un RPES maximal sans conflit d'un RPES N. Cet algorithme procède par itérations successives. A chaque itération, le but est de supprimer un certain nombre de transitions afin de supprimer un conflit du réseau.

Algorithme IV.1 : extraction d'un RPES maximal sans conflits

1 - Tester s'il existe une place $p \in N$ telle que p° possède plus d'une transition de sortie.

1.1 - Si aucune place ne vérifie cette condition, alors fin de l'algorithme.

1.2 - Dans le cas contraire, faire :

1.2.1 - Supprimer toutes les transitions de sortie de p, à l'exception d'une. Supprimer aussi les arcs arrivant et sortant de ces transition.

1.2.2 - Supprimer itérativement toutes les places sources et toutes les places puits, ainsi que leurs transitions d'entrée et de sortie et les arcs correspondants.

1.2.3 - Aller en 1.

Il est clair que pour pouvoir extraire tous les RPES sans conflit maximaux, nous devons relancer l'algorithme IV.1 en faisant à chaque fois un choix adéquat des transitions à supprimer dans l'étape 1.2.1. Il est important de remarquer que ce choix n'est pas complètement libre, car on doit prendre en compte les choix réalisés antérieurement. Par exemple, si deux places satisfaisant la condition de l'étape 1 de l'algorithme ont une transition en commun, alors si cette transition n'a pas été supprimée au niveau de la première place elle ne doit pas l'être au niveau de la deuxième. Pour faciliter l'application de cet algorithme, nous avons mis en place une procédure qui détermine d'avance tous les ensembles de transitions qui doivent être supprimées dans chaque utilisation de l'algorithme.

Le résultat suivant montre que si le réseau obtenu par l'application de l'algorithme IV.1 est un RPES, alors c'est un RPES maximal sans conflit.

Résultat IV.1

Soit un RPES N et soit NC le réseau résultant de l'application de l'algorithme IV.1 sur N. Si NC n'est pas vide, est connexe et possède au moins une transition d'entrée et une transition de sortie du réseau original, alors NC est un RPES maximal sans conflit de N.

Preuve :

a - A partir du pas 1.2.1 de l'algorithme IV.1 nous pouvons conclure que le réseau NC est sans conflit.

b - Soit t une transition qui n'est ni une transition d'entrée ni une transition de sortie de N. Si la transition t est une transition de sortie (resp. d'entrée) du réseau NC, ceci signifie que les places d'entrée (resp. de sortie) de la transition t ont été supprimées dans l'application de l'algorithme IV.1. Cependant, d'accord avec le pas 1.2.2 de l'algorithme, la transition t doit elle aussi être supprimée. Donc, la transition t ne peut être ni une transition de sortie ni une transition d'entrée de NC et, en conséquence, les transitions d'entrée (resp. de sortie) de NC sont des transitions d'entrée (resp. de sortie) de N.

c - Supposons qu'il existe un RPES connexe sans conflit NC*, tel que $N \supseteq NC^* \supset NC$. Alors, au moins une des quatre situations suivantes est vérifiée.

c_1 - Il existe une place $p \in NC$ et une transition $t \in NC^*\text{-}NC$ tels que $(p,t) \in F^*$, où F^* est l'ensemble des arcs de NC*. Ce cas est présenté dans la figure IV.4.a. Du fait que NC est un RPES connexe sans conflit, il existe une transition $t^* \in NC$ telle que (p,t^*) est un arc de NC. En conséquence, (p,t^*) est aussi un arc de NC*. Donc, la place p possède deux transitions de sortie, et donc NC* n'est pas un RPES sans conflit.

c_2 - Il existe une place $p \in NC$ et une transition $t \in NC^*\text{-}NC$ tels que $(t,p) \in F^*$. Ce cas est illustré dans la figure IV.4.b. Dans ce cas, l'obtention du réseau NC implique que nous devons supprimer la transition t. Mais dans l'application de l'algorithme IV.1 la suppression de la transition t n'est possible que si une de ses places d'entrée (sortie) est devenue une place source (puits). Le fait que cette place est une place source (puits) implique que ses transitions d'entrée (sortie) ont été supprimées. Ce processus peut continuer jusqu'à ce qu'on trouve que les transitions d'entrée (sortie) d'une place puits (source) correspondent à des transitions de sortie des places sujettes à conflits qui ont été supprimées au moment de la résolution du conflit associé à ces places. Par conséquent, pour avoir la transition t dans NC* nous devons maintenir quelques conflits, et NC* ne serait pas un RPES sans conflit.

c_3 - Il existe une transition $t \in NC$ et une place $p \in NC^*\text{-}NC$ tels que $(p,t) \in F^*$. Ceci est illustré dans la figure IV.4.c. Cette situation ne peut pas apparaître dans l'application de l'algorithme IV.1, car l'obtention du réseau NC implique que nous devons supprimer la place p. Mais dans ce cas, la suppression de la place p n'est possible que si celle-ci était une place source, ce qui, en conséquence, provoquerait aussi la suppression de la transition t. Ceci est contradictoire avec le fait que la transition t appartient à NC.

c_4 - Il existe une transition $t \in NC$ et une place $p \in NC^*\text{-}NC$ tels que $(t,p) \in F^*$. La

figure IV.4.d présente ce cas. De la même façon que dans le cas précédent, nous affirmons que cette situation ne peut pas apparaître dans l'application de l'algorithme IV.1, car l'obtention du réseau NC implique que nous devons supprimer la place p. La suppression de la place p n'est possible que si celle-ci a était une place puits, ce qui, en conséquence, provoquerait aussi la suppression de la transition t. Ceci est contradictoire avec le fait que la transition t appartient à NC.

Q.E.D.

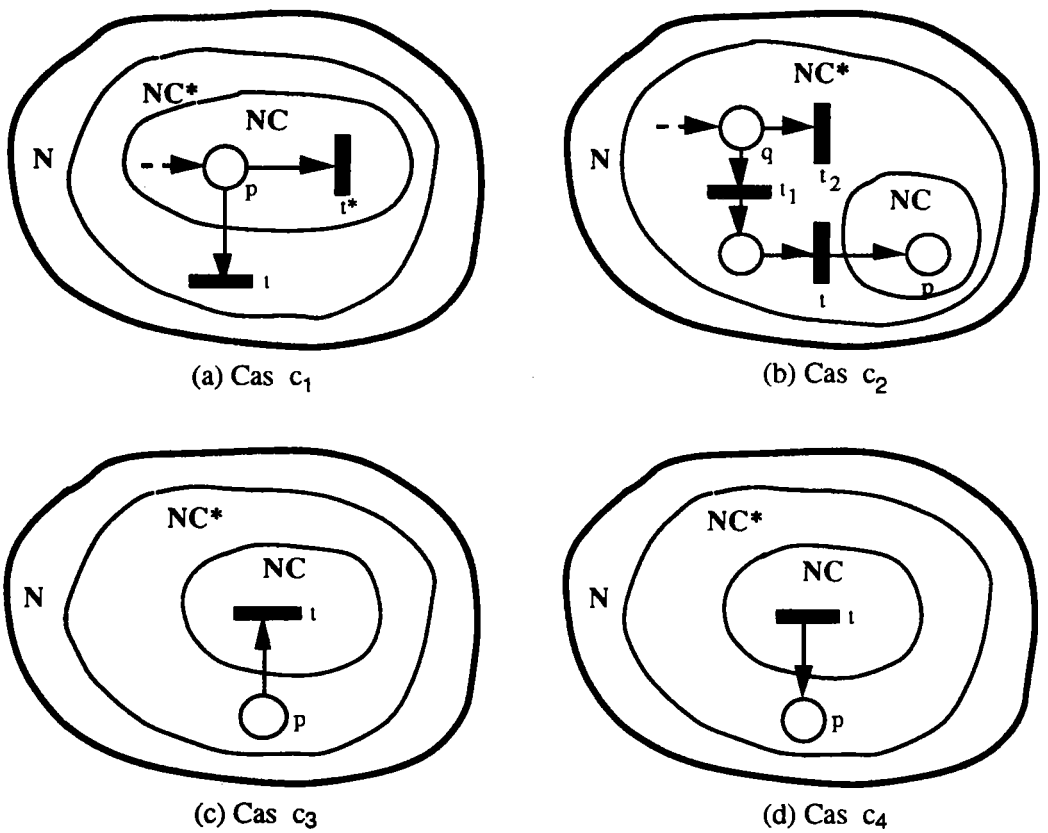


figure IV.4 : Support pour la démonstration du résultat IV.1

Pour illustrer l'application de cet algorithme nous considérons l'exemple présenté dans la figure IV.5 ci-dessous.

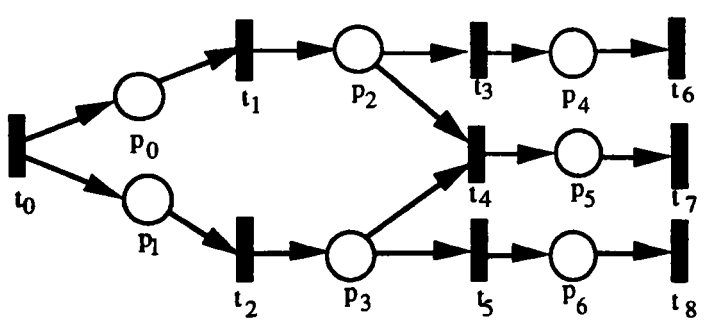


figure IV.5 : Un RPES

Ce RPES possède un conflit au niveau de la place p_2 et un autre au niveau de la place p_3. Etant donné que ces conflits sont couplés nous n'avons que deux possibilités de suppression dans ce cas : soit supprimer les transitions t_3 et t_5, soit supprimer la transition t_4. On considère ce deux cas :

a - On commence par supprimer la transition t_3 (et ses arcs d'arrivé et de sortie). La place p_4 devient donc une place source, ce qui nous amène à supprimer cette place et ensuite la transition t_6. De façon identique, la suppression de la transition t_5 provoque la suppression de la place p_6 et de la transition t_8. Comme il n'y a plus de conflit dans le réseau l'algorithme s'arrête, et on obtient le RPES sans conflit maximal donné dans la figure IV.6.a.

b - par le même raisonnement, avec la suppression de la transition t_4 on obtient le RPES sans conflit maximal donné dans la figure IV.6.b.

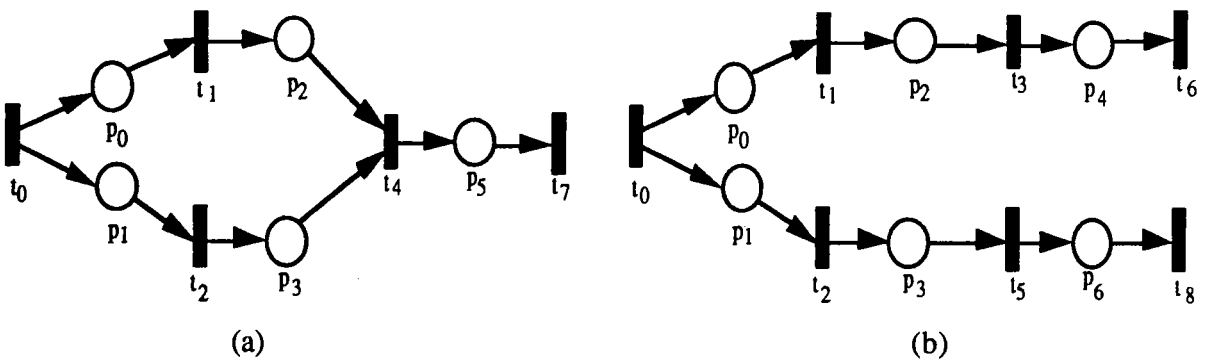


figure IV.6 : Les RPES sans conflit maximaux pour l'exemple de la figure IV.5

Jusqu'à présent nous avons montré comment obtenir les RPES maximaux sans conflit. Or, notre objectif étant d'avoir la plus grande flexibilité opératoire du système, nous souhaitons utiliser les sous-réseaux les plus petits qu'on puisse obtenir, i.e. ceux qui correspondent aux t-invariants à supports minimaux. Pour cela, nous présentons maintenant un algorithme qui permet de décomposer chaque RPES maximal sans conflit en un ensemble de sous-réseaux qui nous intéressent.

Le fonctionnement de cet algorithme est basé sur une recherche de chemins à partir des transitions de sortie du sous-réseau. Deux règles de base sont utilisées : pour chaque transition on explore tous les chemins arrivant ou sortant de la transition, et pour chaque place on n'explore qu'un seul chemin à chaque étape. Dans le cas des places, ce chemin est choisi de façon à générer toutes les possibilités de recherche, en ne modifiant à chaque étape que le choix relatif à une seule place. A la fin d'une étape on obtient un sous-réseau qui peut correspondre ou non à un t-invariant. Le résultat IV.3 donne les conditions qui doivent être vérifiées afin de pouvoir conclure si les transitions du sous-réseau correspondent au support d'un t-invariant à support minimal ou non. La démonstration de ce résultat est basée sur le résultat suivant qui a

été démontré dans [Martinez et Silva 85].

Résultat IV.2

Soit N un réseau de Petri et soit C^{tr} la transposée de la matrice d'incidence de N. Un t-invariant $Y = (y_1, y_2,\ldots,y_q)^{tr}$, $y_i > 0$ pour tout $i = 1, 2,\ldots,q$, est à support minimal si et seulement si $q = rang(C^{tr}) + 1$.

Résultat IV.3

Soit N_X le sous-réseau obtenu à la fin de chaque étape de l'algorithme et soit C^{tr} la transposée de sa matrice d'incidence. Les transitions de N_X sont le support d'un t-invariant à support minimal si et seulement si les deux conditions ci-dessous sont vérifiées :

(i) $C_X^{tr} \cdot y = 0; \qquad y > 0$
(ii) rang $(C_X^{tr}) = |T_X| - 1$

Preuve :

La condition (i) implique que N_X est consistent et, par conséquent, qu'il existe un t-invariant dont le support correspond aux transitions de N_X. La condition (ii) est une conséquence immédiate du résultat IV.2. Donc, si les deux conditions sont vérifiées les transitions de N_X sont le support minimal d'un t-invariant.

Q.E.D.

Cet algorithme est présenté dans la suite. Dans celui-ci nous utilisons trois listes appellées respectivement NŒUDS, SR et PLEX. La liste NŒUDS contient les éléments à partir desquels on doit poursuivre la recherche, la liste SR contient les éléments déjà visités, et la liste PLEX contient les places à partir desquelles on doit tester d'autres chemins. Ces listes sont initialement vides.

Algorithme IV.2 : Obtention des RPES minimaux sans conflit

1 - Pour toutes les transitions de sortie To faire :
  1.1 - Pour chaque place du modèle ayant plus d'une transition d'entrée, fixer l'une de ses transitions d'entrée comme le prochain élément à explorer ;
  1.2 - Placer To dans les listes NŒUDS et SR ;
  1.3 - Tant que la liste NŒUDS n'est pas vide :
    1.3.1 - Prendre un élément de la liste NŒUDS. Noter EL cet élément ;
    1.3.2 - Si EL est une transition faire :
      1.3.2.1 - Inclure toutes les places d'entrée et de sortie de EL dans la liste SR si elle n'y figure pas encore, et dans la liste NŒUDS si elle

n'apparaît dans aucune des deux listes.;

1.3.2.2 - Aller en 1.3.4 ;

1.3.3 - Si EL est une place faire :

1.3.3.1 - Si le prédécesseur de EL dans le chemin est une de ses transitions d'entrée, alors choisir sa transition de sortie comme élément suivant. Sinon :

Si EL possède une seule transition d'entrée, choisir cette transition. Dans le cas contraire, choisir la transition d'entrée qui correspond au choix fixé pour cette place et mettre EL à la fin de la liste PLEX si elle n'y figure pas encore.

Appeler t la transition choisie à cette étape ;

1.3.3.2 - Inclure la transition t dans la liste SR si elle n'y figure pas encore, et dans la liste NŒUDS si elle n'apparaît dans aucune des deux listes.;

1.3.4 - Supprimer EL de la liste NŒUDS ;

1.4 - Vérifier si les transitions de SR sont le support minimal d'un t-invariant ;

1.5 - Tant qu'un choix n'a pas été fait ou que la liste PLEX n'est pas vide faire :

1.5.1 - Prendre la dernière place de la liste PLEX ;

1.5.2 - Supprimer la place de la liste PLEX si toutes ses transitions d'entrée ont été explorées. Sinon, modifier le choix du prochain élément à explorer à partir de cette place et aller en 1.2 ;

1.6 - Aller en 1 ;

2 - Fin

Nous ferons deux remarques concernant cet algorithme. D'abord, dans la détermination d'un sous-réseau (i.e. dans le pas 1.2) on ne poursuit pas l'exploration à partir d'un élément qui a été déjà rencontré auparavant, ce qui évite tout problème de bouclage et garantit que l'algorithme s'arrête. Deuxièmement, étant donnée la manière dont la modification du choix du chemin à explorer est fait en 1.4.2, nous pouvons obtenir dans certains cas, un sous-réseau identique à un réseau obtenu antérieurement. Ceci est le cas, par exemple, quand il existe un chemin reliant deux transitions d'entrée d'une place qui a été mise dans la liste PLEX. Par conséquent, nous devons comparer chaque nouveau sous-réseau à ceux obtenus au cours des étapes précédentes.

IV.3 Planification à court terme

Dans la section précédente nous avons montré comment modéliser un système de production non-cyclique par des RPES. Nous considérons maintenant la première étape de

l'évaluation de ce système, laquelle correspond à la résolution du problème de planification à court terme. Dans cette étape, nous supposons que les quantités de chaque type de produit à fabriquer à la fin de chaque période élémentaire sont connues. L'objectif est donc de définir quelles sont les opérations à réaliser sur chacun des produits et le nombre de fois que ces opérations sont effectuées au cours de chaque période élémentaire afin de satisfaire les demandes en optimisant un certain critère de fonctionnement. Rappelons qu'une opération est un couple constitué d'un produit et d'une machine choisie pour effectuer une transformation donnée, et que dans le modèle dont nous disposons la fabrication de chaque type de produits peut se faire suivant des routages alternatifs. Plusieurs critères d'optimisation peuvent être considérés Dans notre cas, le critère peut être la minimisation de la somme totale des coûts de stockage et de rupture des produits, ou la minimisation de la différence des charges des machines. Il est important de souligner que l'utilisation de l'un ou l'autre de ces critères peut modifier de façon significative la durée totale de production.

Nous avons vu dans le chapitre II que les systèmes dont les modèles sont décomposables peuvent être contrôlés de façon à préserver les bonnes propriétés qualitatives. Donc nous ne considérons ici que ces types de systèmes. Dans les section qui suivent nous montrons comment résoudre le problème de planification en utilisant les sous-réseaux extraits du modèle (voir les sections précédentes). Les sous-réseaux que nous utilisons sont ceux qui correspondent à des t-invariants à support minimal.

IV.3.1 Définitions et notations

Le problème planification consiste à organiser la production de n produits sur un horizon de planification H composé de Q périodes élémentaires. Nous considérons que chaque période élémentaire i a une durée fixe de u^* unités de temps durant lesquelles la production peut s'effectuer.

La demande $d_{i,j}$ de chaque produit j à la fin de chaque période i étant connue, l'objectif de la planification est de déterminer les quantités $x_{i,j}$ de produits de type j qui seront effectivement fabriqués durant la période i. Le fait que les quantités fabriquées durant une période peuvent être différentes des demandes, peut provoquer le cumul (ou la rupture) de certains produits à la fin de chaque période. Le niveau de stock $ST_{i,j}$ pour chaque produit j à la fin d'une période i est donné par (en supposant que le niveau de stock initial pour chaque produit est nul) :

$$ST_{i,j} = \sum_{k=1}^{i} \left(x_{i,j} - d_{i,j} \right)$$

Ce problème peut être représenté graphiquement comme indiqué dans la figure IV.7, où $D_i = [d_{i,j} : j = 1,...,n]$ est le vecteur dont les composantes sont les demandes, et $X_i = [x_{i,j} : j = 1,...,n]$ est le vecteur dont les composantes sont les niveaux de production.

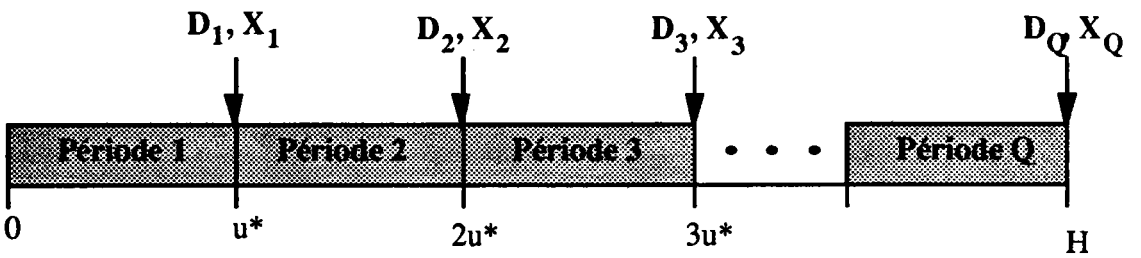


figure IV.7 : Le problème de planification

Soit N le modèle d'un système de production non-cyclique et soit *m* le nombre de RPES connexes sans conflits extraits de N. Définissons NC_1, NC_2, ..., NC_m, l'ensemble de ces RPES. Pour tout k=1, 2, ..., m, NC_k est un sous-réseau dérivé d'un t-invariant à support minimal $V_k = \left[v_1^k,...,v_q^k\right]$. La résolution du problème de planification consiste dans notre cas, à définir le nombre de fois que chaque sous-réseau est activé durant chaque période élémentaire. Nous cherchons donc à déterminer des vecteurs :

$$B_i = \sum_{k=1}^{m} a_{i,k} \cdot V_k$$, les coefficients $a_{i,k}$ étant des entiers non négatifs et i = 1, 2,..., Q.

Quel que soit le critère considéré, il s'agit de minimiser une fonction de coût *c(A,D)* dépendant du vecteur des coefficients $A = [a_{i,k} : i = 1, 2,..., Q;\ k = 1, 2,...m]$ et du vecteur des demandes $D = [d_{i,j} : i = 1, 2,..., Q;\ j = 1, 2,...n]$. La formulation complète de ce problème dans le cas des deux critères considérés est présentée dans la section suivante. Le nombre de fois que chaque sous-réseau est utilisé étant connu, on peut facilement calculer le nombre de fois que chaque transition du modèle doit être franchie au cours de chaque période élémentaire.

Rappelons que rien ne garantit que la solution qu'on obtient au niveau de la planification sera réalisable au niveau de l'ordonnancement. Ceci est du au fait de que nous n'avons pas considéré la disponibilité ou non des ressources dans le modèle utilisé pour la planification. Donc, pour avoir une certaine marge de flexibilité, nous adopterons la stratégie qui consiste à réduire artificiellement la durée de la période au moment des calculs de la planification. En conséquence, dans certains cas, il sera nécessaire de retourner plusieurs fois au niveau de la planification (en réduisant la durée de la période à chaque fois) jusqu'à ce qu'une solution

puisse être trouvée au niveau de l'ordonnancement.

IV.3.2 Formulation du problème

IV.3.2.1 Modèle linéaire à variables entières

Le premier critère qu'on considère correspond à la minimisation de la somme totale des coûts de stockage et de rupture des produits. En d'autres termes, ceci signifie qu'on pénalise le critère à chaque fois que la fabrication d'un produit durant une période donnée ne correspond pas à la demande fixée pour ce produit à la fin de cette période.

Nous supposons que le stock initial de chaque produit est nul. Le critère à minimiser c s'écrit dans ce cas :

$$c = \sum_{j=1}^{n}\left[p_j^+ \sum_{s=1}^{Q}\left[\sum_{i=1}^{s}\left(x_{i,j}-d_{i,j}\right)\right]^+\right] - \sum_{j=1}^{n}\left[p_j^- \sum_{s=1}^{Q}\left[\sum_{i=1}^{s}\left(x_{i,j}-d_{i,j}\right)\right]^-\right] \qquad (1)$$

où

$a^+ = \mathrm{Max}(a,0)$ et $a^- = \mathrm{Min}(a,0)$;

p_j^+ est le coût de stockage d'une unité de produit de type j durant une période élémentaire ;

p_j^- est le coût de rupture d'une unité de produit de type j durant une période élémentaire.

Le nombre de produits $x_{i,j}$ de type j à fabriquer durant la période élémentaire i correspond, dans le cas du modèle dont nous disposons, à la somme du nombre de franchissements de toutes les transitions de sortie du RPES associé au produit j. Ceci peut être exprimé de la manière suivante :

$$x_{i,j} = \sum_{t\in T_j}\sum_{k=1}^{m} a_{i,k} v_t^k \text{, pour i = 1, 2,..., Q et j = 1, 2, ..., n.} \qquad (2)$$

où :

$a_{i,k}$ est le nombre de fois où le sous-réseau NC_k est utilisé durant la période i ;

T_j est l'ensemble des transitions de sortie correspondant au produit de type j ;

v_t^k est le coefficient correspondant à la transition t dans le t-invariant associé au sous-réseau NC_k. Si la transition t n'apparaît pas dans le sous-réseau, ce coefficient prend la valeur zéro ;

m est le nombre total de sous-réseaux sans conflit utilisés.

Donc, à partir de (1) et (2) nous pouvons réécrire le critère à minimiser comme indiqué ci-dessous :

$$c = \sum_{j=1}^{n}\left[p_j^+ \sum_{s=1}^{Q}\left[\sum_{i=1}^{s}\left(\sum_{t\in T_j}\sum_{k=1}^{m} a_{i,k}v_t^k - d_{i,j}\right)\right]^+\right] - \sum_{j=1}^{n}\left[p_j^- \sum_{s=1}^{Q}\left[\sum_{i=1}^{s}\left(\sum_{t\in T_j}\sum_{k=1}^{m} a_{i,k}v_t^k - d_{i,j}\right)\right]^-\right] \quad (3)$$

Au niveau de l'utilisation des ressources, il est clair que la charge totale de chaque ressource dans chaque période élémentaire ne doit pas dépasser la durée u utilisable dans la période. Cette durée est inférieure ou égale à la durée de la période ($u \le u^*$). Donc, la contrainte suivante doit être respectée :

$$\sum_{t\in T_r}\left(\sum_{k=1}^{m} a_{i,k}v_t^k\right)\theta_t \le u, \quad \text{pour } i = 1, 2,...,Q \text{ et } r = 1, 2, ..., M. \quad (4)$$

où:

T_r est l'ensemble des transitions correspondant à la ressource R ;
θ_t est le temps de franchissement de la transition t ;
u est la durée utilisable dans une période élémentaire ;
M est le nombre total de ressources disponibles.

Finalement, ce problème peut être transformé en un problème de programmation linéaire de la manière suivante :

$$PP1 \begin{cases} \textbf{MIN } C = \sum_{j=1}^{n}\left[p_j^+ \sum_{s=1}^{Q} y_s^j\right] + \sum_{j=1}^{n}\left[p_j^- \sum_{s=1}^{Q} z_s^j\right] \\ \text{s.c.} \\ \text{(a) la relation (4) est vérifiée} \\ \text{(b) } y_s^j \ge \sum_{i=1}^{s}\left(\sum_{t\in T_j}\sum_{k=1}^{m} a_{i,k}v_t^k - d_{i,j}\right); \\ \quad z_s^j \ge \sum_{i=1}^{s}\left(d_{i,j} - \sum_{t\in T_j}\sum_{k=1}^{m} a_{i,k}v_t^k\right); \\ \quad s = 1,...Q \text{ et } j = 1,...,n \\ \text{(c) } a_{i,k} \ge 0;\ i = 1,...Q \text{ et } k = 1,...,m, \text{ sont des entiers} \\ \text{(d) } y_s^j \ge 0 \text{ et } z_s^j \ge 0;\ s = 1,...Q \text{ et } j = 1,...,n, \text{ sont des entiers} \end{cases}$$

Résolution du problème *PP1*

Le problème *PP1* est un problème de programmation linéaire en nombres entiers. Nous avons mis en œuvre deux méthodes pour résoudre ce problème : l'une fournit la solution exacte mais peut être coûteuse en temps de calcul, et l'autre est une heuristique qui fournit une solution approchée avec des temps de calcul acceptable. Dans les deux cas nous utilisons les fonctions disponibles dans la bibliothèque d'optimisation **OSL** (Optimization Subroutine Library) qui tourne sur un ordinateur du type IBM RS6000.

La première méthode consiste à résoudre le problème *PP1* tel qu'il a été défini antérieurement. La méthode utilisée pour résoudre ce problème est basée sur la procédure de séparation et évaluation. On fera deux remarques concernant le temps de calcul de la solution du problème *PP1*. D'abord, nous avons constaté que la durée de résolution est sensible à la variation des valeurs des contraintes (par exemple la durée utilisable de la période), et que donc pour un même problème nous pouvons avoir des temps de calcul assez différents suivant les valeurs prises par les paramètres du problème. La deuxième remarque est que le temps de calcul croît vite avec le nombre de variables. Ces remarques ne sont pas surprenantes compte tenu de la méthode utilisée et du fait que dans notre problème toutes les variables doivent prendre des valeurs entières. Ainsi, nous avons mis au point une méthode alternative pour le cas où on souhaite obtenir rapidement une bonne solution. Nous décrivons cette méthode dans ce qui suit.

La deuxième méthode consiste à utiliser une heuristique pour résoudre le problème *PP1*. L'idée de cette heuristique est de résoudre Q fois (Q est le nombre de périodes) le problème *PP1'* dérivé du problème *PP1* dans lequel nous supposons que toutes les variables sont réelles et dans lequel nous ne considérons que les périodes dont la solution n'a pas encore été déterminée. Ceci est résumé dans l'algorithme IV.3 ci-dessous.

Algorithme IV.3 : Heuristique pour la résolution du problème ***PP1***

1 - Faire i = 1 ;

2 - Tant que i ≤ Q faire :

- 2.1 - Résoudre le problème PP1' dans lequel on ne considère que les périodes de i à Q. Nous supposons ici que toutes les variables sont réelles ;
- 2.2 - On ne garde que la solution correspondant à la période i. Les $a_{i,k}$ sont les parties entières des solutions ;
- 2.3 - En partant des produits ayant des coûts de rupture les plus élevés et qui sont les plus en retard, essayer de placer une unité supplementaire de chaque type de produit ;
- 2.4 - Additionner à la demande de chaque produit pour la période i+1 la différence

entre la production et la demande durant la période i.

2.5 - Faire i = i + 1.

Une conséquence du fait que dans l'algorithme IV.3 les solutions sont des nombres réels est que ceci permet d'obtenir rapidement une solution du problème PP1. Comme nous allons le voir dans le chapitre V, l'écart entre le résultat fourni par cette heuristique et celui correspondant à la solution optimale est situé dans des limites acceptables. Néanmoins, nous remarquons que cette heuristique ne garantit pas que dans la solution obtenue les demandes soient toutes satisfaites (même si cela est théoriquement possible). De plus, il est clair que dans une hypothèse où nous n'ordonnançons que la première période (si on considère, par exemple, le cas d'un horizon glissant), il suffit de faire une seule itération de l'algorithme (i.e. faire Q = 1).

Enfin, avant de considérer le second critère, nous ferons la remarque suivante. La complexité de résolution du problème *PP1* dépend du nombre de sous-réseaux qu'on utilise. Donc, dans certains cas il sera préférable de réduire le nombre de ces sous-réseaux (en combinant certains, par exemple) afin de gagner en temps de calculs. Bien entendu, dans ce cas, nous risquons de perdre en précision.

IV.3.2.2 Modèle quadratique

Le critère que nous considérons maintenant correspond à la minimisation de la différence de charge des ressources durant chaque période élémentaire. Nous cherchons donc à trouver une solution du problème de planification de telle sorte que les charges des machines à l'horizon H soient les plus voisines possibles les unes des autres. Les hypothèses de départ sont que le stock initial de chaque produit est nul et que la production d'un produit doit satisfaire la demande sur l'horizon considéré. De plus, nous supposons que les demandes de chaque produit durant chaque période élémentaire sont strictement positives.

Dans ce cas, le critère à minimiser *c* a la forme suivante :

$$c = \sum_{i=1}^{Q}\left[\left(\sum_{r=1}^{M}\sum_{s=r+1}^{M}\left|WL_{r,i} - WL_{s,i}\right|^2\right) + \left(\left|\frac{x_{i,j}}{d_{i,j}} - 1\right|\right)\right] \qquad (5)$$

où

M est le nombre de ressources ;

Q est le nombre de périodes considérées ;

$WL_{r,i}$ est le taux d'occupation de la ressource r durant la période i ;

$WL_{s,i}$ est le taux d'occupation de la ressource s durant la période i ;

$x_{i,j}$ est la production du produit j durant la période i ;
$d_{i,j}$ est la demande du produit j durant la période i ;

La deuxième partie de l'équation (5) correspond à une pénalisation de la fonction objectif à chaque fois que la production d'un produit, dans une période élémentaire, est différente de sa demande.

Le taux d'occupation $WL_{r,i}$ d'une ressource r et durant la période i, est la donnée du temps total d'utilisation de cette ressource divisé par la durée u utilisable dans une période élémentaire. Ceci est donné par :

$$WL_{r,i} = \frac{\sum_{t \in T_r} \left(\sum_{k=1}^{m} a_{i,k} v_t^k \right) \theta_t}{u} \qquad (6)$$

où:

T_r est l'ensemble des transitions correspondant à la ressource r ;
θ_t est le temps de franchissement de la transition t ;
u est la durée utilisable dans une période élémentaire ;
m est le nombre total de sous-réseaux utilisées.

De même que dans le cas précédent, l'utilisation d'une ressource durant une période élémentaire ne doit pas dépasser la durée de la période et, en conséquence, la contrainte (4) doit aussi être respectée dans ce cas. De plus, la production totale de chaque type de produit à la fin de l'horizon de planification doit correspondre exactement à la demande totale de ce produit. Donc, la relation (2) s'écrit dans ce cas :

$$X_j = \sum_{i=1}^{Q} x_{i,j} = \sum_{i=1}^{Q} \sum_{t \in T_j} \sum_{k=1}^{m} a_{i,k} v_t^k = \sum_{i=1}^{Q} d_{i,j} = D_j \text{ ,pour } j = 1, 2, \ldots, n. \qquad (7)$$

Nous définissons $y_{r,s,i} = | WL_{r,i} - WL_{s,i} |$ et $z_{i,j} = | x_{i,j}/d_{i,j} - 1 |$. Nous pouvons supprimer les valeurs absolues en introduisant des variables supplémentaires :

$$\begin{cases} y_{r,s,i} \geq WL_{r,i} - WL_{s,i} \\ y_{r,s,i} \geq WL_{s,i} - WL_{r,i} \end{cases} \qquad \begin{cases} y_{i,j} \geq \dfrac{x_{i,j}}{d_{i,j}} - 1 \\ y_{i,j} \geq 1 - \dfrac{x_{i,j}}{d_{i,j}} \end{cases} \qquad (8)$$

En considérant les relations (5), (6), (7) et (8), le problème de planification peut être mis sous la forme du problème de programmation quadratique suivant :

$$
PP2 \begin{cases}
\text{MIN } C = \sum_{i=1}^{Q}\left[\sum_{r=1}^{M}\sum_{s=r+1}^{M}\left(y_{r,s,i}\right)^2 + \left(z_{i,j}\right)\right] \\
\text{s.c.} \\
\text{(a) le relations (4) et (7) sont vérifiées} \\
\text{(b) } y_{r,s,i} \geq \frac{1}{u}\sum_{k=1}^{m}\left[\left(\sum_{t\in Q_r} a_{i,k}\cdot v_t^k\cdot\theta_t\right)-\left(\sum_{t\in Q_s} a_{i,k}\cdot v_t^k\cdot\theta_t\right)\right]; \\
\quad y_{r,s,i} \geq \frac{1}{u}\sum_{k=1}^{m}\left[\left(\sum_{t\in Q_s} a_{i,k}\cdot v_t^k\cdot\theta_t\right)-\left(\sum_{t\in Q_r} a_{i,k}\cdot v_t^k\cdot\theta_t\right)\right]; \\
\quad r = 1,\ldots M;\ s = r+1,\ldots M \text{ et } i = 1,\ldots,Q \\
\text{(c) } d_{i,j}\cdot z_{i,j} - x_{i,j} \geq -d_{i,j} \\
\quad x_{i,j} + d_{i,j}\cdot z_{i,j} \geq d_{i,j} \qquad i = 1,\ldots,Q,\ j = 1,\ldots,n \\
\text{(d) } a_{i,k} \geq 0;\ i = 1,\ldots Q,\ k = 1,\ldots,m \text{ ,sont des entiers} \\
\text{(e) } y_{r,s,i} \geq 0;\ r = 1,\ldots M;\ s = r+1,\ldots M \text{ et } i = 1,\ldots, Q
\end{cases}
$$

Résolution du problème PP2

Pour résoudre le problème ***PP2*** nous utilisons également les fonctions disponibles dans la bibliothèque d'optimisation OSL. Etant donné que OSL ne permet pas la définition de variables entières dans la résolution de problèmes de programmation quadratique, nous déterminons la solution du problème ***PP2*** en deux étapes :

(i) D'abord on résout le problème ***PP2*** en considérant que les coefficients $a_{i,k}$ peuvent prendre des valeurs réelles ;

(ii) Ensuite on résout un problème de programmation linéaire en nombres entiers où chaque coefficient $a_{i,k}$ est borné inférieurement (supérieurement) par l'entier immédiatement inférieur (supérieur) à la solution réelle. Le problème considéré ici est celui formé par les contraintes (4) et (7) et dans lequel on cherche simplement à trouver une solution entière.

Dans le cas où les demandes ne peuvent pas être satisfaites sur l'horizon de planification, l'algorithme ci-dessus ne fournit aucune solution. Dans le cas contraire, il est clair que la solution qu'on obtient à la suite de deux étapes de l'algorithme n'est pas optimale. Cependant, compte tenu du fait que la solution obtenue en première partie de l'algorithme est optimale, on peut espérer que la solution qu'on obtient dans la deuxième partie est proche de la solution optimale. Dans le chapitre V nous analysons la qualité des résultats obtenus à la suite d'une série d'expérimentations.

IV.4 Ordonnancement

Le problème d'ordonnancement est un problème NP-difficile [French 1982]. Les réseaux de Petri temporisés ont été largement utilisés dans l'analyse de performances des systèmes pour lesquels l'ordonnancement était donné, mais très peu de travaux ont été consacrés à l'utilisation des réseaux de Petri dans la détermination de l'ordonnancement lui même. Le fait de considérer les routages alternatifs introduit une difficulté supplémentaire, car même dans le domaine de l'ordonnancement le nombre de travaux dans lequel ce problème a été pris en compte est très faible. Dans ce cadre, les seuls travaux d'ordonnancement basés sur les réseaux de Petri qui nous connaissons sont dûs à [Lee et DiCesari 92] et [Shih et Sekigushi 91]. Ces travaux sont basés essentiellement sur la construction d'un arbre de décision et la recherche d'un chemin dans cet arbre. Quelques règles heuristiques.sont utilisées pour (i) réduire la taille de l'arbre obtenu, et (ii) guider la recherche. Les algorithmes de recherche utilisés sont dérivés de ceux utilisés dans le domaine de l'intelligence artificielle.

Nous développons dans cette section deux nouvelles méthodes d'ordonnancement basées sur les réseaux de Petri, que nous utilisons pour la vérification de la faisabilité du plan de production.

Le plan de production qui a été déterminé au cours l'étape précédente spécifie le nombre de fois que chaque transition du modèle doit être franchie durant chaque période élémentaire. Comme nous l'avons déjà souligné, ce plan est calculé sans prendre en compte le fait que les ressources peuvent ne pas être disponibles au moment où elles sont affectées. Il faut donc vérifier si nous parvenons à organiser le système de sorte que les franchissements prévus au cours de chaque période élémentaire puissent effectivement être réalisés. L'objectif de l'ordonnancement est donc de définir les instants où chaque transition du modèle doit commencer à être franchie. Pour cela, nous devons définir une séquence de franchissements des transitions de façon à supprimer tous les conflits structurels du modèle.

Comme nous l'avons montré antérieurement, le modèle que l'on considère en ordonnancement est dérivé de celui de la planification par l'inclusion d'une place contenant un jeton pour chaque ressource (voir la figure IV.3 pour un exemple). Par la suite, nous ferons l'hypothèse que les transitions sont franchies dès qu'elles sont franchissables, et nous montrons comment on résout le problème d'ordonnancement. Deux méthodes ont été mises au point pour cela : l'une est basée sur l'algorithme du recuit simulé, l'autre est basée sur la procédure par séparation et évaluation. Dans les paragraphes suivants nous décrivons la mise en œuvre de ces deux méthodes.Une étude comparative de leurs performances est présentée dans le chapitre V. Nous dédions aussi un paragraphe à la présentation d'une nouvelle méthode qui

est en cours de développement.

IV.4.1 Méthode basée sur le recuit simulé

L'approche du recuit simulé ([Kirkpatrick 83], [Laarhoven et Aarts 87]) est une approche de recherche par voisinages qui a comme principal avantage le fait qu'elle permet d'éviter de tomber dans des optima locaux. Dans cette approche, un certain nombre de paramètres sont utilisées pour contrôler le déroulement de l'algorithme. La définition des valeurs de ces paramètres est fonction du type d'application, mais elles se situent en général entre des limites connues. Les paramètres de contrôle utilisés sont les suivants :

- la température initiale (T_0) ;
- la température finale (T_f) ;
- le nombre maximal de solutions acceptées pour un palier de température (N_a) ;
- le nombre maximal de solutions rejetées pour un palier de température (N_r) ;
- la raison géométrique de la diminution de température (ΔT) comprise entre 0 et 1.

La première étape de cet algorithme consiste à déterminer une configuration initiale du système et à calculer la valeur du critère pour cette configuration. Ensuite, on procède par itérations successives : à chaque itération, on perturbe la dernière configuration acceptée du système et on recalcule la valeur du critère. Cette nouvelle configuration est acceptée si elle améliore la valeur du critère. Sinon, elle peut être acceptée avec un probabilité qui décroît avec le volume des calculs effectués antérieurement et l'écart entre la nouvelle solution et l'ancienne. Si l'utilisateur est satisfait de la meilleure configuration trouvée, il peut arrêter à tout instant le déroulement de l'algorithme.

Une description détaillée de l'algorithme du recuit simulé est donnée ci-dessous. Il comporte essentiellement deux boucles imbriquées : la boucle extérieure contrôle le processus de diminution de la température, et la boucle intérieure contrôle la longueur des paliers de température. L'algorithme s'arrête quand la température atteint une température donnée.

Algorithme IV.4 : Algorithme général du recuit simulé

1- Donner la température initiale T_0 et la température finale T_f. Faire $T := T_0$;
2- Donner la raison géométrique ΔT de la suite de températures, le nombre d'acceptations N_a et le nombre de refus N_r;
3- Générer une configuration initiale C_0 et calculer la valeur du critère x_0 correspondant à cette configuration ;
4- Accepter C_0 comme la meilleure configuration trouvée. Faire $C^*=C_0$, $x^*=x_0$;

5- Tant que ($T \leq T_f$) faire :

5.1- Initier Nb_accept = 0 et Nb_rejet = 0.;

5.2- Tant que (Nb_accept < N_a et Nb_rejet < N_r) faire :

5.2.1- Générer au hasard une configuration C_1 dans le voisinage de C_0. Calculer la valeur du critère x_1 pour cette configuration ;

5.2.2- Si $x_1 \leq x_0$ alors accepter la configuration C_1, sinon accepter C_1 avec une probabilité $\exp\{(x_1 - x_0)/T\}$. Si C_1 est acceptée faire $C_0=C_1$, $x_0=x_1$, Nb_accept=Nb_accept+1, sinon faire Nb_rejet =Nb_rejet+1 ;

5.2.3- Si $x_0 \leq x^*$ alors la configuration C_0 devient la meilleure solution. Faire $C^* = C_0$, $x^* := x_0$;

5.3- Calculer la nouvelle température $T := T^* \Delta T$;

6- Editer C* comme la meilleure configuration.

Le processus de calcul de la valeur du critère d'une configuration donnée, le processus de génération d'une configuration initiale, ainsi que celui du passage d'une configuration à une autre, dépendent de l'application. Nous examinons chacun de ces points dans ce qui suit.

Dans notre cas, le critère qu'on considère correspond au temps total de fabrication (makespan) obtenu après avoir ordonnancé les opérations à exécuter sur la période. Pour calculer la valeur de ce critère nous faisons une simulation.

La détermination d'une configuration initiale consiste dans notre application à faire en sorte que tous les conflits structurels du modèle soient résolus. Il faut donc décider de l'ordre d'affectation des jetons aux transitions de sortie pour toute place comportant plusieurs transitions de sortie. Par exemple, si t_1 et t_2 sont deux transitions de sortie d'une place p, la séquence (t_1, t_1, t_2,...) pour la place p signifie que les deux premiers jetons arrivés dans p sont destinés à effectuer deux tirages de la transition t_1, le troisième jeton permet le franchissement de la transition t_2 et ainsi de suite.

Dans le modèle utilisé pour l'ordonnancement (voir la figure IV.3 par exemple), deux types de décisions sont à prendre : les décisions du premier type concernent le choix des ressources pour effectuer les opérations si plusieurs ressources sont possibles (appelée CR : Choix Ressource) ; les décisions du second type concernent les choix des opérations à effectuer sur une ressource si plusieurs opérations sont en compétition (appelée CO : Choix Opération). Donc, pour définir une configuration initiale, nous devons choisir une séquence pour toute place représentant une ressource (séquence CO) et une séquence pour toute place représentant un stock tampon et qui a plusieurs transitions de sortie (séquence CR).

Il est clair que si on génère les deux types de séquences CR et CO indépendamment, on risque de bloquer le système. A titre d'exemple, considérons le modèle donnée par la figure IV.8 dans lequel on suppose que chaque transition doit être franchie au moins une fois et que le marquage initial est celui indiqué dans la figure.

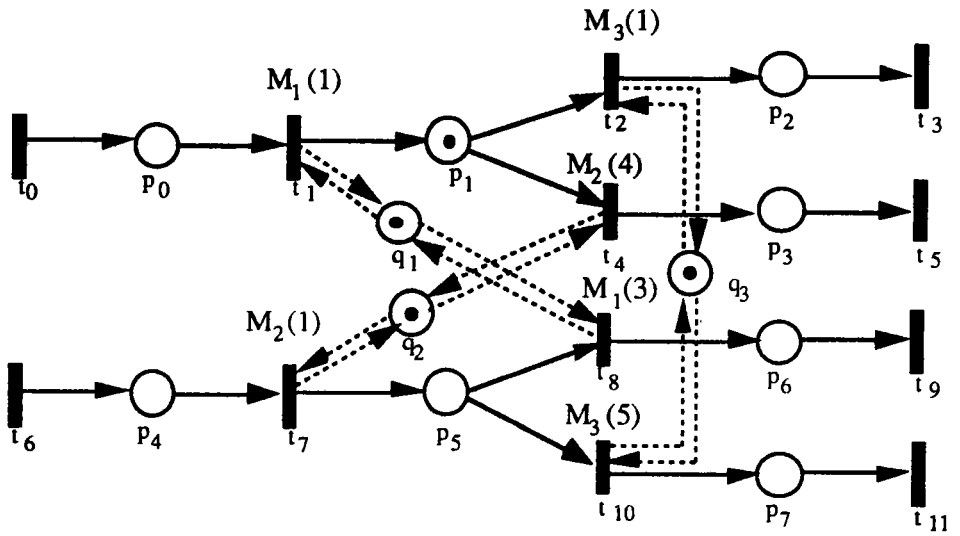


Figure. IV.8 : Exemple de blocage

Le système sera bloqué si les séquences CR et CO sont générées de telle sorte qu'elles commencent comme indiqué dans la suite.

CO: (q_1) t_8, t_1,... ;
(q_2) t_4, t_7,... ;
(q_3) t_{10}, t_2,... ;

CR: (P_1) t_2, t_4,... ;
(P_2) t_8, t_{10},... ;

Pour éviter ce problème nous utilisons la stratégie suivante : nous générons au hasard les séquences CR relatives aux places appartenant aux gammes de fabrication qui ont plus d'une transition de sortie, et on construit les séquences CO relatives aux places ressources au fur et à mesure du déroulement de la simulation . Dans ce dernier cas, la séquence relative à une place ressource q est construite par : (i) l'identification de toutes les transitions associées à cette ressource et qui sont franchissables à chaque pas de la simulation, et (ii) le choix au hasard d'une des transitions parmi celles qui ont été sélectionnées à l'étape précédente.

Le dernier point à considérer concerne la détermination d'une configuration voisine. Il est évident que les seules modifications possibles concernent les séquences CR, car les séquences CO sont générées au cours de la simulation. Donc, pour obtenir une configuration voisine nous modifions un certain nombre de séquences CR (30% du nombre total de ces séquences),

lesquelles sont choisies au hasard. Cette modification consiste à permuter au hasard la position de deux transitions différentes.

Le désavantage d'un algorithme de type recuit-simulé est qu'on ne peut jamais assurer l'optimalité de la solution trouvée sauf dans le cas où la ressource goulot est saturée. De plus, même si la taille du modèle est petite, le temps de calcul peut être relativement long, alors que la solution n'est pas nécessairement bonne.

IV.4.2 Méthode basée sur la procédure par séparation et évaluation

La procédure par séparation et évaluation consiste à décomposer successivement un sous-ensemble de solutions en des sous-ensembles encore plus petits. Pour chaque sous-ensemble obtenu, on calcule une borne inférieure de toute solution de ce sous-ensemble. Si cette borne est supérieure ou égale à la valeur du critère ("makespan") de la meilleure solution déjà trouvée, on est sûr qu'il n'existe pas de solution améliorante dans ce sous-ensemble. Par conséquent, on abandonne la recherche dans ce sous-ensemble. Cette approche est souvent représentée par une arborescence où chaque nœud représente un sous-ensemble. Dans la suite du paragraphe, nous parlons de "nœuds", au lieu de sous-ensemble de solutions. Les nœuds descendants d'un nœud sont les sous-ensembles obtenus en décomposant les sous-ensembles représentés par celui-ci.

Dans cette méthode, nous devons disposer d'une borne supérieure qui est la valeur du critère de la meilleure solution déjà trouvée. Nous utilisons la durée réelle d'une période élémentaire comme la borne supérieure initiale, car les solutions qui nous intéressent sont celles où la durée totale est inférieure ou égale à la durée réelle de la période.

Dans notre algorithme, chaque nœud représente une solution partielle où au moins une transition commence à être franchie à l'instant τ. Dans cette solution partielle, on connaît le nombre de franchissements restants σ_t pour chaque transition t. On a également un ensemble de transitions en cours de franchissement noté F. Pour chaque transition $t \in F$, on connaît le temps de franchissement restant à l'instant τ, soit θ'_t. Si on désigne par T_k l'ensemble des transitions représentant les opérations effectuées par la ressource k, alors il n'existe pas deux transitions i et j telles que $i \in T_k$, $j \in T_k$, $i \in F$ et $j \in F$ (la ressource k effectue au plus une opération à la fois). Autrement dit, $\text{card}(F \cap T_k) \leq 1$. Comme nous l'avons souligné plus haut, il faut associer à chaque nœud une borne inférieure. Pour cela nous introduisons une quantité φ_k pour toute ressource k (k = 1, 2, ..., M) telle que

$$\varphi_k = \begin{cases} \theta'_t & \text{si } t \in F \cap T_k, \\ \min_{t \in F} \theta'_t & \text{si } F \cap T_k = \emptyset. \end{cases}$$

En effet, si la ressource k est occupée à l'instant τ, alors, φ_k représente la durée pendant laquelle la ressource k continuera à être occupée. Par conséquent, elle ne peut commencer à effectuer une nouvelle opération qu'à partir de l'instant $\tau+\varphi_k$. Si la ressource k est inoccupée à l'instant τ, alors aucune des transitions appartenant à T_k n'est franchissable (les places d'entrée ne contiennent pas suffisamment de jetons). Il faut que le franchissement d'au moins une transition soit achevé pour qu'il soit possible qu'une des transitions appartenant à T_k devienne franchissable. Cette durée est précisément représentée par φ_k. Dans touts les cas, $\tau+\varphi_k$ représente l'instant au plus tôt où une des transitions appartenant à T_k devient franchissable.

En outre, même si la ressource k peut commencer à effectuer une opération à l'instant $\tau+\varphi_k$, l'ordonnancement ne peut être terminé avant que la ressource k ait traité sa charge restante. Celle-ci peut être minorée par $\sum_{t \in T_k} \sigma_t \times \theta_t$. La durée totale de production (makespan) est donc supérieure ou égale à $\tau+\varphi_k+\sum_{t \in T_k} \sigma_t \times \theta_t$, pour k=1, 2, ..., M. Autrement dit, la quantité

$$LB = \tau + \max_{k=1,2,\ldots,M} \left\{ \sum_{t \in T_k} \sigma_t \times \theta_t + \varphi_k \right\}$$

est une borne inférieure du makespan, compte tenu de la solution partielle représentée par le nœud.

Lors de la séparation d'un nœud, nous créons des nœuds descendants après la fin du franchissement de la (ou des) première(s) transition(s) en cours de franchissement dans le nœud, en considérant toutes les combinaisons possibles des franchissements des transitions.

Nous identifions tout d'abord l'ensemble des transitions franchissables Φ_k pour chaque ressource libre k. Une transition est franchissable si et seulement si chacune de ses places d'entrée contient au moins un jeton. On crée autant de nœuds qu'il y a de combinaisons possibles en prenant un élément de chaque Φ_k. Bien entendu, ces nœuds ne sont pas tous réalisables en raison de conflits structurels. Si deux transitions $t \in F_k$ et $t' \in F_{k'}$ partagent une même place d'entrée qui ne contient qu'un seul jeton, alors t et t' ne peuvent pas être franchies

en même temps. Dans ce cas, ce nœud est remplacé par deux nouveaux nœuds tels que dans l'un, t est franchie mais pas t', et dans l'autre, t' est franchie mais pas t.

Dans un nœud descendant, l'ensemble des transitions en cours de franchissement est constitué des nouvelles transitions dont le franchissement vient de commencer et des transitions du nœud ascendant dont le franchissement n'est pas encore achevé.

En ce qui concerne le choix du nœud à séparer, nous utilisons une méthode en "profondeur d'abord" modifiée : parmi les nœuds créés le plus récemment, l'un de ceux qui ont la plus petite borne inférieure est choisi pour la séparation suivante. Un choix plus fin dans ce cas permettrait d'accélérer la recherche. Lorsque deux nœuds ont la même borne inférieure, on choisit le nœud contenant le plus de transitions pour lesquelles une des places de sortie n'a pas de jeton. Ceci permettra d'accélérer le flux des jetons. En cas d'égalité, on choisit un nœud contenant une transition dont le nombre de jetons dans la place d'entrée est le plus petit. Ce choix est justifié par le fait que si cette transition est tirée, alors la place d'entrée devient vide plus rapidement et par conséquent à la prochaine séparation, une des transitions d'entrée de cette place aura plus de chance d'être tirée.

Le désavantage d'une méthode de type procédure par séparation et évaluation est que le besoin en mémoire et le temps de calcul augmentent très vite avec la taille du problème à résoudre. C'est pourquoi l'application de ce type de méthode se limite à des problèmes de taille raisonnable. Dans la mise en œuvre de cette méthode, nous laissons à l'utilisateur le choix entre continuer ou interrompre l'exécution après chaque solution intermédiaire réalisable. Notons que la première solution admissible (i.e. le premier ordonnancement dont la durée totale est inférieure ou égale à u*) peut être acceptée.

IV.4.3 Une nouvelle proposition

Afin de conclure notre étude sur l'ordonnancement, nous présentons maintenant une nouvelle heuristique basée sur la détermination d'un chemin critique. Nous tenons à préciser que cette heuristique est en cours de développement, et que notre objectif en la présentant ici est simplement de montrer qu'il existe d'autres alternatives à explorer dans le cadre de l'utilisation des réseaux de Petri pour la détermination de l'ordonnancement.

Contrairement au deux méthodes présentées précédemment, l'idée de cette heuristique est de commencer par un ordonnancement réalisable et d'essayer d'améliorer l'ordonnancement (i.e. réduire le temps de production) à chaque itération. Une manière simple pour obtenir un ordonnancement réalisable consiste à procéder comme nous l'avons fait dans la méthode basée

sur recuit simulé : fixer les séquences pour les places CR et déterminer les séquences pour les places CO à l'aide d'une simulation. A partir de cela, on peut déterminer les quantités $S_t(k)$ correspondant à l'instant de début du $k^{ème}$ franchissement de la transition t, et $F_t(k) = S_t(k)+\theta_t$ qu'indique l'instant de fin de ce franchissement.

Pour améliorer l'ordonnancement, l'idée de base est de déterminer à chaque itération le chemin critique, lequel correspond à une séquence de paires définie de la façon suivante :

$$CC = <(t_{i_1}, k_{\alpha_1}),\ldots, (t_{i_r}, k_{\alpha_r})>$$

tel que :

(i) $S_{t_{i_1}}(k_{\alpha_1}) = 0$;

(ii) $F_{t_{i_j}}(k_{\alpha_j}) = S_{t_{i_{j+1}}}(k_{\alpha_{j+1}})$, pour tout j = 1,..., r-1;

(iii) $F_{t_{i_j}}(k_{\alpha_j}) = \underset{(t,k)}{MAX}\{F_t(k)\}$. Ceci correspond à la durée de la production (makespan).

A partir de la définition ci-dessus, il est clair que la condition nécessaire (mais non suffisante) pour réduire le temps de production est de réduire $F_{t_{i_j}}(k_{\alpha_j})$ en avançant certains franchissements de transitions appartenant au chemin critique. Ceci n'est possible que si nous retardons quelques franchissements de transitions qui ne sont pas dans le chemin critique.

Afin de réduire la complexité des calculs, l'idée est de déterminer à priori quels sont les changements qui peuvent améliorer (réduire) le temps total de production. Pour cela, on calcule pour chaque transition son *degré de liberté*, lequel est le temps maximal dont le franchissement de la transition peut être retardé sans que ceci augmente la valeur du temps total de production. Le calcul du degré de liberté de chaque transition doit prendre en compte l'ordonnancement et aussi l'ordre d'utilisation des ressources pour chaque opération. L'objectif est donc d'avancer le franchissement d'une transition du chemin critique si cela conduit à retarder le franchissement d'une autre transition d'une quantité inférieure à son degré de liberté.

Donc le processus d'obtention de la solution consiste à boucler tant que des changements sont possibles. A chaque itération, on recalcule d'abord le chemin et ensuite on détermine un couple de transitions à permuter. Cette heuristique semble prometteuse et peut susciter de nouveaux développements.

IV.5 Une première application numérique

Nous présentons maintenant un petit exemple détaillé pour illustrer la modélisation et

l'évaluation d'un système de production non-cyclique. Des exemples de taille plus importantes seront considérés dans la deuxième partie du chapitre V.

L'exemple que nous proposons ici correspond à un système composé de six machines, notées M_i, i=1,2,...6. Ce système fabrique trois types de produits, notés P_1, P_2 et P_3, dont les gammes de fabrication sont représentées dans la figure IV.9. Dans cette figure, chaque boîte représente soit une opération élémentaire, soit un ensemble d'opérations élémentaires qui sont effectuées en série (indiqué par une flèche) ou en parallèle (indiqué par un point virgule). Par exemple, la première partie de la fabrication du produit P_1 peut être effectuée soit par M_1 suivie de M_2, soit par M_3 suivie de M_5, soit par M_1 suivie de M_3.

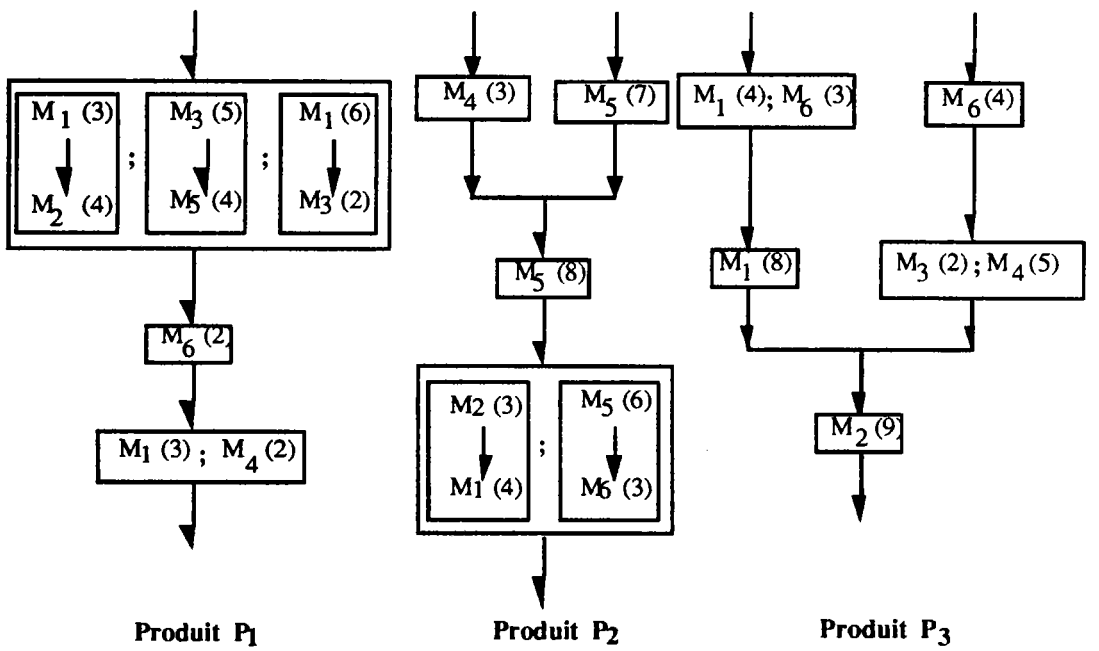


figure IV.9 : Gammes de fabrication des trois produits

a) Modélisation

Le modèle RPES correspondant à ce système de production est donné par la figure IV.10. Dans le souci de faciliter la visualisation du modèle, nous n'avons pas représenté dans cette figure les places ressources. La liste des transitions associées à chacune de ces places est la suivante :

$q_1 = <t_1, t_7, t_{10}, t_{20}, t_{26}>$;
$q_2 = <t_2, t_{19}, t_{33}>$;
$q_3 = <t_4, t_8, t_{28}, t_{32}>$;
$q_4 = <t_{12}, t_{15}, t_{18}, t_{32}>$;
$q_5 = <t_5, t_{17}, t_{22}>$;
$q_6 = <t_9, t_{23}, t_{27}, t_{30}>$.

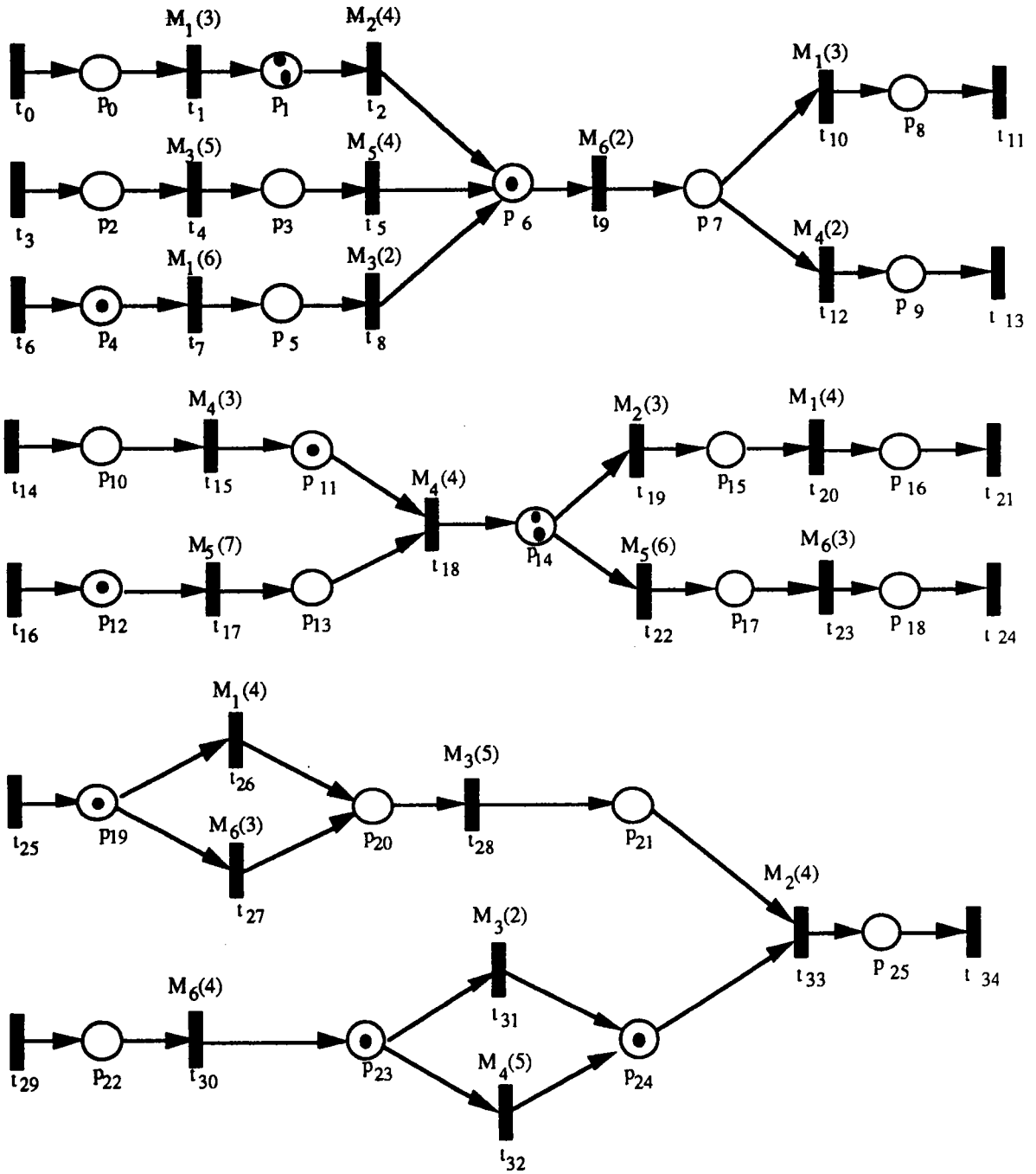


figure IV.10 : Modèle du système

b) Planification

La résolution du problème de planification est réalisée en considérant les hypothèses suivantes :

(i) l'horizon de planification H correspond à trois périodes élémentaires ;

(ii) la durée de chaque période élémentaire est de 45 unités de temps (i.e. u* = 45). Pour

augmenter les chances de pouvoir trouver un ordonnancement réalisable, nous considérons dans les calculs une durée fictive de 43 unités de temps (i.e. u* = 43) ;

(iii) la demande de chaque produit est donnée dans la table IV.6 ;

Périodes	Produits		
	P_1	P_2	P_3
1	6	4	5
2	5	6	2
3	3	3	4

Table IV.6 : Demandes

(iv) le critère d'optimisation qu'on considère est la somme totale des coûts de stockage et de rupture à la fin des trois périodes. Le coût de stockage d'une unité de produit durant une période élémentaire est :

- 10 pour le produit 1 ;
- 20 pour le produit 2 ;
- 40 pour le produit 3.

Le coût de rupture d'une unité de produit durant une période élémentaire est :

- 30 pour le produit 1 ;
- 20 pour le produit 2 ;
- 20 pour le produit 3.

On peut décomposer le modèle de la figure IV.10 en 12 sous-réseaux sans conflit qui correspondent à des t-invariants à support minimal : 6 pour le modèle du produit 1, 2 pour le modèle du produit 2 et 4 pour le modèle du produit 3. Les t-invariants associés à chacun de ces sous-réseaux sont présentés dans les tables IV.7, IV.8 et IV.9.

T-invariants	Transitions													
	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}
V_1	1	1	1	0	0	0	0	0	0	1	1	1	0	0
V_2	0	0	0	1	1	1	0	0	0	1	1	1	0	0
V_3	0	0	0	0	0	0	1	1	1	1	1	1	0	0
V_4	1	1	1	0	0	0	0	0	0	1	0	0	1	1
V_5	0	0	0	1	1	1	0	0	0	1	0	0	1	1
V_6	0	0	0	0	0	0	1	1	1	1	0	0	1	1

Table IV.7 : T-invariants pour le modèle du produit 1

T-invariants	Transitions										
	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{20}	t_{21}	t_{22}	t_{23}	t_{24}
V_7	1	1	1	1	1	1	1	1	0	0	0
V_8	1	1	1	1	1	0	0	0	1	1	1

Table IV.8 : T-invariants pour le modèle du produit 2

T-invariants	Transitions									
	t_{25}	t_{26}	t_{27}	t_{28}	t_{29}	t_{30}	t_{31}	t_{32}	t_{33}	t_{34}
V_9	1	1	0	1	1	1	1	0	1	1
V_{10}	1	1	0	1	1	1	0	1	1	1
V_{11}	1	0	1	1	1	1	1	0	1	1
V_{12}	1	0	1	1	1	1	0	1	1	1

Table IV.9 : T-invariants pour le modèle du produit 3

En appliquant la relation (2) dans le cas de cet exemple, les quantités $x_{i,j}$ de produits du type j à fabriquer durant chaque période élémentaire i sont données par :

$$\begin{cases} x_{i,1} = \sum_{k=1}^{6} a_{i,k}\left(v_{12}^k + v_{14}^k\right) \\ x_{i,2} = \sum_{k=7}^{8} a_{i,k}\left(v_8^k + v_{11}^k\right) \\ x_{i,3} = \sum_{k=9}^{12} a_{i,k}\, v_{10}^k \end{cases} \qquad (9)$$

où :

$v_k = \left[v_1^k, v_2^k, \ldots, v_{14}^k\right]$ pour k = 1, 2, 3, 4, 5, 6

$v_k = \left[v_1^k, v_2^k, \ldots, v_{11}^k\right]$ pour k = 7, 8

$v_k = \left[v_1^k, v_2^k, \ldots, v_{10}^k\right]$ pour k = 9, 10, 11, 12

Compte tenu des valeurs des coefficients des t-invariants présentés dans les tables IV.7, IV.8 et IV.9, les relations données en (9) peuvent être réécrites comme :

$$x_{i,1} = \sum_{k=1}^{6} a_{i,k};\quad x_{i,2} = \sum_{k=7}^{8} a_{i,k};\quad x_{i,3} = \sum_{k=9}^{12} a_{i,k}; \qquad \text{pour } i = 1,2,3 \qquad (10)$$

Nous considérons maintenant la contrainte (4) dans le cas de la ressource M_1. Les opérations effectuées par cette ressource sont modélisées par les transitions $<t_1, t_7, t_{10}, t_{20}, t_{26}>$. La contrainte correspondante est donc :

$$\sum_{k=1}^{6} a_{i,k}\left(3v_2^k + 6v_8^k + 3v_{11}^k\right) + \sum_{k=7}^{8} a_{i,k}\left(4v_7^k\right) + \sum_{k=9}^{12} a_{i,k}\left(4v_2^k\right) \leq 43 \qquad \text{pour } i = 1,2,3$$

En considérant les tables IV.7, IV.8 et IV.9 ces contraintes peuvent être réécrites comme :

$$6a_{i,1}+3a_{i,2}+9a_{i,3}+3a_{i,4}+6a_{i,6}+4a_{i,7}+4a_{i,9}+4a_{i,10} \leq 43 \qquad (11.1)$$
pour i = 1, 2, 3.

Les opérations effectuées par la ressource M_2 sont modélisées par les transitions $<t_2, t_{19}, t_{33}>$. La contrainte (4) s'écrit dans ce cas :

$$\sum_{k=1}^{6} a_{i,k}\left(4v_3^k\right) + \sum_{k=7}^{8} a_{i,k}\left(3v_6^k\right) + \sum_{k=9}^{12} a_{i,k}\left(4v_9^k\right) \leq 43 \qquad \text{pour } i = 1,2,3$$

En considérant les tables IV.7, IV.8 et IV.9 ces contraintes peuvent être réécrites comme :

$$4a_{i,1}+4a_{i,4}+3a_{i,7}+4a_{i,9}+4a_{i,10}+4a_{i,11}+4a_{i,12} \leq 43 \qquad (11.2)$$
pour i = 1, 2, 3.

Les opérations effectuées par la ressource M_3 sont modélisées par les transitions $<t_4, t_8, t_{28}, t_{31}>$. La contrainte (4) implique la relation suivante :

$$\sum_{k=1}^{6} a_{i,k}\left(5v_5^k + 2v_9^k\right) + \sum_{k=9}^{12} a_{i,k}\left(5v_4^k + 2v_7^k\right) \leq 43 \qquad \text{pour } i = 1,2,3$$

A partir des tables IV.7, IV.8 et IV.9 nous pouvons réécrire cette inégalité comme :

$$5a_{i,2}+2a_{i,3}+5a_{i,5}+2a_{i,6}+7a_{i,9}+5a_{i,10}+7a_{i,11}+5a_{i,12} \leq 43 \qquad (11.3)$$
pour i = 1, 2, 3.

Nous considérons maintenant la ressource M_4 dont les opérations sont modélisées par les transitions $<t_{12}, t_{15}, t_{18}, t_{32}>$. Nous écrivons la contrainte (4) pour cette ressource :

$$\sum_{k=1}^{6} a_{i,k}\left(2v_{13}^{k}\right)+\sum_{k=7}^{8} a_{i,k}\left(3v_{2}^{k}+4v_{5}^{k}\right)+\sum_{k=9}^{12} a_{i,k}\left(5v_{8}^{k}\right) \leq 43 \qquad \text{pour } i = 1,2,3$$

De la même façon que dans les cas précédents, les tables IV.7, IV.8 et IV.9 permettent de réécrire cette inégalité comme suit :

$$2a_{i,4}+2a_{i,5}+2a_{i,6}+7a_{i,7}+7a_{i,8}+5a_{i,10}+5a_{i,12} \leq 43 \qquad (11.4)$$
pour i = 1, 2, 3.

Nous appliquons maintenant la contrainte (4) dans le cas de la ressource M_5. Les opérations effectuées par cette ressource sont modélisées par les transitions $<t_5, t_{17}, t_{22}>$, ce qui permet d'écrire :

$$\sum_{k=1}^{6} a_{i,k}\left(4v_{6}^{k}\right)+\sum_{k=7}^{8} a_{i,k}\left(7v_{4}^{k}+6v_{9}^{k}\right) \leq 43 \qquad \text{pour } i = 1,2,3$$

En prenant en compte les tables IV.7, IV.8 et IV.9 nous réécrivons l'inégalité précédente comme suit :

$$4a_{i,2}+4a_{i,5}+7a_{i,7}+13a_{i,8} \leq 43; \qquad (11.5)$$
pour i = 1, 2, 3.

Finalement, nous considérons la ressource M_6. Les transitions modélisant les opérations effectuées par cette ressource sont $<t_9, t_{23}, t_{27}, t_{30}>$, et la contrainte (4) s'écrit :

$$\sum_{k=1}^{6} a_{i,k}\left(2v_{10}^{k}\right)+\sum_{k=7}^{8} a_{i,k}\left(3v_{10}^{k}\right)+\sum_{k=9}^{12} a_{i,k}\left(3v_{3}^{k}+4v_{6}^{k}\right) \leq 43 \qquad \text{pour } i = 1,2,3$$

En prenant en compte les tables IV.7, IV.8 et IV.9 nous réécrivons l'inégalité précédente comme suit :

$$2a_{i,1}+2a_{i,2}+2a_{i,3}+2a_{i,4}+2a_{i,5}+2a_{i,6}+3a_{i,8}+4a_{i,9}+4a_{i,10}+7a_{i,11}+7a_{i,12} \leq 43 \qquad (11.6)$$
pour i = 1, 2, 3.

Les inégalités (11.1) à (11.6) peuvent être résumées comme :

$$\begin{bmatrix} 6 & 3 & 9 & 3 & 0 & 6 & 4 & 0 & 4 & 4 & 0 & 0 \\ 4 & 0 & 0 & 4 & 0 & 0 & 3 & 0 & 4 & 4 & 4 & 4 \\ 0 & 5 & 2 & 0 & 5 & 2 & 0 & 0 & 7 & 5 & 7 & 5 \\ 0 & 0 & 0 & 2 & 2 & 2 & 7 & 7 & 0 & 5 & 0 & 5 \\ 0 & 4 & 0 & 0 & 4 & 0 & 7 & 13 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 2 & 2 & 2 & 0 & 3 & 4 & 4 & 7 & 7 \end{bmatrix} \begin{bmatrix} a_{i,1} \\ a_{i,2} \\ a_{i,3} \\ a_{i,4} \\ a_{i,5} \\ a_{i,6} \\ a_{i,7} \\ a_{i,8} \\ a_{i,9} \\ a_{i,10} \\ a_{i,11} \\ a_{i,12} \end{bmatrix} \leq \begin{bmatrix} 43 \\ 43 \\ 43 \\ 43 \\ 43 \\ 43 \end{bmatrix} \quad (12)$$

pour i = 1, 2, 3.

En considérant les relations (10), le problème *PP1* peut donc être formulé dans le cas de cet exemple par :

$$\begin{cases} \text{MIN } C = 10\sum_{s=1}^{3} y_s^1 + 20\sum_{s=1}^{3} y_s^2 + 40\sum_{s=1}^{3} y_s^3 + 30\sum_{s=1}^{3} z_s^1 + 20\sum_{s=1}^{3} z_s^2 + 20\sum_{s=1}^{3} z_s^3 \\ \text{s.c.} \\ \text{(a) la rélation (12) est vérifiée} \\ \text{(b) } y_s^1 \geq \sum_{i=1}^{s}\left(\sum_{k=1}^{6} a_{i,k} - d_{i,1}\right);\ y_s^2 \geq \sum_{i=1}^{s}\left(\sum_{k=7}^{8} a_{i,k} - d_{i,2}\right);\ y_s^3 \geq \sum_{i=1}^{s}\left(\sum_{k=9}^{12} a_{i,k} - d_{i,3}\right); \\ \quad z_s^1 \geq \sum_{i=1}^{s}\left(d_{i,1} - \sum_{k=1}^{6} a_{i,k}\right);\ z_s^2 \geq \sum_{i=1}^{s}\left(d_{i,2} - \sum_{k=7}^{8} a_{i,k}\right);\ z_s^3 \geq \sum_{i=1}^{s}\left(d_{i,3} - \sum_{k=9}^{12} a_{i,k}\right) \\ \quad s = 1,2,3 \\ \text{(c) } a_{i,k} \geq 0;\ i = 1,2,3,4 \text{ et } k = 1,2,\ldots,12, \text{ sont des entiers} \\ \text{(d) } y_s^j \geq 0 \text{ et } z_s^j \geq 0;\ s = 1,2,3 \text{ et } j = 1,2,3 \text{ sont des entiers} \end{cases}$$

Nous avons résolu ce problème en appliquant l'algorithme qui fournit la solution optimale. Comme résultat nous obtenons les valeurs des coefficients $a_{i,k}$, lesquels représentent le nombre de fois que chaque sous-réseau est activé durant chaque période élémentaire. A partir de cela, on peut aisément déterminer le nombre de fois que chaque transition du modèle est franchie pendant chaque période élémentaire. Ceci est présenté dans la table IV.10.

Périodes	Transitions																
	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}
1	3	3	3	2	2	2	1	1	1	6	1	1	5	5	4	4	4
2	3	3	3	2	2	2	0	0	0	5	3	3	2	2	5	5	5
3	0	0	0	0	0	0	3	3	3	3	0	0	3	3	4	4	4

Périodes	Transitions																
	t_{17}	t_{18}	t_{19}	t_{20}	t_{21}	t_{22}	t_{23}	t_{24}	t_{25}	t_{26}	t_{27}	t_{28}	t_{29}	t_{30}	t_{31}	t_{32}	t_{33}	t_{34}
1	4	4	3	3	3	1	1	1	4	1	3	4	4	4	3	1	4	4
2	5	5	5	5	5	0	0	0	3	1	2	3	3	3	3	0	3	3
3	4	4	4	4	4	0	0	0	4	1	3	4	4	4	3	1	4	4

Table IV.10 : Nombre de franchissements des transitions

La valeur du critère correspondant à la solution obtenue est 40, ce qui signifie qu'il y a stockage ou rupture durant certaines périodes. Les résultats concernant la production effective de chaque produit sont donnés dans la table IV.11 ci-dessous. Nous pouvons constater que la production totale de chaque type de produit correspond à la demande totale pour ce produit.

Période	Produit 1		Produit 2		Produit 3	
	Production	Demande	Production	Demande	Production	Demande
1	6	6	4	4	4	5
2	5	5	5	6	3	2
3	3	3	4	3	4	4

Table IV.11 : Production des produits

Ayant déterminé le nombre de franchissements des transitions, nous pouvons déterminer la charge totale pour chacune des ressources du système dans chaque période élémentaire. Ceci est présenté dans la table IV.12. Durant chaque période élémentaire nous avons une seule machine goulot, dont la charge est, pour les trois périodes, égale à 43, 43 et 39 unités de temps.

Période	Machine					
	M_1	M_2	M_3	M_4	M_5	M_6
1	34	37	38	43	42	40
2	42	39	31	39	43	28
3	38	28	32	39	28	31

Table IV.12 : Charge des machines

c) Ordonnancement

Pour l'ordonnancement des opération définies dans chacun des périodes nous utilisons l'algorithme basé sur le recuit simulé. Etant donné que la planification a été réalisée en considérant une durée de période artificielle de 43 unités de temps, nous disposons ici d'une marge de deux unités de temps pour réaliser l'ordonnancement (n'oublions que la durée réelle est égale à 45 unités de temps). Les résultats concernant le temps total de la production dans chaque période après l'ordonnancement sont :

Période	Temps de fabrication après ordonnancement
1	43
2	43
3	39

Table IV.13 : Durées après ordonnancement

Les résultats ci-dessus montrent d'une part que l'ordonnancement est réalisable, et d'autre part que la solution obtenue durant chaque période correspond bien à la solution optimale, car les durées totales sont identiques aux charges des machines goulots. Les résultats détaillés de l'ordonnancement sont présentés sous la forme de diagrammes de Gantt dans la figure IV.11. Ces diagrammes définissent l'ordre et les instants de passage des différents types de produits sur les ressources. Par exemple, durant la première période la ressource M_1 travaille sur des produits du type 1 dans l'intervalle [0,6], ensuite sur des produits du type 3 dans l'intervalle [6,10], et ainsi de suite.

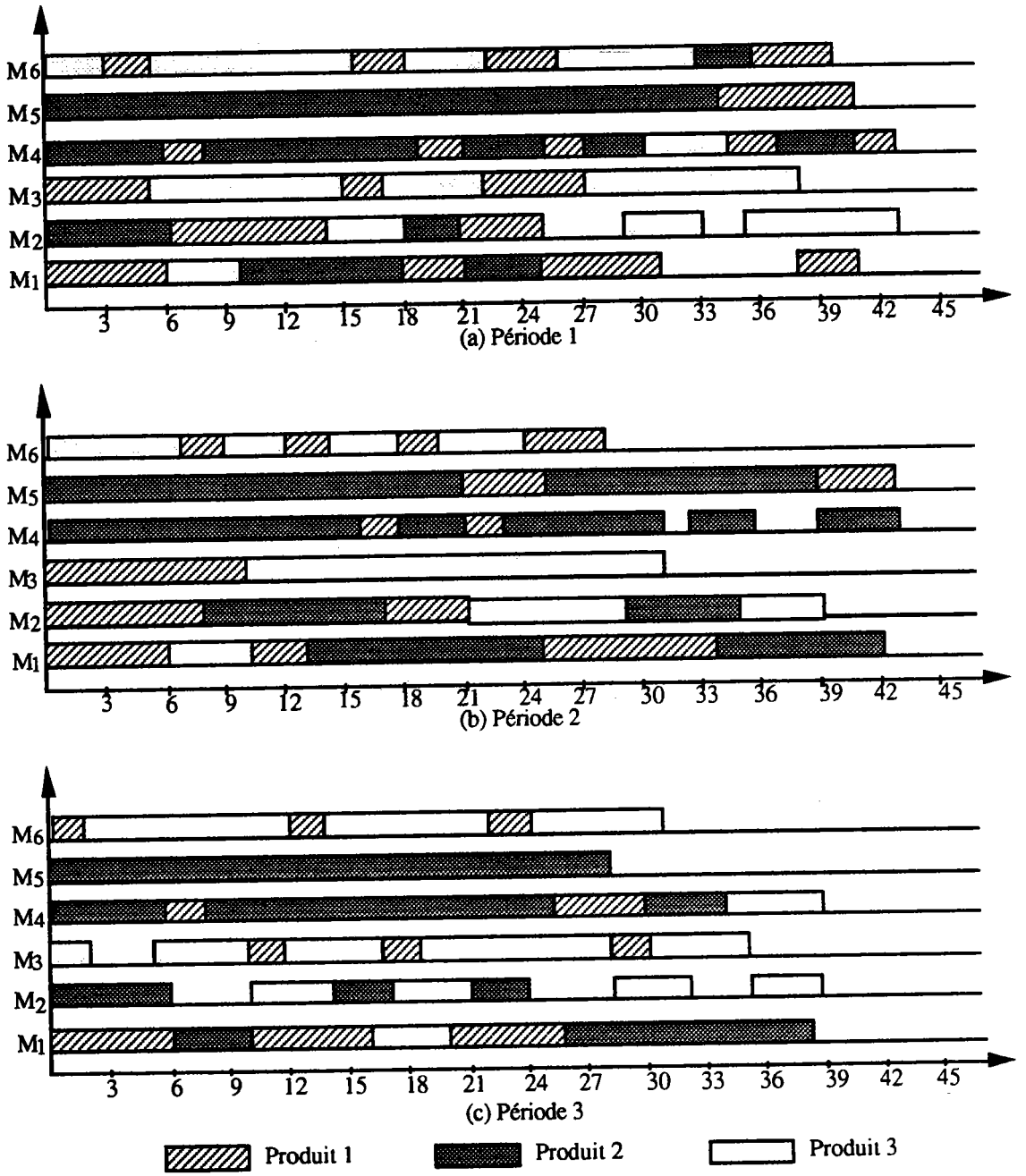


Figure IV.11 : Ordonnancement des périodes

IV.6 Conclusion

Dans ce chapitre nous avons montré comment modéliser et évaluer les systèmes de production non-cycliques. L'évaluation de ces systèmes passe d'abord par la planification à court terme, et ensuite par la détermination d'un ordonnancement du plan de production obtenu au cours de l'étape précédente.

La modélisation de ces systèmes consiste, initialement, en représenter la gamme de fabrication de chaque produit par un RPES. Ce premier modèle sert de support à l'étape de planification. Par la suite, on introduit les places ressources qui permettent d'empêcher l'utilisation simultanée d'une même ressource dans plus d'une opération. Ce nouveau modèle sert de support à l'étape d'ordonnancement.

Pour résoudre le problème de planification nous cherchons à combiner les sous-réseaux sans conflit extraits du modèle RPES de chaque produit, ce qui introduit un certain degré de rigidité dans le fonctionnement du système mais, par contre, permet d'assurer que le modèle possède toutes les propriétés qualitatives qui sont intéressantes du point de vue des systèmes de production. Les sous-réseaux les plus petits qu'on utilise sont ceux dont l'ensemble des transitions est le support d'un t-invariant à support minimal. Cependant, on peut envisager l'utilisation d'autres sous-réseaux obtenus par combinaison de précédents, afin de trouver un compromis entre la qualité des résultats (i.e. la flexibilité opératoire du système) et la complexité (temps) de calcul. Pour augmenter les chances de trouver un ordonnancement du plan de production qui est défini dans cette étape, on réduit artificiellement la durée de la période pendant les calculs.

Deux méthodes d'ordonnancement ont été mises en œuvre : l'une d'elles est basée sur la méthode du recuit simulé et l'autre est basée sur la procédure par séparation et évaluation. Comme nous l'avons montré, la deuxième méthode n'a d'intérêt que dans le cas des modèles de petite taille. Il est important de rappeler que nous nous contentons en général d'obtenir à cette étape une solution réalisable, c'est-à-dire une solution dans laquelle le temps total de fabrication (makespan) est plus petit ou égal à la durée réelle de la période.

La méthodologie présentée dans ce chapitre a été mise en œuvre sous la forme d'un logiciel intégré. La description de ce logiciel, ainsi que la validation des résultats sur une batterie d'exemples, fait l'objet du prochain chapitre.

Chapitre V

Evaluation des algorithmes

V.1 Introduction

Les résultats développés dans le chapitre précédent ont été intégrés dans le logiciel **MASP** (Modélisation et Analyse des Systèmes de Production), lequel permet de modéliser et d'évaluer un système de production non-cyclique. Il a été réalisé en langage C et sa mise en œuvre a été faite sur un ordinateur IBM RS6000. Le choix de cette machine s'impose par le fait que nous faisons appel à la bibliothèque d'optimisation OSL, qui tourne sur cette machine. Une description de la structure interne, ainsi que de l'interface utilisateurs de ce logiciel, est donnée en annexe.

L'objectif de ce chapitre est d'abord de faire une étude des différents algorithmes utilisés pour l'évaluation d'un système de production non-cyclique. Cette étude permettra d'analyser les performances et la qualité des résultats fournis par chaque algorithme. Dans un deuxième temps, nous réalisons une série complète d'expérimentations afin d'établir les tailles maximales des systèmes qui peuvent être traités. Pour la réalisation de chaque étude, nous utilisons le logiciel sur une batterie d'exemples générés au hasard.

Après avoir décrit la façon dont les exemples utilisés dans les différentes études sont générés, nous présentons les résultats de trois études comparatives : les deux premières concernent les algorithmes de résolution du problème de planification (i.e. les problèmes *pp1* et *pp2* présentés dans le chapitre IV), et la dernière concerne les deux algorithmes proposés pour la résolution du problème d'ordonnancement (i.e.l'algorithme basé sur la procédure par séparation et évaluation ainsi que l'algorithme basé sur le recuit simulé). Finalement, dans la dernière partie, nous appliquons la méthodologie développée à un jeu de dix exemples de tailles importantes.

V.2 Définition des exemples utilisés

Notre but final étant d'évaluer dans sa totalité l'application de la méthodologie que nous proposons, nous avons cherché dans la littérature des exemples numériques de résolution de problèmes de planification et d'ordonnancement avec la prise en compte des routages des produits. N'ayant pu trouver de tels exemples, nous avons décidé de tester notre méthodologie sur des exemples que nous avons générés nous mêmes (voir ci-dessus). Afin de garder une certaine cohérence entre les différentes expérimentations que nous menons dans ce chapitre, nous avons décidé de générer tous les exemples utilisés de la même manière.

Pour générer les exemples, nous avons défini quinze structures de gammes de fabrication qui peuvent représenter le processus de fabrication d'un type de produit donné. Les modèles RPES de ces gammes sont donnés dans la figure V.1. Pour obtenir un nouvel exemple de test, nous réalisons les deux étapes suivantes :

(i) nous choisissons le nombre de types de produits ainsi que le nombre de ressources utilisées dans le modèle ;

(ii) nous affectons au hasard une des structures de gamme de la figure V.1 à chaque type de produit. En même temps, nous affectons une ressource et une durée opératoire à chaque opération de la gamme (i.e. à chaque transition autre que les transitions d'entrée et de sortie).

L'affectation d'une ressource à une opération, ainsi que la durée opératoire, sont choisies au hasard. Les durées correspondent, dans notre cas, à des entiers choisis dans un intervalle [1,..., 20] unités de temps. Il est clair que même si le même ensemble de gammes est utilisé pour représenter le processus de fabrication de différents types de produits, il y a de fortes chances que ces modèles soient différents.

Il est important de souligner que tous les RPES représentés dans la figure V.1 sont des réseaux décomposables. Donc, les modèles générés de la manière indiquée ci-dessus sont des modèles décomposables.

V.3 Etudes comparatives

V.3.1 Comparaison des méthodes de planification

Dans la formulation du problème de planification que nous avons présenté dans le chapitre précédent, nous avons considéré deux critères d'optimisation : (i) la minimisation de la somme des coûts de stockage et de rupture des produits et (ii) l'équilibrage des charges des

machines. Ceci nous conduit à résoudre, respectivement, un problème de programmation linéaire en nombres entiers ou un problème de programmation quadratique dans lequel certaines variables sont des entiers. Pour la résolution du problème associé au premier critère nous disposons d'une méthode exacte (i.e. une méthode qui fournit la solution optimale), mais nous avons aussi proposé une heuristique. Le problème associé au deuxième critère n'est résolu qu'en utilisant une heuristique, pour des raisons que nous avons exposées antérieurement.

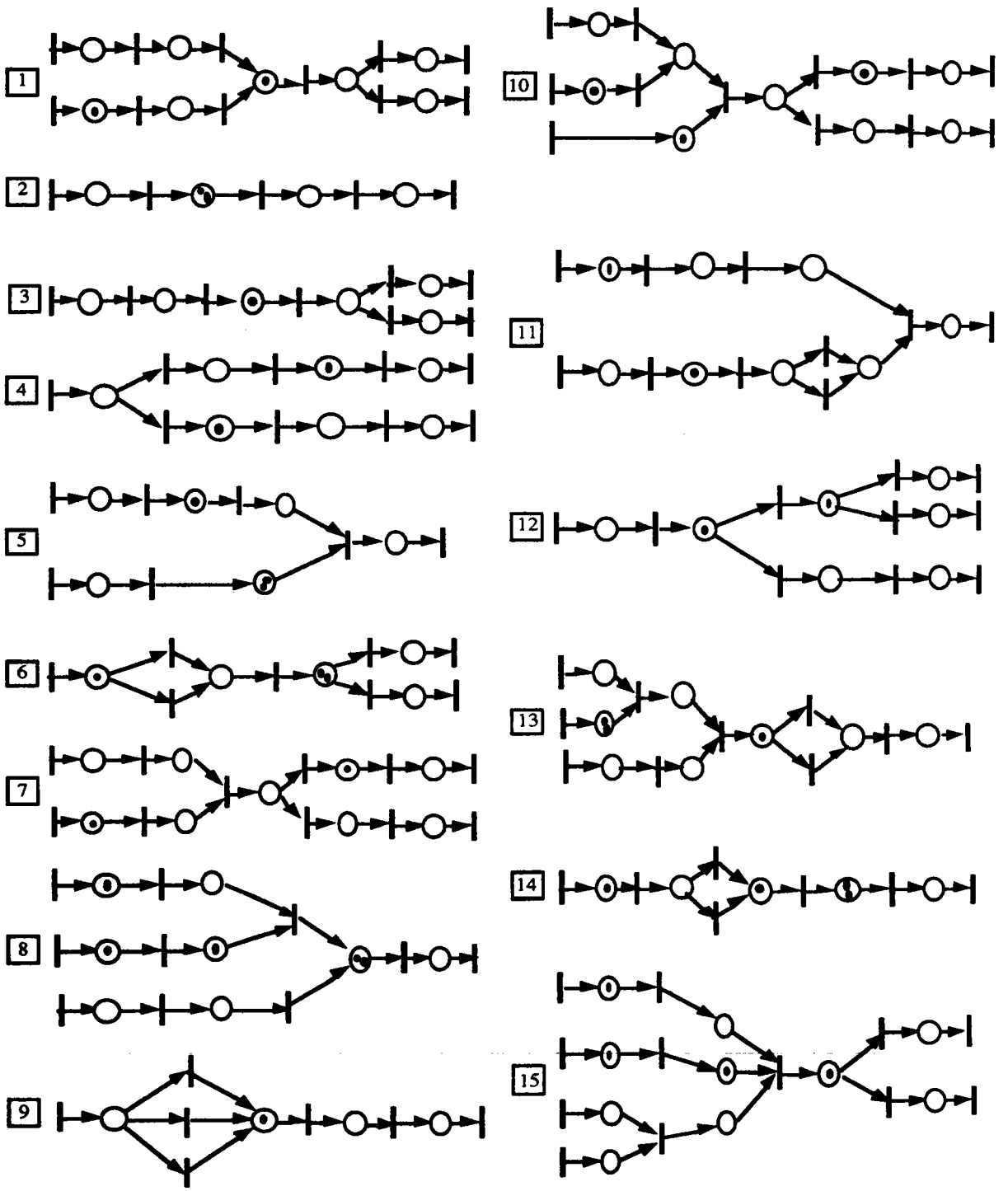


Figure V.1 : Structures des gammes utilisées dans les exemples numériques

Afin de pouvoir juger de la qualité des résultats dans l'un ou l'autre cas, nous faisons maintenant une étude comparative des méthodes de résolution proposées pour chaque problème. Pour ce qui concerne le premier critère, nous comparons les résultats fournis par la méthode exacte avec ceux fournis par l'heuristique. La comparaison dans le cas du deuxième critère est plus difficile, car nous ne disposons pas d'une méthode pour déterminer la solution optimale. Donc, nous nous contentons de comparer la solution fournie à la fin de l'heuristique avec la solution obtenue dans le cas où toutes les variables peuvent prendre des valeurs réelles (cette solution donne une borne inférieure de la solution optimale du problème considéré).

Les données de planification sont choisies au hasard dans les intervalles suivants : [1,..., 15] pour le nombre de produits, [1,..., 10] pour les coûts de stockage, et.[5,..., 20] pour les coûts de rupture. Le nombre de périodes ainsi que la durée de celles-ci, sont définis en fonction du type de problème.

V.3.1.1 Premier critère

Comme nous l'avons déjà indiqué, la détermination de la solution optimale du problème de planification dans ce cas utilise un algorithme basé sur une procédure de type séparation et évaluation. Cet algorithme fait partie de la bibliothèque d'optimisation OSL. Cependant, nous avons pu constater que les temps de calcul liés à l'utilisation de cet algorithme peuvent être très importants. De plus, dans plusieurs cas, l'algorithme n'arrive pas au bout des calculs pour une simple question de saturation de la mémoire. Ceci est une conséquence du nombre de combinaisons possibles des sous-réseaux durant chaque période élémentaire, mais aussi du nombre de périodes considérées. Même pour des problèmes de petite taille la complexité du problème peut atteindre des niveaux élevés. En guise d'exemple, dans un problème à trois produits et dans lequel l'horizon de planification est composé de 5 périodes élémentaires, le modèle de chaque produit étant décomposé en 6 sous-réseaux, nous avons au total 216^5 combinaisons possibles des sous-réseaux. Ceci rend la détermination de la solution impossible dans les cas où il faudrait exploiter une bonne partie de l'arborescence.

Cependant, il existe un cas où on est sûr que l'algorithme exact aboutira à la solution du problème dans des temps raisonnables, même pour les cas où la complexité du problème est importante. Ceci correspond au cas idéal où la durée des périodes est suffisamment large pour satisfaire toutes les demandes prévues durant ces périodes. En effet, la solution en nombres entiers est identique à la solution du problème en nombres réels, ce qui en règle général est calculé de manière rapide. Evidement, dans un tel cas la planification n'a pas de sens.

Donc, pour des questions d'efficacité, nous serons amenés à utiliser l'algorithme

heuristique dans la plupart des cas. L'utilisation de l'algorithme exact reste limité à des problèmes où la combinatoire n'est pas trop importante. Ainsi, dans cette étude nous limitons l'horizon de planification à trois périodes élémentaires, et nous choisissons des exemples pour lesquels le nombre de combinaisons possibles n'est pas trop élevé. Pour cela, nous avons défini deux groupes d'exemples (table V.1) avec les caractéristiques suivantes :.

- le premier groupe (ex. 1 à 5 dans la table V.1) est composé d'exemples de petits tailles qui, par conséquent, ne posent pas de difficulté majeure pour l'utilisation de l'algorithme exact. Les exemples de ce groupe utilisent toutes les structures données dans la figure V.1.

- le deuxième groupe (ex. 6 à 10 dans la table V.1) correspond à des exemples de grande taille, mais pour lesquels toutes les gammes de fabrication des produits sont représentés par la structure numéro 2 de la figure V.1 (ce qui signifie qu'on impose qu'il n'y ait qu'un seul routage par produit). Ceci étant, le nombre de combinaisons est égale au nombre de produits (donc, elle n'est pas très importante), ce qui nous permet de pouvoir traiter ces exemples avec l'algorithme exact.

Type de modèle	1	2	3	4	5	6	7	8	9	10
Nb. de types de Produits	6	7	7	8	9	20	30	40	50	100
Nb. de ressources	6	5	7	5	5	10	10	20	20	20

table V.1 : Caractéristiques des modèles

Nous traitons trois jeux d'exemples. Chaque jeu est composé d'un modèle de chacun des types présentés dans la table V.1 (soit au total trente exemples différents).

Les résultats obtenus sont présentés dans la table V.2, où la colonne "rupture totale" donne la différence (en nombre de produits) entre la somme des demandes et la somme des productions de chaque produit. Donc, à chaque fois qu'une valeur dans cette colonne est non nulle, cela implique la rupture d'un ou de plusieurs types de produits à la fin de l'horizon de planification (i.e. les demandes totales de ces produits n'ont pas pu être satisfaites).

Dans la résolution des problèmes associés à chacun des exemples, nous réduisons le plus possible la durée des périodes afin d'augmenter la difficulté de résolution. Nous nous arrêtons soit lorsque l'algorithme exact n'attend pas la solution suite à la saturation de la mémoire, soit lorsque le temps de calcul est devenu trop important. Dans ce dernier cas, nous interrompons les calculs.

Premier jeu d'exemples							
Ex	**Durée de la Période**	**Valeur du critère**		**Rupture totale**		**Temps de calcul (s)**	
		sol. exacte	sol. heur.	sol. exacte	sol. heur.	sol. exacte	sol. heur.
1	390	58	64	1	4	122,21	0,93
2	570	8	14	2	2	10,87	1,67
3	480	9	9	0	0	9,92	1,89
4	1000	72	72	4	4	600,52	2,54
5	800	29	30	0	0	236,98	2,89
6	700	89	90	5	5	46,57	1,97
7	1300	56	60	9	10	10,82	2,78
8	850	70	80	2	3	202,15	3,52
9	1000	85	88	28	32	122,35	5,88
10	1750	77	77	8	8	336,87	13,79

(a)

Deuxième jeu d'exemples							
Ex	**Durée de la Période**	**Valeur du critère**		**Rupture totale**		**Temps de calcul (s)**	
		sol. exacte	sol. heur.	sol. exacte	sol. heur.	sol. exacte	sol. heur.
1	380	6	11	0	1	2,18	0,95
2	560	3	10	0	0	168,20	1,18
3	400	4	4	0	0	90,57	0,91
4	750	10	17	0	0	3,02	1,04
5	580	0	7	0	0	4,60	1,27
6	820	33	43	5	7	36,64	1,26
7	1140	13	14	1	1	181,72	1,68
8	1050	68	68	13	13	48,12	2,04
9	1060	30	35	1	4	12,08	2,85
10	2200	22	23	9	10	14,79	8,97

(b)

Troisième jeu d'exemples							
Ex	**Durée de la Période**	**Valeur du critère**		**Rupture totale**		**Temps de calcul (s)**	
		sol. exacte	sol. heur.	sol. exacte	sol. heur.	sol. exacte	sol. heur.
1	430	2	9	0	2	27,69	1,12
2	850	3	3	0	0	36,52	0,79
3	500	5	10	0	0	112,97	1,48
4	700	9	12	0	0	14,07	1,20
5	880	4	6	0	1	22,84	0,90
6	900	28	29	8	9	5,27	1,05
7	1200	67	68	7	7	33,50	1,89
8	900	12	12	4	4	8,40	2,13
9	1000	35	36	6	6	18,10	3,13
10	1900	47	47	32	33	19,85	8,15

(c)

table V.2 : Résultats de la résolution du problème ***pp1***

Les valeurs moyennes des résultats présentés dans la table V.2 sont données ci-dessous (table V.3).

Valeurs moyennes						
Durée de la période	**Valeur du critère**		**Rupture totale**		**Temps de calcul (s)**	
	sol. exacte	sol. heur.	sol. exacte	sol. heur.	sol. exacte	sol. heur.
901,40	31,80	34,93	4,83	5,53	85,34	2,72

Table V.3 : Valeurs moyennes des résultats de la table V.2

Pour ce qui concerne la valeur du critère, les résultats présentés dans les tables V.2 et V.3 montrent que la méthode heuristique fournit des résultats satisfaisants. La valeur du critère obtenue en utilisant l'heuristique est en moyenne supérieure de dix pour cent à celle obtenue à l'aide de l'algorithme exact. Des écarts plus importants sont observés dans les cas où la valeur du critère n'est pas trop élevée, ce qui est normal car dans ces cas le fait d'avoir une rupture ou un stockage supplémentaire d'un produit a une conséquence plus importante sur la dégradation de la valeur du critère.

Comme on peut le constater dans la table V.2, les temps des calculs liés à la méthode heuristique sont assez faibles (quelques secondes) par rapport à ceux de l'algorithme exact. De plus, on remarque que dans le cas de la méthode heuristique, ces temps augmentent, en général, de façon proportionnelle à la taille du modèle, tandis que dans le cas de la méthode exacte les temps de calcul ne peuvent être associés à aucune loi particulière car il y a des oscillations brusques d'un cas à l'autre.

On voit bien dans la table V.2 que l'utilisation de la méthode heuristique a tendance à provoquer une rupture moyenne des stocks plus importante que la méthode exacte, comme nous l'avons déjà souligné dans le chapitre précédent. Cependant, nous obtenons toujours des résultats satisfaisants car, dans dix-huit cas sur trente, les résultats des deux méthodes sont identiques, et il n'y a que trois cas où cette différence est supérieure à une unité. Les valeurs moyennes données dans la table V.3 permettent de vérifier que, dans ce cas, l'écart entre les deux méthodes se situe également aux environs de dix pour cent.

V.3.1.2 Deuxième critère

Nous faisons dans cette section une étude de la méthode de résolution du problème *pp2* présenté dans la chapitre IV. Comme nous l'avons dit précédemment, nous cherchons à comparer la solution obtenue à l'aide de notre algorithme avec la solution en nombres réels du

même problème. Nous rappelons que la méthode de résolution mise en place consiste à résoudre d'abord le problème en nombres réels et, à partir de la solution de celui-ci, à déterminer la solution en nombres entiers. Donc, à la fin de l'exécution de l'algorithme, nous disposons de toutes les informations nécessaires à notre étude comparative.

Comme dans le cas précédent, nous utilisons dans cette étude trois jeux de dix exemples ayant les caractéristiques données dans la table V.4. Il existe toutefois une différence importante par rapport aux exemples utilisés précédemment, dans le sens où il n'y a plus de contrainte au niveau de la structure de gamme utilisée pour représenter un produit, i.e. toutes les structures de gammes sont utilisées car on travaille en réels. L'horizon de planification est fixé à cinq périodes élémentaires.

Type de modèle	1	2	3	4	5	6	7	8	9	10
Nb. de types de Produits	6	7	8	9	10	20	30	40	50	60
Nb. de ressources	6	5	5	5	10	10	10	15	15	15

table V.4 : Caractéristiques des modèles

La table V.5 présente les résultats obtenus pour les trois jeux d'exemples, et la table V.6 les valeurs moyennes correspondant à ces résultats.

Dans la résolution des problèmes, nous avons réduit progressivement la durée des périodes jusqu'aux limites à partir desquelles l'algorithme n'arrivait plus à trouver de solution. Il ne faut pas oublier que, dans ce cas, les demandes totales des produits doivent impérativement être satisfaites à la fin de l'horizon de planification, et que si cela n'est pas possible aucune solution n'est fournie. En procédant de cette manière, nous augmentons au maximum la difficulté de résolution du problème ce qui doit produire de plus grands écarts entre la solution réelle et celle obtenue par notre algorithme.

Comme le montrent les résultats de la table V.6, notre méthode heuristique donne des résultats qui sont en moyenne dix pour cent supérieurs à ceux correspondants à la solution réelle. Evidement, cette différence serait plus petite si on avait comparé les résultats avec ceux fournis par une méthode qui donne la solution optimale du problème en nombres entiers.

Premier jeu d'exemples				
Ex	**Durée de la Période**	**Valeur du critère**		**Temps de calcul (s)**
		solution réelle	solution heuristique	
1	420	9,17	9,26	1,93
2	540	2,45	3,04	3,05
3	1100	7,47	7,61	1,87
4	720	2,53	2,66	2,94
5	620	23,10	24,63	29,10
6	840	14,31	15,74	74,80
7	1300	12,49	14,62	167,70
8	1100	25,51	26,30	798,05
9	1400	23,87	25,59	1339,23
10	1500	17,16	19,68	2091,45

(a)

Deuxième jeu d'exemples				
Ex	**Durée de la Période**	**Valeur du critère**		**Temps de calcul (s)**
		solution réelle	solution heuristique	
1	550	20,24	20,45	4,05
2	1050	9,72	9,78	3,02
3	790	3,95	4,45	5,46
4	780	12,38	12,97	5,77
5	550	17,67	18,88	47,81
6	750	10,08	12,54	103,11
7	1200	10,61	12,46	172,81
8	1500	19,56	22,54	758,47
9	1500	6,24	9,42	812,21
10	2100	11,88	14,90	1824,97

(b)

Troisième jeu d'exemples				
Ex	**Durée de la Période**	**Valeur du critère**		**Temps de calcul (s)**
		solution réelle	solution heuristique	
1	450	13,22	13,97	5,43
2	650	5,84	5,98	2,48
3	490	2,21	3,43	4,87
4	650	7,54	8,83	7,34
5	400	28,43	29,96	39,30
6	1000	23,01	23,63	117,40
7	1200	4,40	5,79	209,40
8	1300	15,79	17,76	893,80
9	1400	9,22	12,89	1407,91
10	1900	21,08	23,15	1945,13

(c)

Tables V.5 : Résultats de la résolution du problème ***pp2***

On remarque dans la table V.5 que plus les exemples sont grands, plus l'écart entre la solution réelle et la solution heuristique est important. N'oublions pas que nous pénalisons la fonction objectif pour toute rupture et pour tout stockage des produits durant chacune des périodes, et que cette pénalisation est proportionnelle à la différence entre la production et la demande de chaque produit (voir chapitre 4). Donc, étant donné que dans le cas réel on peut avoir des productions fractionnaires, cette pénalisation risque d'être moins importante que pour le cas de la solution entière, laquelle est obtenue en tronquant la solution réelle. Ainsi, plus on a de produits, plus cette différence a tendance à augmenter.

Valeurs moyennes			
Durée de la période	**Valeur du critère**		**Temps de calcul (s)**
	solution réelle	solution heuristique	
991,66	13,04	14,43	429,36

Table V.6 : Valeurs moyennes des résultats de la table V.5

De même, on vérifie une augmentation importante du temps de calcul pour les derniers exemples de chaque table. Une des raisons de cela est que la préparation des données pour la résolution du problème avec OSL prend beaucoup de temps. En fait, nous avons pu constater que dans le cas des exemples de tailles plus importantes, presque la moitié du temps de calcul est dépensé pour cette préparation. Une autre raison est que l'obtention de la solution entière à partir de la solution réelle demande un certain nombre d'opérations de comparaison qui consomment d'autant plus de temps que le nombre de produits est important. Ces deux points nécessitent certainement d'être améliorés.

V.3.2 Comparaison des méthodes d'ordonnancement

Nous faisons maintenant une étude comparative des performances des deux méthodes utilisées pour la résolution du problème d'ordonnancement, i.e. la méthode basée sur le recuit simulé (par la suite référencée comme **méthode_1**) et la méthode basée sur la procédure par séparation et évaluation (par la suite référencée comme **méthode_2**). Pour pouvoir bien mener cette étude à bien nous allons nous limiter à des exemples de petite taille, car la méthode_2 souffre des mêmes problèmes que la méthode exacte utilisée dans le problème du premier critère de planification.

Nous appliquons les deux méthodes sur un jeu de trente deux exemples différents (huit exemples pour chacun des types données dans la table V.7). Comme on peut le constater dans cette table, nous limitons la taille des exemples à six produits et/ou cinq machines. Cette

limitation est dûe au fait que nous nous sommes aperçu qu'une légère augmentation d'une de ces deux valeurs nous conduisait déjà à des cas où la méthode_2 ne donnait pas de solution.

Type de modèle	1	2	3	4
Nombre de types de Produits	3	4	5	6
Nombre de ressources	5	4	5	4

table V.7 : Caractéristiques des modèles

Les données nécessaires à l'ordonnancement (i.e. le nombre de franchissements de chaque transition du modèle) sont les résultats de la planification. Pour cela, nous considérons un horizon de planification composé de cinq périodes élémentaires. Les demandes ainsi que les coûts de stockage et de rupture sont générés au hasard. Pour chaque exemple utilisé, nous ordonnançons la période pour laquelle la charge moyenne des machines est la plus élevée. Nous avons procédé de cette manière avec l'objectif de rendre la résolution du problème plus compliquée. Néanmoins, comme nous allons le voir dans le tableau des résultats, on n'a pas atteint le but souhaité car les problèmes ont été , en général, simple à résoudre. Ceci est une conséquence du faible nombre de produits et de machines dans chaque modèle.

Nous arrêtons les deux méthodes dans tous les cas où on est sûr que la meilleure solution à été trouvée, i.e. dans le cas où la durée totale obtenue est égale à la charge de la machine goulot de la période que nous ordonnançons. Etant donné que dans certains cas le temps de calcul lié à la méthode_2 peut être très long, nous l'interrompons après un temps raisonnable. Les temps de calcul importants sont dûs au fait que la borne inférieure n'est égale à la valeur optimale du critère que lorsque la solution partielle est presque complète, et pour cela il faut parcourir une bonne partie de l'arborescence avant de trouver la solution optimale. Dans ce cas, nous gardons la meilleure solution trouvée (ces cas sont signalés par une étoile dans la table des résultats).

Les résultats de l'ordonnancement sont donnés dans la table V.8, qui contient les temps de production ("makespan") trouvés et les temps de calcul pour chacune des deux méthodes. Les trois premières colonnes de cette table correspondent à des données de planification, respectivement, la durée de la période élémentaire, la charge moyenne et la charge maximale des machines. Les temps de calcul sont en unités de CPU. Dans la méthode_1 nous avons utilisé les paramètres de contrôle suivants : $T_0 = 100$, $T_f = 0{,}01$, $N_a = 5$, $N_r = 5$, $\Delta T = 0{,}95$. Les résultats correspondent à une seule exécution de la méthode.

Ex	Durée pér.	Charge moyen.	Charge max.	Méthode_1		Méthode_2	
				makespan	temps calcul	makespan	temps calcul
1	300	282,2	296	302	38,05	300 *	26,63 *
2	300	236,4	294	297	43,69	297 *	0,20 *
3	300	221,2	297	297	0,04	297	0,19
4	300	241,0	294	294	0,04	294	0,12
5	300	224,2	290	290	0,40	290	0,16
6	300	208,2	286	296	27,04	296	0,95
7	300	188,8	286	286	0,09	286	0,14
8	300	163,6	299	299	0,36	299	1,45

(a) résultats pour les exemples de type 1

Ex	Durée pér.	Charge moyen.	Charge max.	Méthode_1		Méthode_2	
				makespan	temps calcul	makespan	temps calcul
1	300	291,5	299	299	0,51	299	0,38
2	300	278,3	299	299	2,25	299	1736,50
3	300	156,5	299	299	0,10	299	0,13
4	300	257,7	299	305	38,53	303 *	0,27 *
5	300	232,2	294	296	28,28	296	0,81
6	300	258,7	300	300	4,15	300	382,42
7	300	228,5	297	302	28,41	302 *	0,26 *
8	300	259,2	298	298	0,51	298	0,39

(b) résultats pour les exemples de type 2

Ex	Durée pér.	Charge moyen.	Charge max.	Méthode_1		Méthode_2	
				makespan	temps calcul	makespan	temps calcul
1	320	251,4	318	318	8,60	318	282,76
2	320	272,2	317	317	0,64	317	0,39
3	320	308,4	313	313	0,18	313	211,97
4	320	238,2	318	318	45,04	319 *	0,29 *
5	320	257,2	320	320	0,52	320	0,54
6	320	274,0	318	318	3,50	319 *	2312,52 *
7	320	258,0	308	308	0,30	308	0,39
8	320	296,0	312	316	60,97	326 *	2450,62 *

(c) résultats pour les exemples de type 3

Ex	Durée pér.	Charge moyen.	Charge max.	Méthode_1		Méthode_2	
				makespan	temps calcul	makespan	temps calcul
1	550	444,7	542	542	21,83	582 *	774,57 *
2	550	521,7	550	550	1,38	550	2,56
3	550	506,7	540	540	0,10	540	0,63
4	550	543,0	550	550	3,95	550	1,93
5	550	529,2	542	542	0,57	542	2,48
6	550	496,5	549	556	73,36	553	76,34
7	550	535,0	550	550	4,17	550	14,39
8	550	421,5	546	546	0,29	546	2,09

(d) résultats pour les exemples de type 4

Table V.8 : Résultats de l'ordonnancement

Comme nous pouvons le constater dans la table V.8, dans la presque totalité des cas la méthode_1 obtient les mêmes valeurs pour le "makespan" que la méthode_2, et il n'y a que deux cas où les résultats de la méthode_2 sont meilleurs. Pour ce qui concerne les temps des calculs, on constate que ceux-ci peuvent être assez longs dans le cas de la méthode_2, même quand on considère des petits exemples. Cependant, on ne peut pas dire qu'une méthode soit supérieure à l'autre, car dans la moitié des cas la méthode_1 est la plus performante, et dans l'autre moitié c'est la méthode_2 qui l'est. Ceci étant, on peut conclure que pour des petits exemples l'utilisation d'une ou de l'autre méthode est sans conséquence. Evidement pour des exemples plus importants nous serons contraints à utiliser seulement la méthode_1.

Il est important de souligner ici que normalement on n'effectue que l'ordonnancement sur la première période, car on se place dans le contexte d'un *horizon glissant*. Dans ce cas, on recalcule la planification à chaque début de période en prenant en compte les demandes qui n'ont pas pu être satisfaites dans la période précédente et les nouvelles demandes. Pour cela, on rajoute à chaque fois une période supplémentaire à la fin de l'horizon de planification de façon à ce que celui-ci soit toujours constant.

V.4 Applications numériques

Nous consacrons cette section à une évaluation complète (planification plus ordonnancement) d'un jeu de dix exemples de tailles importantes. Cette évaluation va nous permettre de vérifier les performances des algorithmes dans des cas généraux. Au niveau de la planification nous considérons comme critère d'optimisation la somme des coûts de stockage et de rupture, et nous utilisons la méthode heuristique pour la résolution du problème. Pour l'ordonnancement de chacune des périodes nous utilisons la méthode basée sur le recuit simulé.

Nous supposons que l'horizon de planification pour chacun des exemples considérés est composé par cinq périodes élémentaires. Pour chaque exemple, nous avons choisi une durée de période élémentaire suffisamment petite pour saturer la machine la plus lente. Nous considérons aussi que l'ordonnancement d'une période est un succès si la durée totale après l'ordonnancement ne dépasse pas une quantité dix pour cent supérieure à la durée de période considérée au niveau de la planification.

Les demandes de chaque type de produit ainsi que leurs coûts de stockage et de rupture sont aussi tirés au hasard selon les intervalles suivants :

demandes : [1, 15] ;
coûts : [1, 10].

La table V.9 donne les caractéristiques des dix exemples tests que nous utilisons, ainsi que le nombre moyen d'opérations dans chacune des gammes de fabrication qui sont utilisées dans chaque exemple. Dans la table V.10 nous détaillons ces exemples en présentant le nombre de fois que chaque type de structure de gamme de la figure V.1 apparaît dans chacun des exemples.

Exemple	1	2	3	4	5	6	7	8	9	10
Nb de produits	10	15	20	25	30	35	40	45	50	60
Nb de ressources	10	10	10	10	10	15	15	15	15	20
Longueur moyenne de la gamme	5,30	5,93	5,40	5,36	5,40	5,48	5,55	5,75	5,70	5,93

Table V.9 : Définition des exemples

Exemple	**Numéro de la structure de gamme**														
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	1	2	0	1	0	1	0	1	1	0	1	0	0	1	1
2	0	1	0	0	1	1	3	5	0	1	1	1	1	0	0
3	1	3	0	1	3	0	0	3	0	0	2	2	3	0	2
4	2	3	0	1	1	3	3	3	0	2	0	0	2	3	2
5	1	2	1	3	3	3	1	4	4	1	1	1	3	2	0
6	3	4	2	1	3	4	3	1	3	2	1	3	0	2	3
7	1	6	5	4	2	3	1	1	1	5	4	1	1	3	2
8	4	2	2	0	1	6	3	5	5	4	3	3	3	4	0
9	3	3	3	3	2	6	4	4	3	4	1	3	5	2	4
10	4	1	7	3	1	3	5	3	5	6	4	5	4	3	6

Table V.10 : Nombre de fois que les structures apparaissent dans les exemples

La table V.11 donne les résultats de la planification. Dans cette table, nous donnons la durée de période utilisée, la valeur du critère, la charge moyenne de la machine goulot dans les cinq périodes considérés, et le temps des calculs. Toutes les unités de temps sont en secondes. Comme nous pouvons le constater dans cette table, les temps de calcul augmentent de façon significative quand on passe du premier au dernier exemple. Cependant, ces temps sont encore

raisonnables pour la taille des problèmes auxquels ils correspondent.

Néanmoins, deux facteurs peuvent influencer directement les temps de résolution du problème de planification associé à un exemple donné : (i) le nombre de périodes qui composent l'horizon de planification, et (ii) la durée de chaque période. En modifiant l'un ou l'autre de ces deux paramètres on peut rendre la résolution du problème de planification plus ou moins difficile, ce qui par conséquent donnera des temps de calcul différents. On a pu constater pour l'ensemble des exemples utilisés, que seul le changement de la durée de la période peut provoquer des différences importantes dans la durée des calculs. En guise d'exemple, si l'on fixe une durée de période égale à 3000 pour l'exemple numéro dix de la table V.11, le temps de calcul est réduit à 48 s au lieu des 123 s obtenus avec une durée égale à 1270.

PLANIFICATION				
Ex.	**Durée de la période**	**Valeur du critère**	**Charge moyenne de la machine goulot**	**Temps de calcul**
1	710	2	648,8	3
2	870	19	864,0	4
3	830	20	827,2	9
4	950	31	948,2	21
5	1300	237	1293,2	20
6	1000	27	994,8	29
7	1200	3	1197,6	28
8	1050	44	1048,6	89
9	1400	21	1396,4	48
10	1270	17	1268,8	123

Table V.11 : Résultats de la planification

Les résultats obtenus dans l'ordonnancement sont présentés dans la table V.12. Nous n'avons donné dans cette table que le résultat de l'ordonnancement de la première période, car

comme nous l'avons expliqué précédemment, on est dans le cas d'un horizon glissant. On constate que dans certains cas les temps de calcul sont assez faibles, tandis que dans d'autres cas ces temps deviennent très importants. Ceci est dû au fait que nous interrompons l'exécution de la méthode d'ordonnancement dès que la meilleure solution correspondant à la période en question a été trouvée. Donc, cette différence de temps s'explique par le fait que dans certains cas la solution a été trouvée presque toute de suite et que, dans d'autres cas, il aura fallu aller jusqu'à au bout du processus du recuit simulé pour déterminer cette solution.

Nous tenons à préciser que pour la plupart des exemples la valeur du "makespan" est identique à la charge de la machine goulot dans la période considérée. Aussi, tous les résultats présentés dans la table V.12 correspondent aux résultats obtenus dans la première utilisation de l'algorithme d'ordonnancement, car ces résultats étaient déjà plus petits que la durée maximale admissible.

On a pu constater une nette détérioration des performances de la méthode basée sur le recuit simulé avec l'accroissement de la taille du problème (voir, par exemple, l'exemple numéro huit). Ceci n'est pas surprenant compte tenu du fait que dans cette méthode nous utilisons la simulation du modèle comme base de la détermination d'un ordonnancement. Donc, il est normal d'avoir des temps de calcul plus importants au fur et à mesure que la taille du modèle croît. Il ne faut pas oublier que le temps de calcul dépend aussi des valeurs des paramètres utilisés dans le recuit simulé, lesquels déterminent le nombre total de simulations qui seront réalisées dans chaque utilisation de la méthode.

Il est important de rappeler que nous nous contentons, en général, d'obtenir une solution réalisable. Cependant, afin de pouvoir faire une analyse correcte des performances de la méthode d'ordonnancement, nous avons laissé l'algorithme se dérouler jusqu'à son arrêt naturel.

Les résultats que nous venons de présenter pour l'ensemble des exemples montrent bien les points faibles et aussi les limitations des algorithmes utilisés. Néanmoins, il est difficile de fixer des limites précises concernant l'application de l'ensemble des algorithmes, car il existe plusieurs facteurs qui peuvent modifier leurs performances. Nous avons vu que du côté de la planification l'algorithme est efficace et on peut envisager le traitement de problèmes de plus grande taille que ceux qui sont utilisés dans notre étude. La principale limitation vient donc de l'ordonnancement, où on risque d'avoir des temps de calcul assez importants.

ORDONNANCEMENT		
Exemple	**Durée de l'ordonnancement (makespan) de la première période**	**Temps de calcul (s)**
1	479	1
2	864	1
3	829	844
4	996	29
5	1294	2
6	996	40
7	1200	356
8	1050	2902
9	1400	5
10	1270	845

Table V.12 : Résultats de l'ordonnancement

V.5 Conclusion

Dans ce chapitre, nous avons fait une étude des performances des différents algorithmes utilisés pour l'évaluation du modèle. L'objectif a été d'une part de comparer les résultats fournis pour chacun d'eux, et d'autre part d'établir les limites concernant leur application.

Nous avons vu qu'au niveau de la planification les méthodes heuristiques proposées donnent des résultats satisfaisants avec des temps de calcul assez faibles, et donc qu'elles sont applicables à des problèmes de taille importante. Les limitations se situent surtout à niveau de l'ordonnancement. La seule méthode possible dans le cas des problèmes de grandes tailles est celle basée sur le recuit simulé.

Chapitre VI

Conclusion générale

VI.1 Description du travail

Comme nous l'avons vu dans le chapitre I, on peut séparer les systèmes de production selon la manière dont la production est organisée en deux types : les systèmes qui fonctionnent de manière **cyclique** (répétitive) et les systèmes qui fonctionnent de manière **non-cyclique** (non-répétitive). De nombreux travaux ont été consacrés à l'utilisation des réseaux de Petri dans la modélisation, l'analyse et l'évaluation des systèmes de production cycliques. Comme nous l'avons montré dans le chapitre III, ces systèmes peuvent être modélisés à l'aide des graphes d'événements, dont les propriétés ont permis de dégager des résultats importants concernant le fonctionnement des systèmes qu'ils modélisent. En particulier, les résultats obtenus pour ces systèmes, tant en ce qui concerne le cas déterministe ou le cas stochastique, permettent de calculer et d'optimiser les performances du système dès qu'un ordonnancement est donné. Le problème de détermination de l'ordonnancement est toujours NP-difficile pour ces systèmes, mais des heuristiques ont été développées qui permettent d'obtenir un bon ordonnancement.

Nous nous sommes intéressés, dans cette thèse, à l'utilisation des réseaux de Petri pour *la modélisation et l'évaluation des systèmes de production non-cycliques*. A cet égard, l'intérêt essentiel de notre démarche est d'avoir pu proposer une méthodologie fondée sur l'utilisation d'un même type de modèle à toutes les étapes de l'étude, et d'avoir développé des résultats qui sont entièrement basés sur la structure des modèles physiques. Nous avons abouti au développement d'une méthodologie efficace et peu contraignante de modélisation et d'évaluation des systèmes non-cycliques qui, en plus, permet de garantir certaines propriétés qualitatives du modèle . Comme nous l'avons vu, l'évaluation du modèle passe dans ce cas par réalisation d'une planification à court terme et d'un ordonnancement.

Un premier travail a consisté à définir et à étudier les propriétés qualitatives d'une nouvelle classe des réseaux de Petri : les *RPES* (Réseaux de Petri avec des transitions d'Entrée et de Sortie). Dans cette étude, nous nous sommes concentrés sur les propriétés qualitatives qui sont demandées lorsque des systèmes de production sont concernés. En particulier, nous avons donné les conditions pour que le RPES soit vivant, borné et réversible. Cette nouvelle classe permet de modéliser le fonctionnement d'un système de production non-cyclique. Une étude approfondie a permis par la suite de définir les caractéristiques que doivent vérifier les modèles RPES pour qu'on puisse, d'une part, utiliser ce modèle dans la résolution des problèmes de planification et d'ordonnancement, et d'autre part, assurer les bonnes propriétés qualitatives du modèle. Ceci a conduit à la définition et à l'utilisation des RPES qui sont décomposables en des sous-réseaux sans conflit (RPES sans conflit).

En ce qui concerne la modélisation, notre démarche repose sur deux hypothèses essentielles : (i) le système est décrit à partir de la définition des *gammes de fabrication* de chaque type de produit, (ii) les durées opératoires sont *déterministes*.

En ce qui concerne l'évaluation du modèle, nous avons d'abord montré comment le problème de planification à court-terme est résolu en combinant l'ensemble des sous-réseaux extraits du modèle RPES représentant le système de production. L'approche que nous avons utilisée pour résoudre ce problème impose une certaine contrainte au niveau de la flexibilité opératoire du système, mais en contrepartie permet de contrôler facilement le système. Pour la résolution du problème de planification, nous avons montré que l'utilisation des méthodes heuristiques que nous avons mises en place fournit des résultats satisfaisants. Dans le cas de l'ordonnancement, deux méthodes heuristiques ont été mises en place, lesquelles sont basées sur la structure des modèles réseaux de Petri. Cependant, en pratique, nous avons pu constater que seulement l'une d'entre elles peut être utilisée dans le cas général, car l'autre ne peut être appliquée qu'à des problèmes de petite taille.

Une série d'expériences nous ont permis d'étudier les performances de toutes les méthodes (exactes et heuristiques) que nous avons utilisées au cours des étapes de ce travail. En plus, nous avons réalisé une évaluation complète d'un batterie d'exemples ayant des tailles importantes afin de pouvoir tirer quelques conclusions concernant les limites pratiques d'application des résultats.

Enfin, comme produit final de ce travail, nous avons mis en place un logiciel dans lequel nous avons intégré tous les algorithmes que nous avons développés dans le cadre de cette thèse. Afin de faciliter l'utilisation de ces algorithmes, ce logiciel a été habillé d'une interface basée sur un système de fenêtrage, ce qui rends son utilisation très agréable.

VI.2 Directions de recherche

Par la suite, nous proposons deux axes de recherche que nous jugeons intéressants d'approfondir dans la suite de ce travail. Nous précisons que ces suggestions portent sur des points précis de notre étude, et non sur la continuation de la mise en œuvre de la méthodologie générale de modélisation des systèmes de production que nous avons présentée dans le chapitre I. Cette méthodologie est en cours de développement.

Comme nous l'avons vu dans le chapitre V, ce sont les méthodes d'ordonnancement qui induisent les limitations les plus importantes pour la taille des modèles à traiter. En d'autres termes, nous avons pu constater à ce moment là que les temps de calcul de la méthode d'ordonnancement utilisée (i.e. celle basée sur le recuit simulé) deviennent importants quand on considère des exemples où le nombre de produits est significatif. Donc, si l'on a le soucis d'utiliser la méthodologie développée dans des cas réels où le nombre de produits peut être très important, il est nécessaire de développer nouvelles heuristiques plus performantes au niveau de l'ordonnancement qui tirent un meilleur profit de la structure des réseaux de Petri. Dans ce sens, nous pensons que l'approche basée sur la détermination d'un chemin critique que nous avons introduite dans le chapitre IV donne une bonne direction de recherche.

Nous avons basé notre étude sur l'utilisation d'un modèle simplifié du système, lequel est dérivé de la structure des gammes de fabrication. Ce modèle, même sous une forme simplifiée, à le mérite de préserver les caractéristiques fondamentales du fonctionnement du système. Une étude théorique reste donc indispensable afin de pouvoir considérer un modèle plus détaillé, et dans ce sens, notre deuxième suggestion concerne la prise en compte dans le modèle des moyens de transport. Il est clair que rien que cela entraîne une difficulté supplémentaire importante, laquelle est la présence de boucles dans le modèle.

Nous concluons en disant que nous pensons avoir montré que notre approche est efficace. De plus, nous croyons que la méthodologie développée va dans le sens des pratiques couramment utilisées dans le milieu industriel, et donc que logiciel mis en place peut trouver des débouchés dans les entreprises.

Bibliographie

[Agerwala et Choed-Amphai 78] : T. Agerwala and Y. Choed-Amphai. *"A Synthesis Rule for Concurrent Systems"*, Proc. 15th Design Automation Conference, Las Vegas, pp. 305-311, 1978.

[Alaiwan et Toudic 85] : H. Alaiwan and J.M. Toudic. *"Recherche des Semi-Flots, des Verrous et des Trappes dans les Réseaux de Petri"*, Technique et Sciences Informatiques, vol.4, n°1, Dunod, France, 1985.

[Baceli *et al.* 89] : F. Baccelli, N. Bambos and J. Walrand. *"Flow Analysis of Stochastic Marked Graphs"*, Proceedings CDC 1989.

[Berthelot 83] : G. Berthelot. *"Transformation et Analyse de Réseaux de Petri"*, Thèse d'Etat, Université Paris VI, Paris, 1983.

[Berthelot 85] : G. Berthelot. *"Checking properties of Nets Using Transformation"*, Lecture Notes in Computer Science - Advances in Petri Nets 1985, Springer-Verlag, pp 19-40, 1985.

[Brams 83] : G.W. Brams. *"Réseaux de Petri : Théorie et Pratique"*, Masson, Paris, France, 1983.

[Brauer *et al.* 90] : W. Brauer, R. Gold and W. Vogler. *"A Survey of Behavior and Equivalence Preserving Refinements of Petri Nets"*, Lecture Notes in Computer Science n° 483, Advances in Petri Nets 1990, pp.1-46,Springer-Verlag, 1990

[Buzzacott et Yao 86] : J. A. Buzzacott and D. D. Yao. *"On Quauing Network Models of Flexible Manufacturing Systems"*, Queuing System Theory and Application, vol. 1, 1986.

[Chretienne 83] : P. Chretienne. *"Les Réseaux de Petri Temporisés"*, Thèse d'Etat, Université Pierre et Marie Curie (Paris VI), Paris, Juin 1983.

[Chretienne 84] : P. Chretienne. *"Exécutions Contrôlées de Réseaux de Petri Temporisés"*, Technique et Sciences Informatiques, vol. 3, n°1, 1984.

[Claver *et al.* 91] : J. F. Claver, G. Harhalakis, J-M. Proth, V. M. Savi and X. XIE. "*A Step-wise Specification of a Manufacturing System Using Petri Nets*", IEEE conference on Systems, Man and Cybernetics, Charlotte, Virginia , October 1991.

[Colom et Silva 91] : J.M. Colom et M. Silva. "*Convex Geometry and Semiflows in P/T Nets*", Advances in Petri Nets 90, Lecture Notes in Computer Science n° 483, Springer-Verlag, Berlin, 1991.

[Commoner *et al* .71] : F. Commoner, A. Holt, S. Even and A.Pnueli. "*Marked directed Graphs*", Journal of Computer and System Science, vol. 5, n°5, 1971.

[David et Alla 89] : R. David et H. Alla. "*Du Grafcet aux Réseaux de Petri*", Edition Hermès, Paris, 1989.

[Dicesare *et al.* 93] : F. Decesare, G. Harhalakis, J.M. Proth, M. Silva et F. Vernadat. "*Practise of Petri Nets in Manufacturing*", Chapman & Hall, London, 1993.

[French 82] : S. French. "*Sequencing and scheduling: an introduction to the mathematics of the job-shop*", John Wiley & Sons, New York, 1982.

[Gross et Harris 74] : D. Gross and C. M. Harris. "*Fundamentals of Queing Theory*", John Wiley and Sons, NY, 1974.

[Harhalakis *et al.* 92] : G. Harhalakis, J-M. Proth, V. M. Savi and F. Vernadat. "*A Botton-Up Approach for the Design of Discrete Manufacturing Systems*", Rensselaer's Third International Conference on Computer Integrated Manufacturing, New York, May 1992.

[Hillion 86] : H.P. Hillion. "*Performance Evaluation of Decision Making Organisations Using Timed Petri Nets*", SM Thesis, report n° TH-1590, LIDS, MIT, Cambridge, MA, 1986.

[Hillion et Proth 88] : H.P. Hillion et J.M Proth. "*Analyse de Fabrication Non Linéaire et Répétitives à L'aide des Graphes d'Evénements Témporisés*", RAIRO, vol 22, n° 2, septembre 1989.

[Hillion et Proth 89] : H.P. Hillion et J.M Proth. "*Performance Evaluation of Job-shop Systems using Timed Event Graphs*", IEEE Trans. on Automatic Control, vol. 34, n°1 , janvier 1989.

[Hillion 89] : H.P. Hillion "*Modélisation et Analyse des Systèmes de Production Discrets par les Réseaux de Petri Temporisés*", Thèse , Université Paris VI, paris, Janvier, 1989.

[Hollocs 91] : B. W. Hollocs. "*Improving Manufacturing Operations with WITNESS computer Simulation*", AT&T Technology, vol 6, n° 1, pp 16-21, 1991.

[Holloway et Krogh 90] : L. E. Holloway and B. H. Krogh. "*Synthesis of Feedback Control Logic for a Class of Controlled Petri Nets*", IEEE Trans. on Automatic Control, vol. 35, n° 5, pp. 514-523, May 1990.

[Ichikawa et Hiraishi 88] : A. Ichikawa and K. Hiraishi. "*Analysis and Control of Discrete Event Systems Represented by Petri Nets*", Lecture Notes in Control and Information Science, vol. 103, pp. 115-134, 1988.

[Jeng et DiCesari 90] : M. D. Jeng and F. DiCesari. *"A Review of Synthesis Techniques for Petri Nets"*, Rensselaer's Second International Conference on Computer Integrated Manufacturing, pp 348-355, Troy, NY, May 1990.

[Kirkpatrick *et al.* 83] : C.D. Kirkpatrick, Gellat Jr. and M.P. Vecchi. *"Optimisation by simulated annealing"*, Science, n° 4598, vol. 220, pp. 671-680, 1983.

[Krogh et Beck 86] : B. H. Krogh and C.L. Beck. *"Synthesis of Place/Transition Nets for Simulation and Control of Manufacturing Systems"*, Proc. IFAC/IFORS Symposium Large Scale Systems, Zurich, 1986.

[Krogh 87] : B. H. Krogh. *"Controlled Petri Nets and Maximally Permissive Feedback Logic"*, Proc. of the 25th Annual Allerton Conference on Communication, Control and Computing, pp. 317-326, Monticello, IL, September 1987.

[Laarhoven et Aarts 87] : P.M.J. Laarhoven and E.H.L. Aarts. *"Simulated Annealing : Theory and applications"*, D. Reidel Publishing Company, Holland, 1987.

[Laftit 91] : S. Laftit. *"Graphes D'Évènements Déterministes et Stochastiques : Application aux Systèmes de Production"*, Thèse de doctorat, Université Paris IX Dauphine, Septembre 1991.

[Laftit *et al.* 92] : S. Laftit, J.M. Proth and X. Xie. *"Optimisation of Invariant Criteria for Event Graphs"*, IEEE Transactions on Automatic Control, vol 37, n° 5, May 1992.

[Lee *et al.* 87] : H. Lee-Kwang, J. Favrel and P. Baptiste. *"Generalized Petri Net Reduction Method"*, IEEE Trans. on Systems, Man and Cybernetics, vol. 17, n° 2, pp. 297-303, March/April 1987.

[Lee et DiCesari 92] : D.Y. Lee et F. DiCesari. *"FMS scheduling using Petri nets and heuristic search"*, Proceedings of the 1992 IEEE international conference on robotics and automation, Nice, France, May 1992.

[Lautenbach 86] : K. Lautenbach. *"Linear Algebraic Techniques for place/Transions Nets"*, Net Theory and Application, Lecture Notes in Computer Science n°254, Springer Verlag, 1986.

[Martinez et Silva 85] : J. Martinez et M. Silva. *" A Simple and Fast Algorithm to obtain all invariants of a Generalized Petri Net"*, Lecture Notes in Computer Science n° 52, Springer-Verlag, Berlin, FRG, 1985.

[Memmi et Roucariol 80] : G. Memmi et G. Roucairol. *"Linear Algebra in Net Theory"*, Net Theory and Application, Lecture Notes in Computer Science n°84, Springer Verlag, 1980.

[Murata 89] : T. Murata. *"Petri Nets: Properties, Analysis and Applications"*, Proceedings of the IEEE, vol. 77, n° 4, pp. 541-580, April 1989.

[Murgiano 90] : C. Murgiano. *"A Tutorial on Witness"*, Proceedings of the 1990 Winter Simulation Conference, pp. 177-186, 9-12 December, New Orleans, LA, USA, 1990.

[O'Reilly et Ryan 92] : J. J. O'Reilly and N. K. Ryan. *"Introduction to SLAM II and SLAMSYSTEM"*, Proceedings of the 1992 Winter Simulation Conference, pp. 352-358, 13-16 December, Arlington, VA, USA, 1992.

[Ousterhout 93] : J.K. Ousterhout. *"An introduction to TCL and TK"*, Addison-Wesley, NY, 1993.

[Pegden *et al.* 90] : C. D. Pegden, R. E. Shannon and R. P. Sadowski. *"Introduction to Simulation with Siman"*, Macgraw-Hill, NJ, 1990.

[Peterson 81] : J. L. Peterson. *"Petri Net Theory and the Modeling of Systems"*, Prentice-hall, Englewood Cliffs, NJ, 1981.

[Petri 62] : C.A. Petri. *"Kommunikation mit Automaten"*, Bonn Institut fur Instrumentelle Mathematik, Schriften des IIM Nr 3, 1962.

[Proth et Savi 92] : J-M. Proth and V.M. Savi. *"A Modular Petri Net Approach for Modeling Complex Manufacturing Systems"*, 8th International Conference on CAD/CAM, Robotics and Factories of the Future, Metz, France, August 1992.

[Proth et Xie 93] : J.M Proth and X.L Xie *"Performance Evaluation and Optimisation of Stochastic timed Event Graphs"*, IEEE transactions on Automatic Control, 1993.

[Proth *et al* 94] : J.M Proth, L. Wang and X.L Xie *"A Class of Petri Nets for Manufacturing System Integration"*, soumis à publication.

[Ramchandani 74]: C. Ramchandani *"Analysis of Asynchronous Concurrent Systems Using Petri Nets"*, Technical Report n° 120, Laboratory of Computer Science, MIT, Cambridge, MA, 1974.

[Ramamoorthy et Ho 80]: C.V. Ramamoorthy et G.S. Ho *"Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets"*, IEEE Trans. on Software Engineering, vol. SE-6, n°5, 1980.

[Shih et Sekiguchi 91] : H. Shih and T. Sekiguchi. *"A timed Petri net ans beam search based on-line FMS scheduling with routing flexibility"*, Proceedings of the 1991 IEEE international conference on robotics and automation, pp. 2548-2553, Sacramento, CA, April 1991.

[Silva et Valette 90] : M. Silva and R. Valette. *"Petri Nets and Flexible Manufacturing"*, Advances in Petri Nets 89 (G. Rozenberg, ed.), LNCS 424, pp. 375-417, Springer-Verlag, Berlin, 1990.

[Suzuki et Murata 83] : I. Suzuki and T. Murata. *"A Method for Stepwise Refinement and Abstraction of Petri Nets"*, Journal of Computer and System Sciences, n° 27, pp. 51-73, 1983.

[Ushio 90] : T. Ushio. *"Maximally Permissive Feedback and Modular Control Synthesis in Petri Nets with External Input Places"*, IEEE Trans. on Automatic Control, vol. 35, n° 7, pp. 844-848, July 1990.

[Valavanis 90] : K. P. Valavanis. *"On the Hierarchical Modeling Analysis and Simulation of Flexible Manufacturing Systems with Extended Petri Nets"*, IEEE Trans. on Systems, Man and Cybernetics, vol. 20, n° 1, pp. 94-110, January/February 1990.

[Valette 79] : R. Valette. *"Analysis of Petri Nets by Stepwise Refinements"*, Journal of Computer and System Sciences, n° 18, pp. 35-46, 1979.

[Villarroel *et al.* 88] : J.L. Villarroel, J. Martinez and M. Silva. *"GRAMAN: a graphic System for Manufacturing Systems Design"*, IMACS international Symposium on System Modeling and Simulation, Cetraro, Italy, 1988.

[Zhou *et al.* 89] : M. Zhou, F. DiCesari and A. Desroches. *"A Top-Down Approach to Systematic Synthesis of Petri Nets Models for Manufacturing Systems"*, Proceedings of the IEEE Robotics and Automation Conference, pp 534-539, Scottsdale, Arizona, May 1989.

Annexe

Le logiciel MASP

A.1 Description du logiciel

A.1.1 Structure du logiciel

Le logiciel MASP est composé essentiellement de trois parties, comme indiqué dans la figure A.1 ci-dessous.

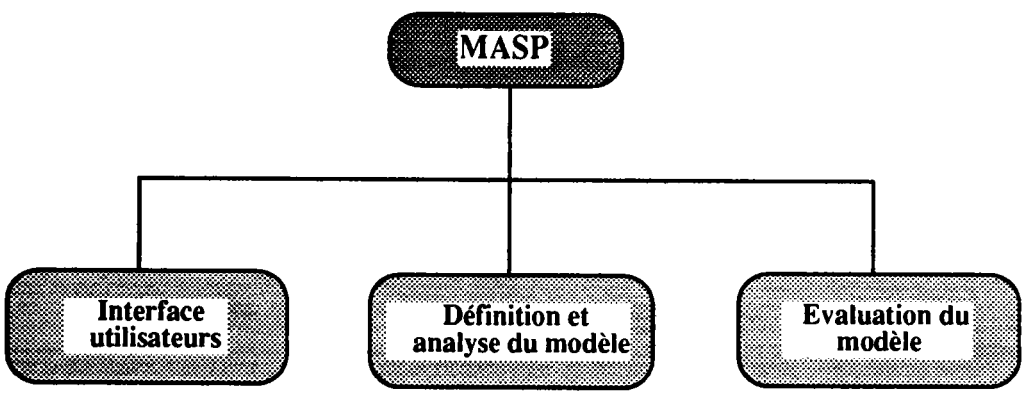


Figure A.1 : Organisation du logiciel MASP

La partie "interface utilisateurs" est responsable de l'interaction avec l'utilisateur, tant pour ce qui concerne la saisie des données nécessaires au fonctionnement du programme, que pour la présentation des résultats. Un système de fenêtrage est à la base de cet interface, ce qui rends l'utilisation du programme simple et conviviale. Ce système de fenêtrage à été développé en langage TCL et en utilisant l'ensemble de "widgets" TK [Ousterhout 93]. Il est important de souligner ici que l'intégration des procédures écrites en langage TCL dans le programme écrit en langage C se fait aisément, car toutes le commandes du langage TCL ont elles mêmes été écrites en langage C.

La partie "définition et analyse du modèle" contrôle toutes les opérations relatives à la modélisation. En plus des procédures de définition et de sauvegarde du modèle, on trouve dans cette partie les procédures utilisées pour la vérification de la décomposabilité du modèle et pour l'obtention des sous-réseaux utilisés dans la planification.

La partie "évaluation du modèle" contient les procédures utilisées pour la résolution des problèmes de planification et d'ordonnancement.

Dans le paragraphe A.2, nous présentons l'ensemble des procédures qui composent le logiciel ainsi que les liens existant entre elles.

A.1.2 Fonctionnement du logiciel

Le menu principal du logiciel MASP comporte les options suivantes :

- définir le modèle
- voir le modèle réseaux de Petri
- voir les résultats de la décomposition
- lancer la planification
- lancer l'ordonnancement
- quitter le programme

Ces options sont activées au fur et à mesure du déroulement du programme. Initialement seules les options "définir le modèle" et "quitter le programme" sont actives. Une fois qu'un modèle a été défini, les options "voir le modèle réseaux de Petri", "voir les résultats de la décomposition" et "lancer la planification" sont à leur tour activées. L'option "lancer l'ordonnancement" ne sera activée qu'après la résolution du problème de planification. Nous examinons chacune de ces options dans la suite.

Pour la définition du modèle l'utilisateur a le choix entre déclarer un nouveau modèle ou récupérer un modèle antérieurement déclaré et stocké dans un fichier. Le modèle peut être déclaré de deux façons :

(i) en décrivant les gammes de fabrication de chaque type de produit ;

(ii) en introduisant directement les modèles réseaux de Petri correspondants à chacune de ces gammes.

La déclaration d'une gamme de fabrication est réalisée à partir de la description des

opérations élémentaires de la gamme et de la définition des liens existants entre ces opérations. Par exemple, si la gamme de fabrication d'un produit P_i est donnée par :

$$P_i : \quad M_1(3) \rightarrow \begin{pmatrix} M_2(8) \\ M_5(1) \\ M_4(5) \end{pmatrix} \rightarrow M_3(2)$$

nous allons définir trois opérations élémentaires, respectivement O_1 ($M_1(3)$), O_2 ($M_2(8)$; $M_5(1)$; $M_4(5)$) et O_3 ($M_3(2)$), et ensuite définir les liens entre ces opérations en disant que l'opération O_1 précéde l'opération O_2 et que O_2 précéde O_3. Au même temps on définit les en-cours existants entre chaque opération et les opérations précédentes. Dans le cas où une opération O_i est précédée par plus d'une opération, l'utilisateur doit aussi définir s'il agit d'une opération d'assemblage ou non. A la fin de la déclaration le programme génère automatiquement les modèles réseaux de Petri associés à chacune des gammes.

L'introduction des modèles réseaux de Petri est faite de manière similaire au cas précédent. Pour cela, il faut définir pour chaque modèle :

- le nombre de places et de transitions ;
- le marquage initial de chaque place ;
- la temporisation de chaque transition, ainsi que la ressource correspondante ;
- les transitions d'entrée et de sortie de chaque place.

Lorsque le modèle a été défini, il est automatiquement sauvegardé dans un fichier. Ensuite le programme vérifie s'il agit d'un modèle décomposable, et si ceci n'est pas le cas, le programme envoie un message d'erreur. L'utilisateur peut dans ce cas introduire un nouveau modèle ou quitter le programme.

L'option "voir modèle réseaux de Petri" permet de visualiser la description du dernier modèle introduit par l'utilisateur. Le modèle est toujours présenté sous la forme des réseaux de Petri, même s'il a été défini à partir de gammes de fabrication. De même, le choix de l'option "voir résultats de la décomposition" permet de visualiser l'ensemble de sous-réseaux sans conflit extraits du modèle. Ces options utilisent une fenêtre texte pour la présentation des données.

Le choix de l'option "lancer la planification" entraîne l'ouverture d'un menu où l'utilisateur choisit le critère à optimiser ainsi que la forme sous laquelle les données de la planification sont introduites. Deux options sont possibles : prendre ces données dans un fichier ou les introduire à l'aide du clavier. Dans le premier cas, l'utilisateur est amené à choisir le

fichier contenant les données. Dans le deuxième cas, une zone de dialogue permet de les introduire progressivement : d'abord on spécifie le nombre et la durée des périodes, ensuite on introduit les demandes pour chaque type de produit à la fin de chaque période élémentaire, ainsi que les coûts de stockage et de rupture. Si cette option a déjà été utilisée auparavant, les données définies antérieurement sont affichées afin que l'utilisateur puisse les modifier ou non.

L'option "lancer l'ordonnancement" fait apparaître un menu où l'utilisateur peut choisir la période à ordonnancer ainsi que la méthode à utiliser. Le choix de la période à ordonnancer peut être quelconque. Une fois l'ordonnancement d'une période terminé, l'utilisateur a le choix entre relancer l'ordonnancement de la même période (en changent la méthode, par exemple), de lancer l'ordonnancement d'une autre période, ou de retourner au menu principal du programme.

Les données ainsi que les résultats obtenus dans les différentes étapes du programme sont sauvegardées dans un fichier de résultats. Ce fichier contient quatre parties : la description du modèle, les résultats de la décomposition, les résultats de la planification et les résultats de l'ordonnancement. Dans ce fichier ne sont stockés que les derniers résultats obtenus au cours des différentes étapes. Le relancement d'une option entraîne l'effacement des données relatives à cette option. Par exemple, la définition d'un nouveau modèle implique que toutes les données du fichier sont effacées ; si on relance la planification, seules les données relatives à la planification et à l'ordonnancement sont effacées.

A.2 Description des procédures utilisées

Les procédures utilisées dans le logiciel MASP ont été reparties dans six fichiers différents, à savoir : **masp.c**, **det_cfio.c**, **calc_inv.c**, **planif.c**, **ordon.c**, **util.c**. De plus, il existe un ensemble de fichiers correspondants à l'interface graphique, lesquels contiennent du code écrit en langage TCL. Nous donnons dans ce qui suit l'architecture du logiciel. La figure A.2 présente l'arbre des appels entre les différentes procédures utilisées qui illustre les liens qui existent entre les procédures. Nous donnons le rôle de chaque procédure ainsi que la description de chacun de ses paramètres. Pour cette présentation, nous organisons les procédures selon le fichier d'où elles font partie.

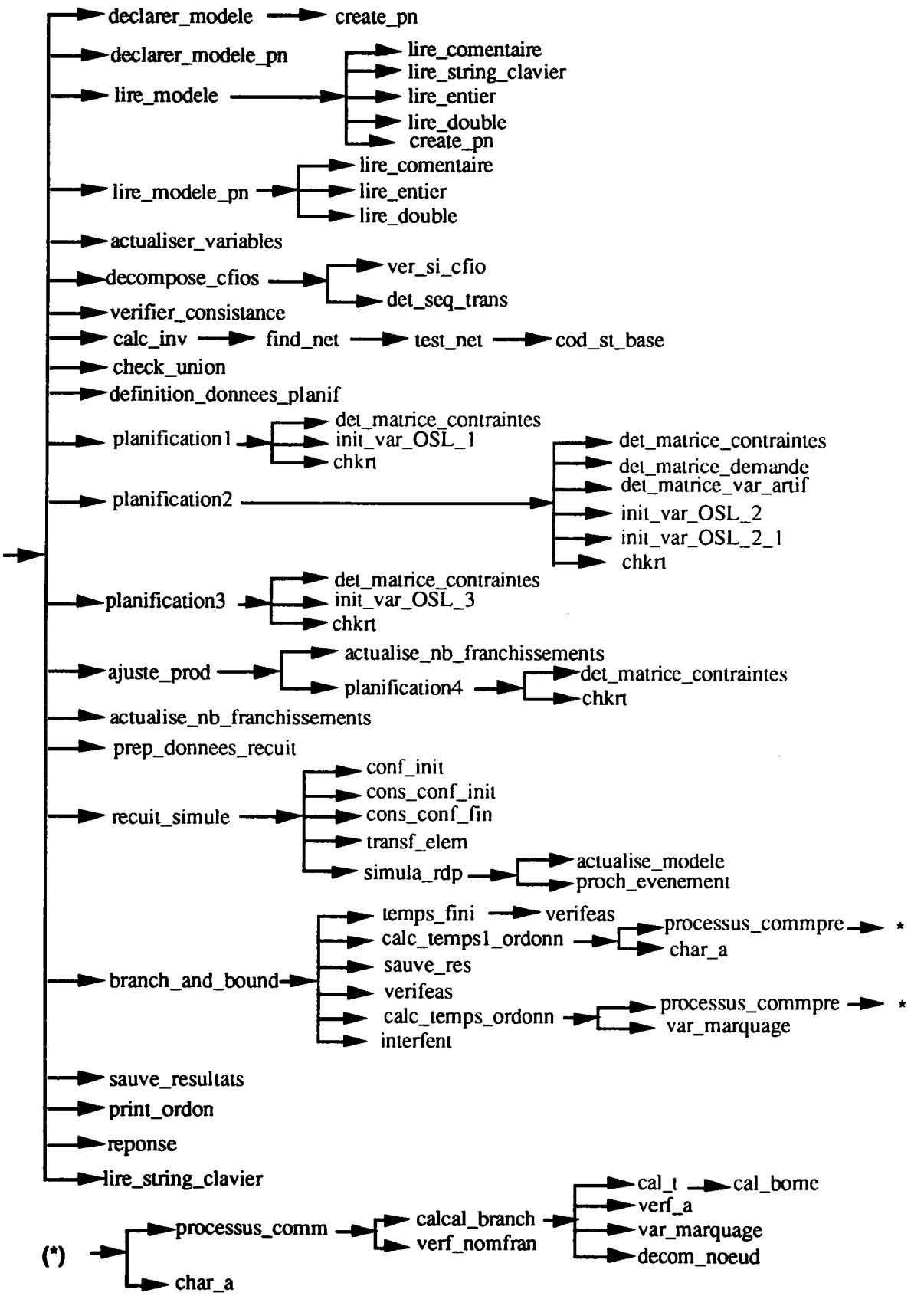


Figure A.2 : Arbre d'appel des procédures du logiciel

A.2.1 Procédures rangées dans le fichier masp.c

void lire_modele (fichier)
Objet : lire dans un fichier la description d'un modèle représenté à partir des gammes de fabrication.
Paramètres : - *fichier* : nom du fichier contenant le modèle.

void lire_modele_pn (fichier)
Objet : lire dans un fichier la description d'un modèle représenté à partir des réseaux de Petri.
Paramètres : - *fichier* : nom du fichier contenant le modèle.

void declarer_modele ()
Objet : déclarer un nouveau modèle à partir de la description des gammes de fabrication des produits, et sauvegarder le modèle dans un fichier (le nom du fichier est demandé à l'utilisateur)
Paramètres : - *aucun.*

void declarer_modele_pn ()
Objet : déclarer un nouveau modèle à partir de la description des modèles réseaux de Petri, et sauvegarder le modèle dans un fichier (le nom du fichier est demandé à l'utilisateur).
Paramètres : - *aucun.*

void definition_donnees_planif (critere, opt, nom_fich)
Objet : réaliser la définition des données pour le problème de planification.
Paramètres : - *critere* : critère choisi ;
- *opt* : indique si les données seront introduites (opt=0) ou lues dans un fichier (opt=1) ;
- *nom_fich* : nom du fichier contenant les données pour le cas où opt=0.

void prep_donnees_recuit (pt_sol, per, nb_invar)
Objet: préparer les données pour le recuit simulé.
Paramètres : - *pt_sol* : tableau contenant le résultat de la planification (i.e. les coefficients indiquant le nombre de fois que chaque sous-réseau est utilisé) ;
- *per* : période à ordonnancer ;
- *nb_invar* : nombre total de sous-réseaux du modèle.

void actualise_nb_franchissements (pt_sol, per, nb_invar)
Objet : déterminer le nombre de franchissements de chaque transition durant une période donnée.
Paramètres : - *pt_sol* : tableau contenant les résultats de la planification ;
- *per* : période considérée ;
- *nb_invar* : nombre total de sous-réseaux du modèle.

void ajuste_prod (sol, ninv, j)
Objet : vérifier si on peut produire quelques produits supplémentaires dans une période donnée. Cette procédure est utilisée dans l'heuristique de résolution du problème de planification où le critère correspond à la somme des coûts de stockage et de rupture.
Paramètres : - *sol* : tableau contenant les résultats de la planification ;
- *ninv* : nombre total de sous-réseaux du modèle ;
- *j* : période considérée.

void sauve_resultats (fich)
Objet : sauvegarder la description du modèle et les résultats de la décomposition de celui-ci dans le fichier des résultats.
Paramètres : - *fich* : nom du fichier où sauvegarder les résultats;

int actualiser_variables ()
Objet : actualiser variables internes du programme.
Paramètres : - *aucun.*

int verifier_consistance (cfio, fichier)
Objet : vérifier si un sous-réseau sans conflits extrait du modèle est consistent.
Paramètres : - *cfio* : structure contenant la description du sous-réseau ;
- *fichier* : nom du fichier où sont stockés les t-invariants du sous-réseau.

int check_union (pn1)
Objet : vérifier si l'union des sous-réseaux sans conflit extraits d'un modèle couvre le réseau.
Paramètres : - *pn1* : structure contenant la description du modèle (RPES) correspondant à un produit.

void create_pn (tb_gamme, npt, ntt, k, nop)
Objet : créer le modèle réseaux de Petri associé à un produit à partir de la description de sa gamme de fabrication.
Paramètres : - *tb_gamme* : structure contenant la description de la gamme de fabrication du produit ;
- *npt* : nombre total de places déjà utilisées dans le modèle global ;
- *ntt* : nombre total de transitions déjà utilisées dans le modèle global
- *k* : indice du produit ;
- *nop* : nombre total d'opérations de la gamme.

A.2.2 Procédures rangées dans le fichier det_cfio.c

int decompose_cfios (pn1)
Objet : extraire tous les sous-réseaux maximaux sans conflits du modèle RPES décrivant la gamme de fabrication d'un produit.
Paramètres : - *pn1* : structure contenant la description du modèle RPES du produit.

int det_seq_trans (pn2, ligne, pos)
Objet : déterminer tous les ensembles maximaux composés de transitions qui ne sont pas en conflit. Cette fonction est utilisée avant la décomposition du modèle et elle est récursive.
Paramètres : - *pn2* : structure contenant la description du modèle RPES d'un produit ;
- *ligne* : indique la ligne de la matrice d'incidence considérée dans chaque appel de la fonction ;
-*pos* : indique la position actuelle de lecture du tableau contenant les places qui possèdent plus d'une transition de sortie dans le réseau.

int ver_si_cfio (matr1, n_ligne, n_col)
Objet : vérifier si le sous-réseau obtenu est un sous-réseau sans conflit.
Paramètres : - *matr1* : matrice d'incidence du sous-réseau ;
- *n_ligne* : nombre de lignes de la matrice d'incidence du sous-réseau ;
- *n_col* : nombre de colonnes de la matrice d'incidence du sous-réseau.

A.2.3 Procédures rangées dans le fichier calc_inv.c

void calc_inv (matr, np, nt, entree, sortie, n_sort, fichier)

Objet : décomposer un sous-réseaux maximal sans conflit en des sous-réseaux dont les transitions correspondent au support d'un t-invariant à support minimal.
Paramètres : - *matr* : matrice d'incidence du sous-réseau maximal ;
- *np* : nombre de lignes de la matrice d'incidence du sous-réseau maximal ;
- *nt* : nombre de colonnes de la matrice d'incidence du sous-réseau maximal ;
- *entree* : tableau identifiant les transitions d'entrée du sous-réseau maximal ;
- *sortie* : tableau identifiant les transitions de sortie du sous-réseau maximal ;
- *n_sort* : nombre de transitions de sortie du du sous-réseau maximal ;
- *fichier* : nom du fichier où sont stockés les t-invariants à support minimal associés à chacun de sous-réseaux obtenus après la décomposition.

void find_net (matr1, np1, nt1, pos, fich1)
Objet : extraire un sous-réseau correspondant au support d'un t-invariant à support minimal.
Paramètres : - *matr1* : matrice d'incidence du sous-réseau maximal ;
- *np1* : nombre de lignes de la matrice d'incidence du sous-réseau maximal ;
- *nt1* : nombre de colonnes de la matrice d'incidence du sous-réseau maximal ;
- *pos* : colonne correspondant à la transition de sortie à partir de laquelle on commence la recherche ;
- *fich1* : nom du fichier où stocker le t-invariant associé au sous-réseau obtenu.

void test_net (matr1, np1, nt1, net, fich2)
Objet : vérifier si les transitions du sous-réseau obtenu sont le support d'un t-invariant à support minimal.
Paramètres : - *matr1* : matrice d'incidence du sous-réseau maximal ;
- *np1* : nombre de lignes de la matrice d'incidence du sous-réseau maximal ;
- *nt1* : nombre de colonnes de la matrice d'incidence du sous-réseau maximal ;
- *net* : structure pour sauvegarder le sous-réseau ;
- *fich1* : nom du fichier où stocker les t-invariants associés au sous-réseau obtenu.

void cod_st_base (nt1, trs, cod)
Objet : associer un nombre entier (un code) à chaque sous-réseau. Ceci sert a faciliter la comparaison des sous-réseaux..
Paramètres : - *nt1* : nombre de transitions du sous-réseau ;
- *trs* : tableau contenant l'identification des transitions appartenant au sous-réseau ;
- *cod* : tableau contenant le code associé aux sous-réseaux déjà obtenus.

A.2.4 Procédures rangées dans le fichier planif.c

int planification1 (sol, nb_invar)
Objet : résoudre le problème de planification dans le cas où le critère correspond à la somme des coûts de stockage et de rupture : solution exacte.
Paramètres : - *sol* : pointeur du tableau pour la sauvegarde des résultats
- *nb_invar* : nombre total de sous-réseaux (t-invariants) utilisés.

int planification2 (sol, nb_invar)
Objet : résoudre le problème de planification dans le cas où le critère correspond à l'équilibrage des charges des machines.
Paramètres : - *sol* : pointeur du tableau pour la sauvegarde des résultats ;
- *nb_invar* : nombre total de sous-réseaux (t-invariants) utilisés ;

int planification3 (sol, nb_invar, ind_per)
Objet : résoudre le problème de planification dans le cas où le critère correspond à la somme des coûts de stockage et de rupture : solution heuristique (réelle).
Paramètres : - *sol* : pointeur du tableau pour la sauvegarde des résultats ;
- *nb_invar* : nombre total de sous-réseaux (t-invariants) utilisés ;
- *ind_per* : première période considérée dans la planification.

int planification4 (sol, nb_invar, mat_chg, tb_prod, tb_dem)
Objet : résoudre le problème de placement de produits dans une période. Cette fonction est utilisée conjointement avec la procédure précédente.
Paramètres : - *sol* : pointeur du tableau pour la sauvegarde des résultats ;
- *nb_invar* : nombre total de sous-réseaux (t-invariants) utilisés ;
- mat_chg : charge libre des machines dans la période ;
- tb_prod : tableau contenant la production durant la période ;
- tb_dem : tableau contenant les demandes durant la période.

void det_matrice_constraintes (pt_matr, nb_invar)
Objet : détermination de la matrice représentant les contraintes associées aux durées maximales d'utilisation des machines durant une période.
Paramètres : - *pt_matr* : structure pour la sauvegarde de la matrice obtenue ;
- *nb_invar* : nombre total de sous-réseaux (t-invariants) utilisés ;

void det_matrice_demande (pt_matr, nb_invar)
Objet : détermination de la matrice des contraintes relatives à la demande des produits.
Paramètres : - *pt_matr* : structure pour la sauvegarde de la matrice obtenue ;
- *nb_invar* : nombre total de sous-réseaux (t-invariants) utilisés ;

void det_matrice_var_artif (pt_matr, nb_invar)
Objet : détermination de la matrice associée à des variables artificielles.
Paramètres : - *pt_matr* : structure pour la sauvegarde de la matrice obtenue ;
- *nb_invar* : nombre total de sous-réseaux (t-invariants) utilisés ;

int init_variables_osl_1 (pt_matr, pt_dels, pt_mrow, pt_mcol, pt_drlo, pt_drup, pt_dclo, pt_dcup, pt_dobj, nlig1, ncol1)
Objet : préparation des données pour la la procédure planification1.
Paramètres : - *pt_matr* : matrice des contraintes ;
- *pt_dels*: tableau des éléments non nuls de la matrice globale utilisée dans la résolution du problème de programmation linéaire en nombres entiers;
- *pt_mrow* : tableau indiquant la ligne des éléments non nuls de la matrice globale ;
- *pt_mcol* : tableau indiquant la colonne des éléments non nuls de la matrice globale ;
- *pt_drlo* : tableau des bornes inférieures de la matrice globale ;
- *pt_drup* : tableau des bornes supérieures de la matrice globale ;
- *pt_dclo* : tableau des bornes inférieures des variables ;
- *pt_dcup* : tableau des bornes supérieures des variables ;
- *pt_dobj* : tableau des coefficients de la fonction objectif ;
- *nlig1* : nombre de lignes de la matrice globale ;
- *ncol1* : nombre de colonnes de la matrice globale ;

int init_variables_osl_2 (pt_matr, pt_matr1, pt_matr2, pt_dels, pt_mrow, pt_mcol, pt_drlo, pt_drup, pt_dclo, pt_dcup, pt_dobj, pt_delsq, pt_mrowq, pt_mcolq, nlig1, ncol1)
Objet : préparation des données pour la première partie de l'algorithme de résolution du problème de programmation quadratique.
paramètres : - *pt_matr* : matrice des contraintes de capacité ;
- *pt_matr1* : matrice des contraintes de demande ;

- *pt_matr2* : matrice correspondante à des variables artificielles ;
- *pt_dels* : tableau des éléments non nuls de la matrice globale utilisée dans la résolution du problème de programmation quadratique - partie linéaire ;
- *pt_mrow* : tableau indiquant la ligne des éléments non nuls de la matrice globale ;
- *pt_mcol* : tableau indiquant la colonne des éléments non nuls de la matrice globale ;
- *pt_drlo* : tableau des bornes inférieures de la matrice globale ;
- *pt_drup* : tableau des bornes supérieures de la matrice globale ;
- *pt_dclo* : tableau des bornes inférieures des variables;
- *pt_dcup* : tableau des bornes supérieures des variables ;
- *pt_dobj* : tableau des coefficients de la fonction objectif ;
- *pt_delsq* : tableau des éléments non nulles de la matrice quadratique ;
- *pt_mrowq* : tableau indiquant la ligne des éléments non nuls de la matrice représentant la partie quadratique du problème ;
- *pt_mcolq* : tableau indiquant la colonne des éléments non nuls de la matrice quadratique ;
- *nlig1* : nombre de lignes de la matrice globale ;
- *ncol1* : nombre de colonnes de la matrice globale ;
- *ind_per*: indice de la première période considérée.

int init_variables_osl_2_1 (pt_matr, pt_matr1, pt_dels, pt_mrow, pt_mcol, pt_drlo, pt_drup, pt_dclo, pt_dcup, pt_dobj, nlig1, ncol1, sol_quad)

Objet : préparation des données pour la deuxième partie de l'algorithme de résolution du problème de programmation quadratique.

paramètres :
- *pt_matr* : matrice des contraintes de capacité ;
- *pt_matr1* : matrice des contraintes de demande ;
- *pt_dels* : tableau des éléments non nuls de la matrice globale utilisée dans la résolution du problème de programmation linéaire en nombres entiers ;
- *pt_mrow* : tableau indiquant la ligne des éléments non nuls de la matrice globale ;
- *pt_mcol* : tableau indiquant la colonne des éléments non nuls de la matrice globale ;
- *pt_drlo* : tableau des bornes inférieures de la matrice globale ;
- *pt_drup* : tableau des bornes supérieures de la matrice globale ;
- *pt_dclo* : tableau des bornes inférieures des variables;
- *pt_dcup* : tableau des bornes supérieures des variables ;
- *pt_dobj* : tableau des coefficients de la fonction objectif ;
- *nlig1* : nombre de lignes de la matrice globale ;
- *ncol1* : nombre de colonnes de la matrice globale ;
- *sol_quad* : pointeur du tableau contenant les résultats de la résolution du problème quadratique.

int init_variables_osl_3 (pt_matr, pt_dels, pt_mrow, pt_mcol, pt_drlo, pt_drup, pt_dclo, pt_dcup, pt_dobj, nlig1, ncol1, ind_per)

Objet : préparation des données pour la procédure planification3.

Paramètres :
- *pt_matr* : matrice de contraintes ;
- *pt_dels* : tableau des éléments non nuls de la matrice globale utilisée dans la résolution du problème de programmation linéaire en nombres réels ;
- *pt_mrow* : tableau indiquant la ligne des éléments non nuls de la matrice globale ;
- *pt_mcol* : tableau indiquant la colonne des éléments non nuls de la matrice globale ;
- *pt_drlo*: tableau des bornes inférieures de la matrice globale ;
- *pt_drup*: tableau des bornes supérieures de la matrice globale;
- *pt_dclo*: tableau des bornes inférieures des variables;
- *pt_dcup*: tableau des bornes supérieures des variables ;

- *pt_dobj*: tableau des coefficients de la fonction objectif ;
- *nlig1* : nombre de lignes de la matrice globale ;
- *ncol1* : nombre de colonnes de la matrice globale ;
- *ind_per*: indice de la première période considérée.

void chkrt(rtname,rtcod)
Objet : procédure d'impression du code d'erreur pour les fonctions OSL.
Paramètres : - *rtname* : nom de la fonction utilisée ;
- *rtcod* : code d'erreur.

A.2.5 Procédures rangées dans le fichier ordon.c

double recuit_simule (opt1, opt2, deb, coufi, charge)
Objet : réaliser l'ordonnancement en utilisant la méthode basée sur le recuit simulé.
Paramètres : - *opt1* : indique si les paramètres de contrôle ont été changés par rapport aux utilisations précédentes ;
- *opt2* : indique si on doit changer ou non l'ordonnancement initial ;
- *deb* : indique le premier appel de cette procédure ;
- *coufi* : indique la valeur du meilleur ordonnancement déjà obtenu ;
- *charge* : indique la charge maximale des machines.

void conf_init ()
Objet : obtenir une solution initiale pour la méthode d'ordonnancement basée sur le recuit simulé.
Paramètres : - *aucun* ;

void cons_conf_int ()
Objet : sauvegarder une solution intermédiaire acceptée (recuit simulé).
Paramètres : - *aucun* ;

void cons_conf_int ()
Objet : sauvegarder une solution qui améliore l'ordonnancement (recuit simulé).
Paramètres : - *aucun* ;

int trans_elem ()
Objet : réaliser une transformation élémentaire (recuit simulé).
Paramètres : - *aucun* ;

double simula_rdp ()
Objet : simuler le modèle (recuit simulé).
Paramètres : - *aucun* ;

void actualise_modele (dt)
Objet : actualiser l'état du modèle à chaque cycle de simulation .
Paramètres : - *dt* : pas de simulation ;

double proch_evennement (temps_sim)
Objet : déterminer le prochain changement d'états.
Paramètres : - *temps_sim* : temps total de simulation ;

double branch_and_bound (interp, per, fich)
Objet : réaliser l'ordonnancement en utilisant la méthode basée sur la procédure par séparation et évaluation.
Paramètres : - *interp* : interprète du langage TCL ;

- *per* : période à ordonnancer ;
- *fich* : nom du fichier de sauvegarde des résultats du programme.

void temps_fini ()
Objet : sauvegarder un nouveau ordonnancement (procédure par séparation et évaluation).
Paramètres : - *aucun* ;

void calcul_temps1_ordonn ()
Objet : calculer le premier ordonnancement (procédure par séparation et évaluation).
Paramètres : - *aucun* ;

void calcul_temps_ordonn ()
Objet : calculer les ordonnancement suivantes (procédure par séparation et évaluation).
Paramètres : - *aucun* ;

void sauve_res ()
Objet : préparer la présentation du meilleur ordonnancement trouvé (procédure par séparation et évaluation).
Paramètres : - *aucun* ;

void verifeas ()
Objet : vérifier les résultats (procédure par séparation et évaluation).
Paramètres : - *aucun* ;

void interfent ()
Objet : préparer les données pour l'exécution de la procédure d'ordonnancement basée sur la procédure de séparation et évaluation.
Paramètres : - *aucun* ;

void processus_commpre ()
Objet : vérifier si le calcul d'une solution est terminé (procédure par séparation et évaluation).
Paramètres : - *aucun* ;

void processus_comm ()
Objet : contrôler le processus d'obtention des solutions dans le cas de la la procédure d'ordonnancement basée sur la procédure de séparation et évaluation.
Paramètres : - *aucun* ;

void calcal_branch ()
Objet : calculer des branches (procédure par séparation et évaluation).
Paramètres : - *aucun* ;

void decom_noeud ()
Objet : décomposer un nœud (procédure par séparation et évaluation).
Paramètres : - *aucun* ;

void ver_a ()
Objet : définir toutes les séquences possibles à partir d'un nœud (procédure par séparation et évaluation).
Paramètres : - *aucun* ;

void cal_t ()
Objet : calculer les états (procédure par séparation et évaluation).
Paramètres : - *aucun* ;

void cal_borne ()

Objet : calculer les bornes (procédure par séparation et évaluation).
Paramètres : - *aucun* ;

void verf_nomfran ()
Objet : vérifier le nombre de franchissements (procédure par séparation et évaluation).
Paramètres : - *aucun* ;

void var_marquage ()
Objet : variation des marquages (procédure par séparation et évaluation).
Paramètres : - *aucun* ;

void char_a ()
Objet : créer la matrice contenant les transitions franchissables pour chaque machine (procédure par séparation et évaluation).
Paramètres : - *aucun* ;

void print_ordon (interp, fichier, nom, duree1, duree2, per)
Objet : présenter les résultats de l'ordonnancement.
Paramètres : - *interp* : interprète du langage TCL ;
- *fichier* : si cette variable est égale à 1 les résultats sont dirigés vers un fichier, sinon ils sont présentés à l'écran ;
- *nom* : nom du fichier où stocker les résultats ;
- *duree1* : durée de la période utilisée dans la planification ;
- *duree2* : durée réelle de la période ;
- *per* : période considérée.

A.2.6 Procédures rangées dans le fichier util.c

int reponse (fp_in, nelm, opt, res)
Objet : sélectionner une option.
Paramètres : - *fp_in* : pointeur du fichier correspondant au clavier ;
- *nelm* : nombre d'options ;
- *opt* : tableau contenant les options possibles ;
- *res* : variable pour la sauvegarde de l'option choisie.

void lire_comentaire (fp_in)
Objet : lire une ligne contenant des commentaires dans un fichier.
Paramètres : - *fp_in* : pointeur du fichier.

int lire_string_clavier (s_out)
Objet : lire une chaîne de caractères introduite par le clavier.
Paramètres : - *s_out* : pointeur pour la sauvegarde de la chaîne lue.

void lire_entier (fp_in)
Objet : lire une ligne contenant un nombre entier dans un fichier .
Paramètres : - *fp_in* : pointeur du fichier.

void lire_double (fp_in)
Objet : lire une ligne contenant un nombre réel en double précision dans un fichier .
Paramètres : - *fp_in* : pointeur du fichier.

A.2.7 Fichiers destinés à l'interfaçage graphique

Il existe au total quinze fichiers contenant le code pour l'interfaçage graphique. Ces fichiers sont utilisés chaque fois qu'on a besoin d'introduire ou de présenter des données et/ou des résultats. Chaque fichier contient le code pour réaliser une certaine partie de l'interface comme décrit ci-dessous.

- **decl_mod.tcl** : introduction d'un modèle à partir des gammes de fabrication ;
- **decl_pn.tcl** : introduction d'un modèle à partir des réseaux de Petri ;
- **dem_fichier.tcl** : demander à l'utilisateur le nom d'un fichier ;
- **demande.tcl** : introduction des données pour la planification ;
- **file_sel.tcl** : sélection d'un fichier ;
- **menu.tcl** : présentation du menu principal du programme ;
- **menu1_recuit.tcl** : menu pour le choix des paramètres du recuit simulé ;
- **menu2_recuit.tcl** : menu pour le choix des options de la méthode basée sur le recuit simulé;
- **menu1_ordo.tcl** : menu pour la définition de la période et choix de la méthode d'ordonnancement ;
- **menu_mod.tcl** : menu pour le choix de la forme d'introduction du modèle ;
- **menu1_planif.tcl** : menu pour choix de la méthode de planification et pour la définition de la manière dont les données de planification sont introduits ;
- **mess_erreur.tcl** : présenter un message d'erreur ;
- **message.tcl** : présenter un message autre qu'un message d'erreur ;
- **oui_non.tcl** : demander une confirmation du type oui/non à l'utilisateur ;
- **show_text.tcl** : présenter résultats sous la forme de texte.

A.3 Présentation de l'interfaçage graphique du logiciel MASP

Nous présentons par la suite des copies d'écran illustrant l'interfaçage graphique dans les différentes phases d'utilisation du logiciel MASP. L'ordre des figures est la suivante :

- Figures A.1 - A.5 concernent la déclaration du modèle ;

- Figures A.6 - A.8 concernent la planification ;

- Figures A.8 - A.12 concernent l'ordonnancement.

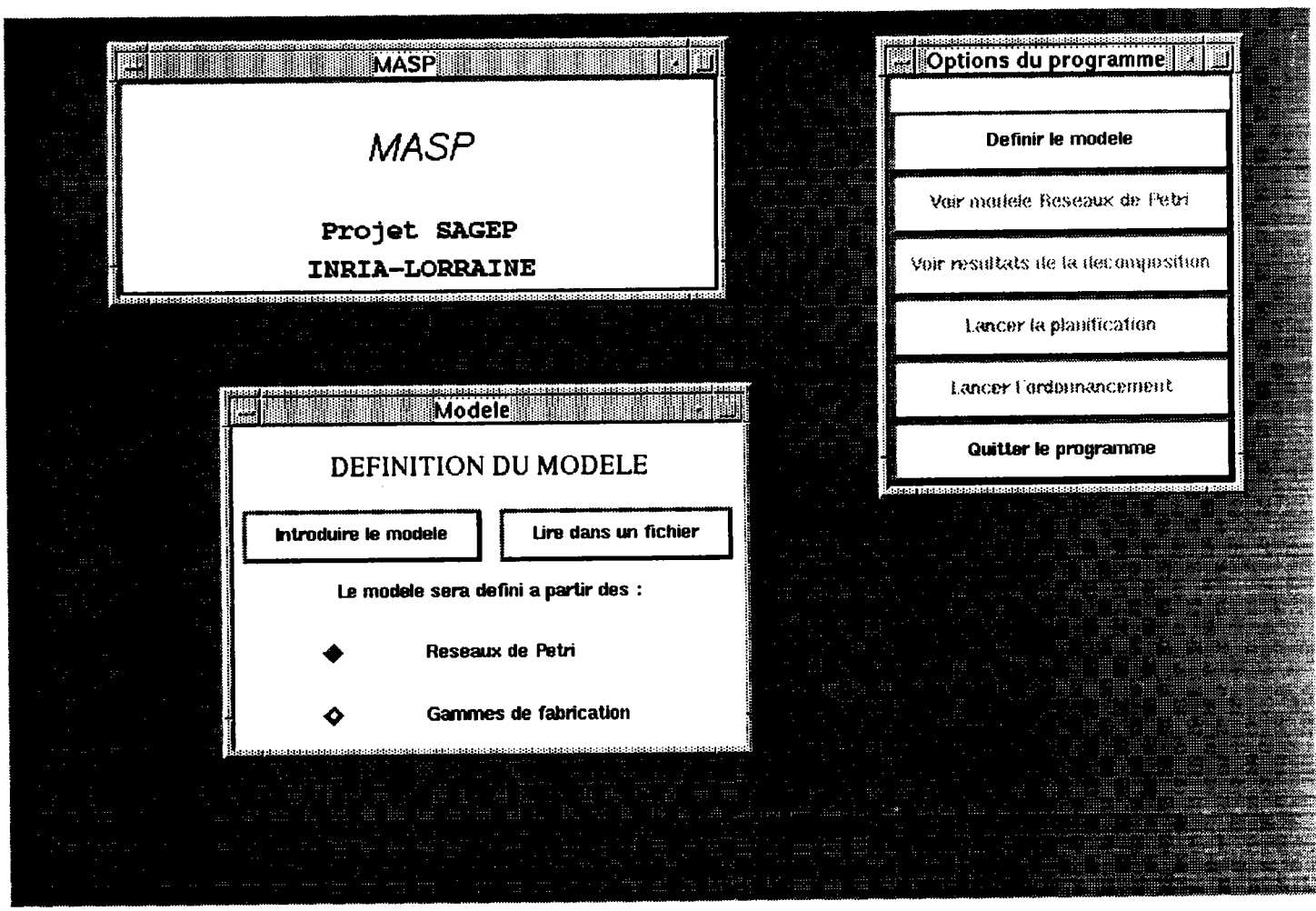


Figure A.2 : Choix de la forme d'introduction du modèle

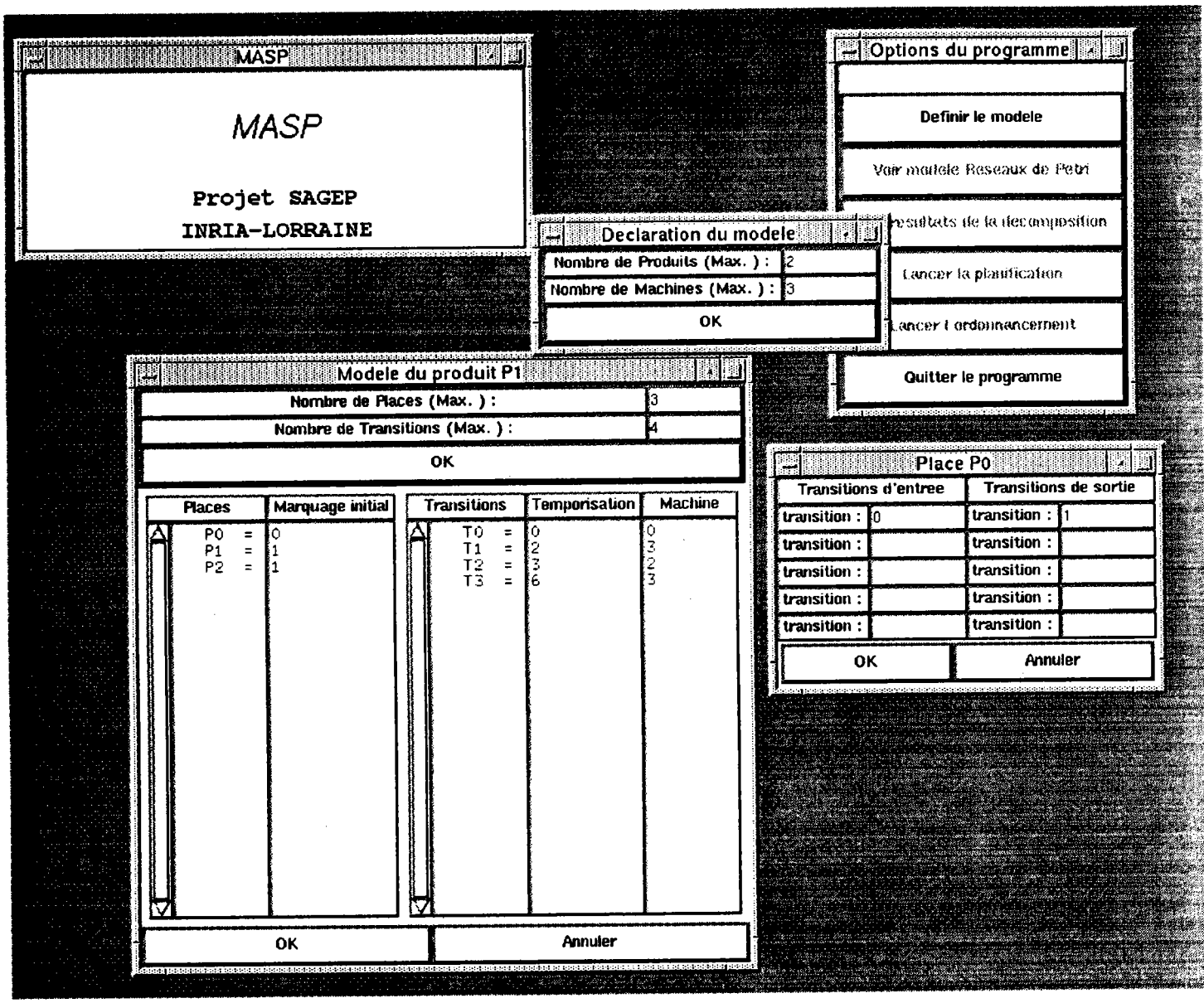


Figure A.3 : Définition d'un modèle à partir des réseaux de Petri

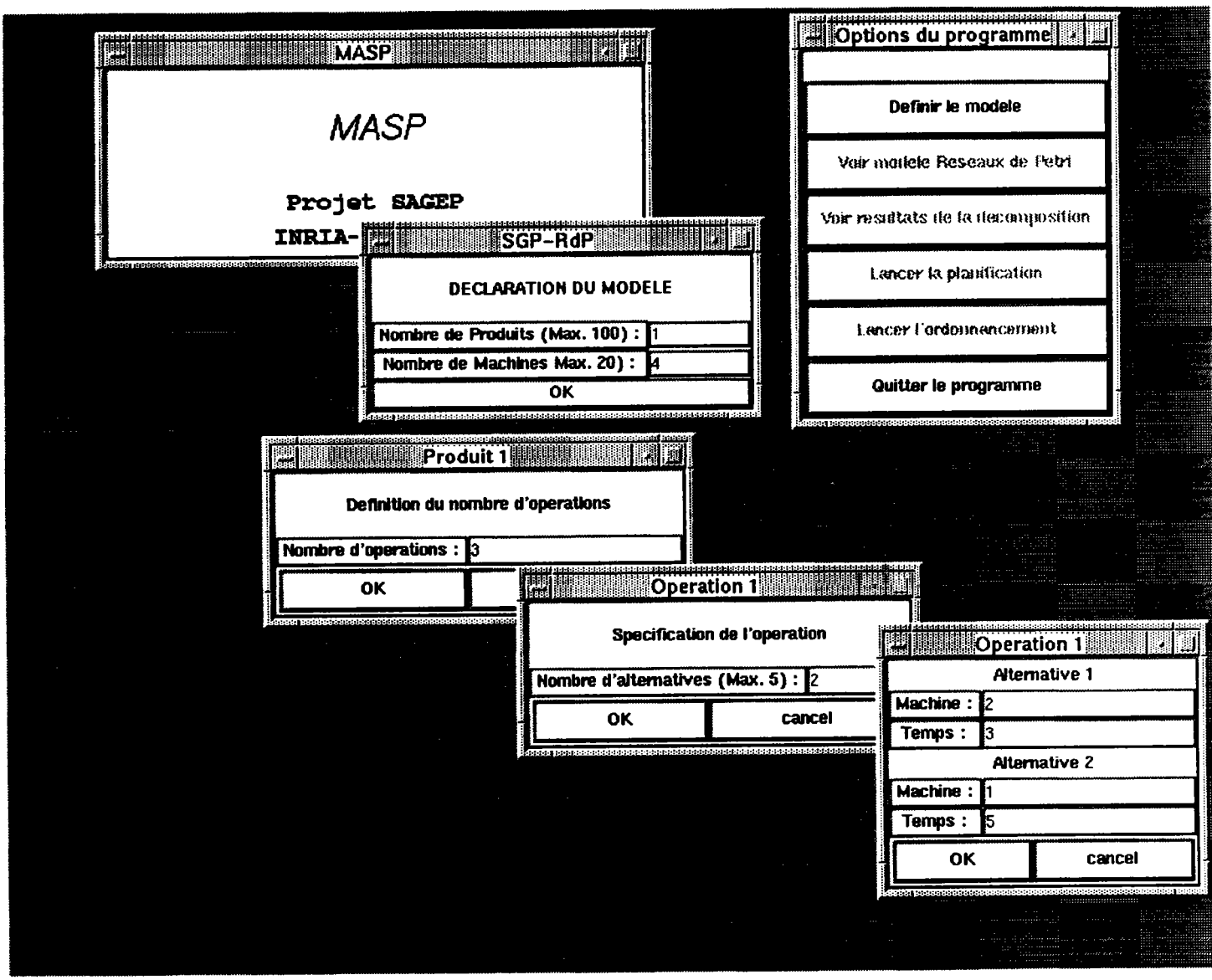


Figure A.4 : Définition d'un modèle à partir des gammes de fabrication

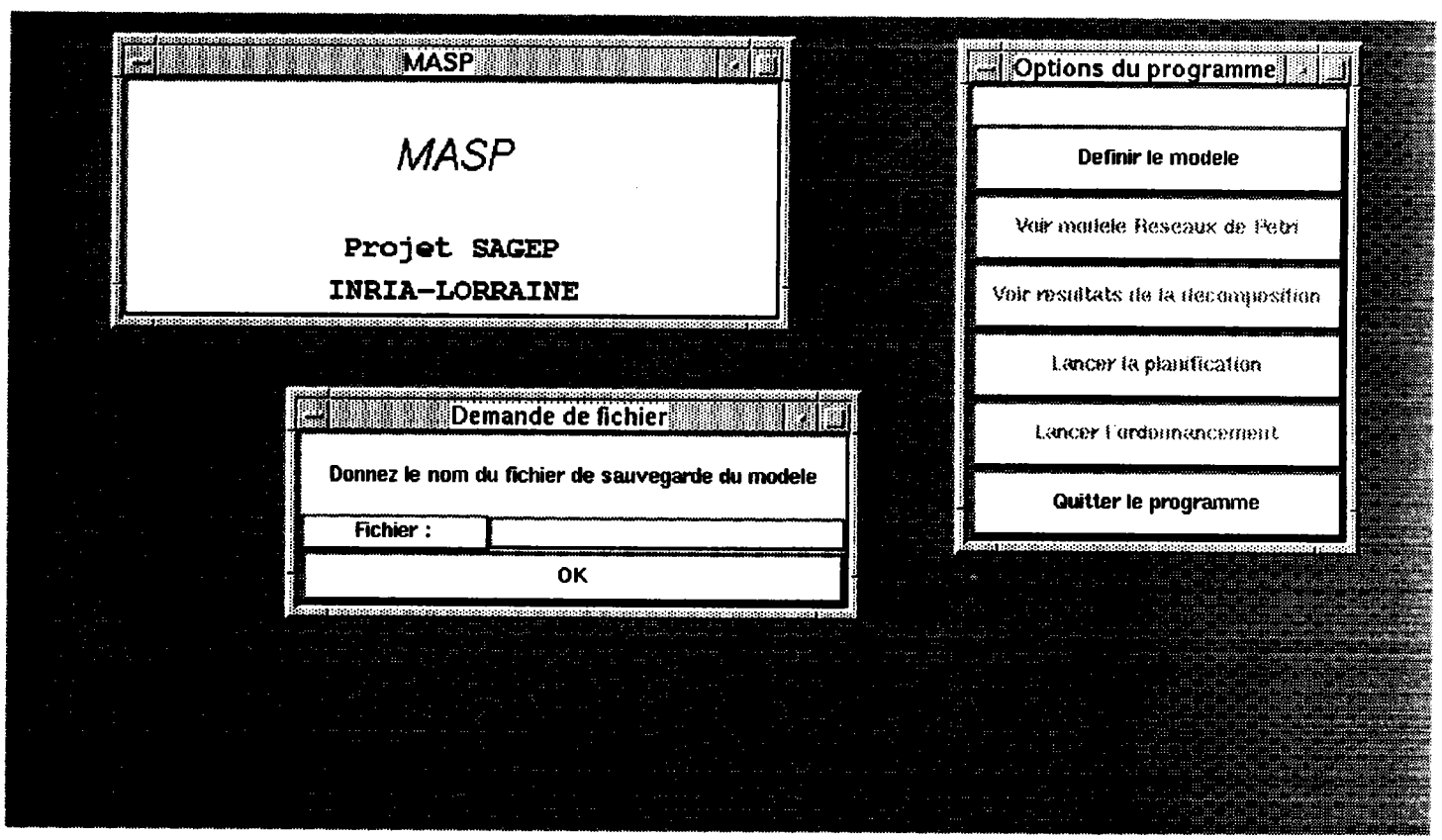


Figure A.5 : Définition du nom de fichier pour la sauvegarde du modèle

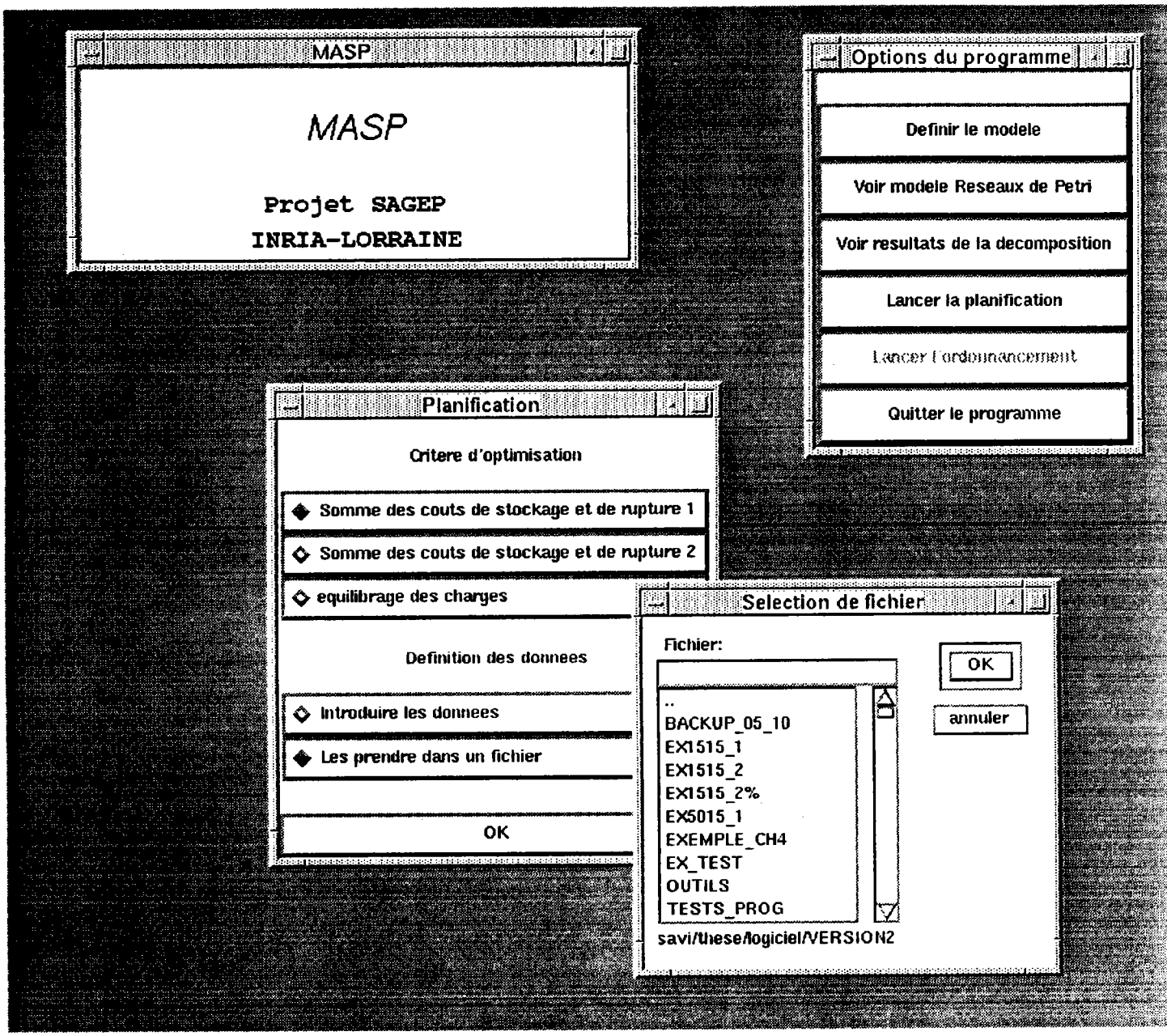


Figure A.6 : Choix du critère pour la planification et sélection du fichier contenant les données

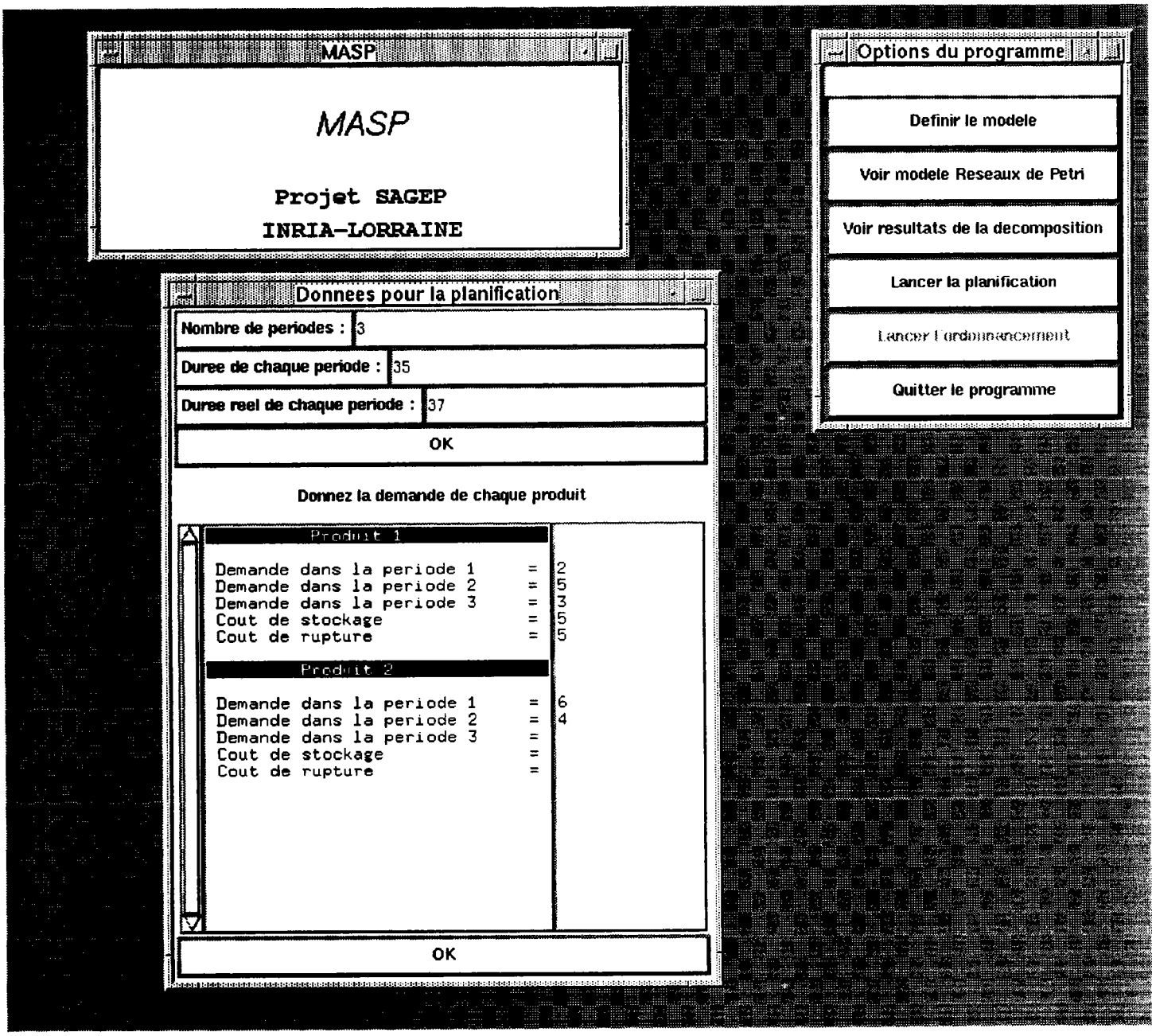


Figure A.7 : Introduction des données pour la planification

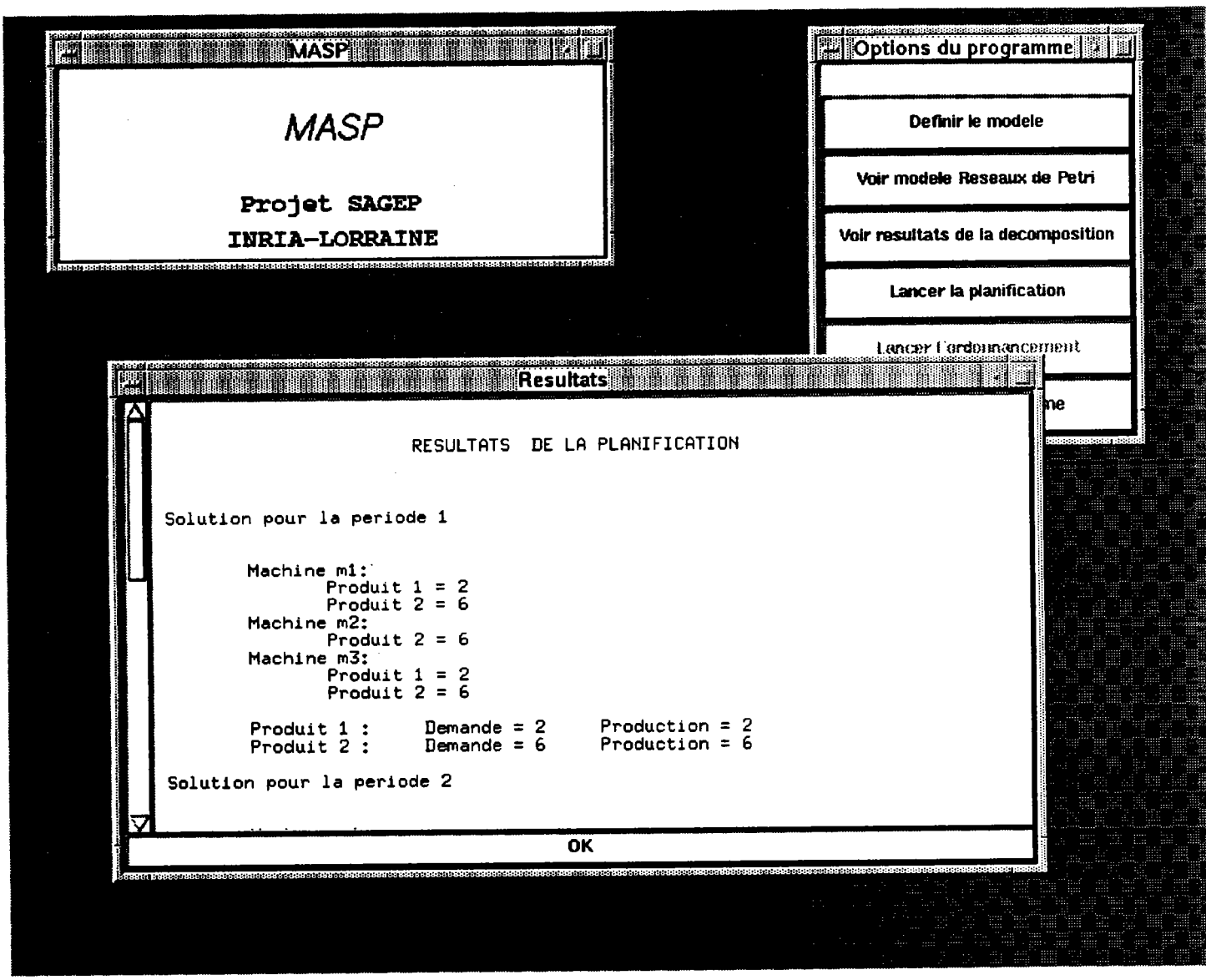


Figure A.8 : Présentation des résultats de la planification

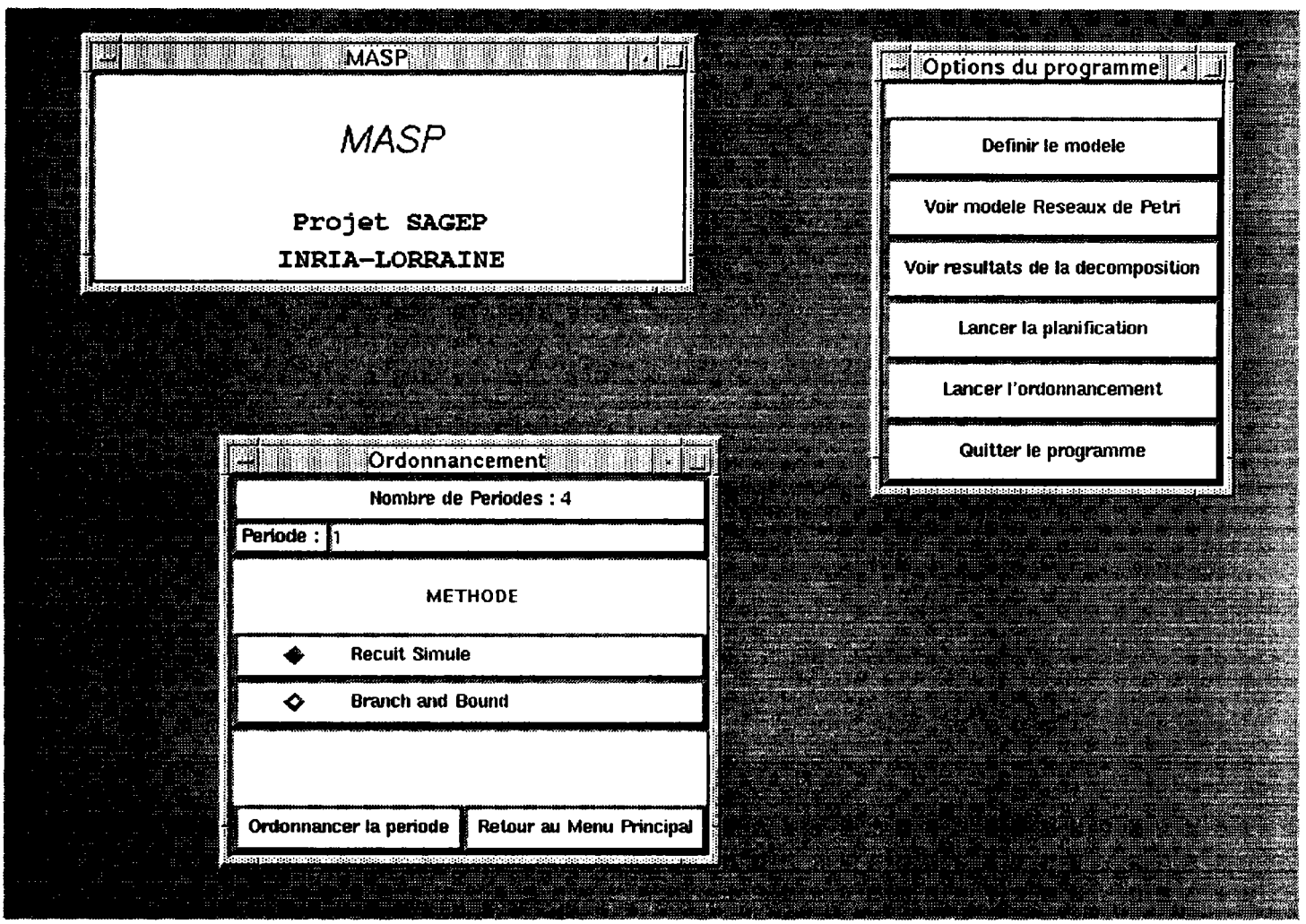


Figure A.9 : Sélection de la période à ordonnancer et de la méthode à utiliser dans l'ordonnancement

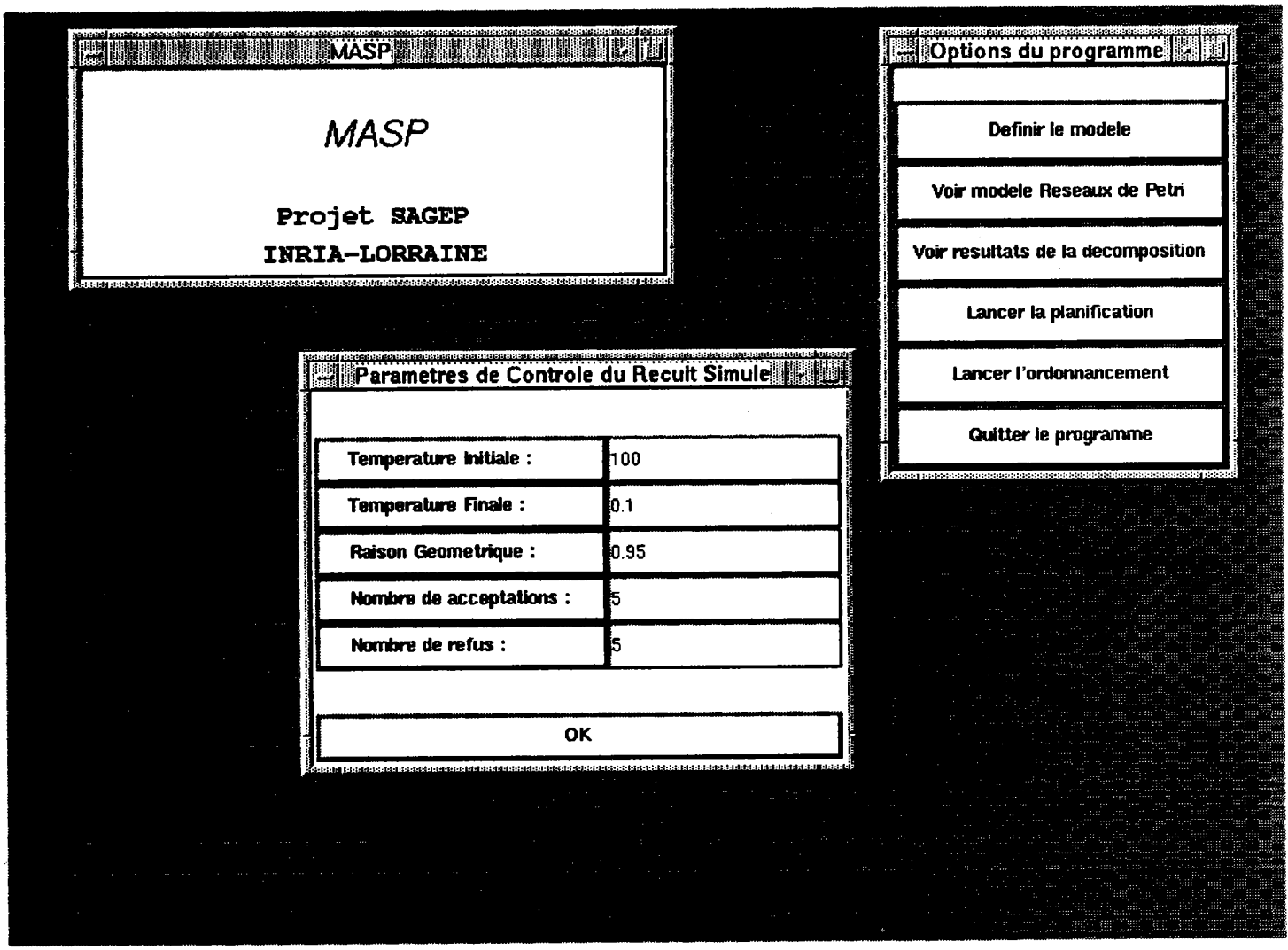


Figure A.10 : Introduction des données pour la méthode d'ordonnancement basée sur le recuit simulé

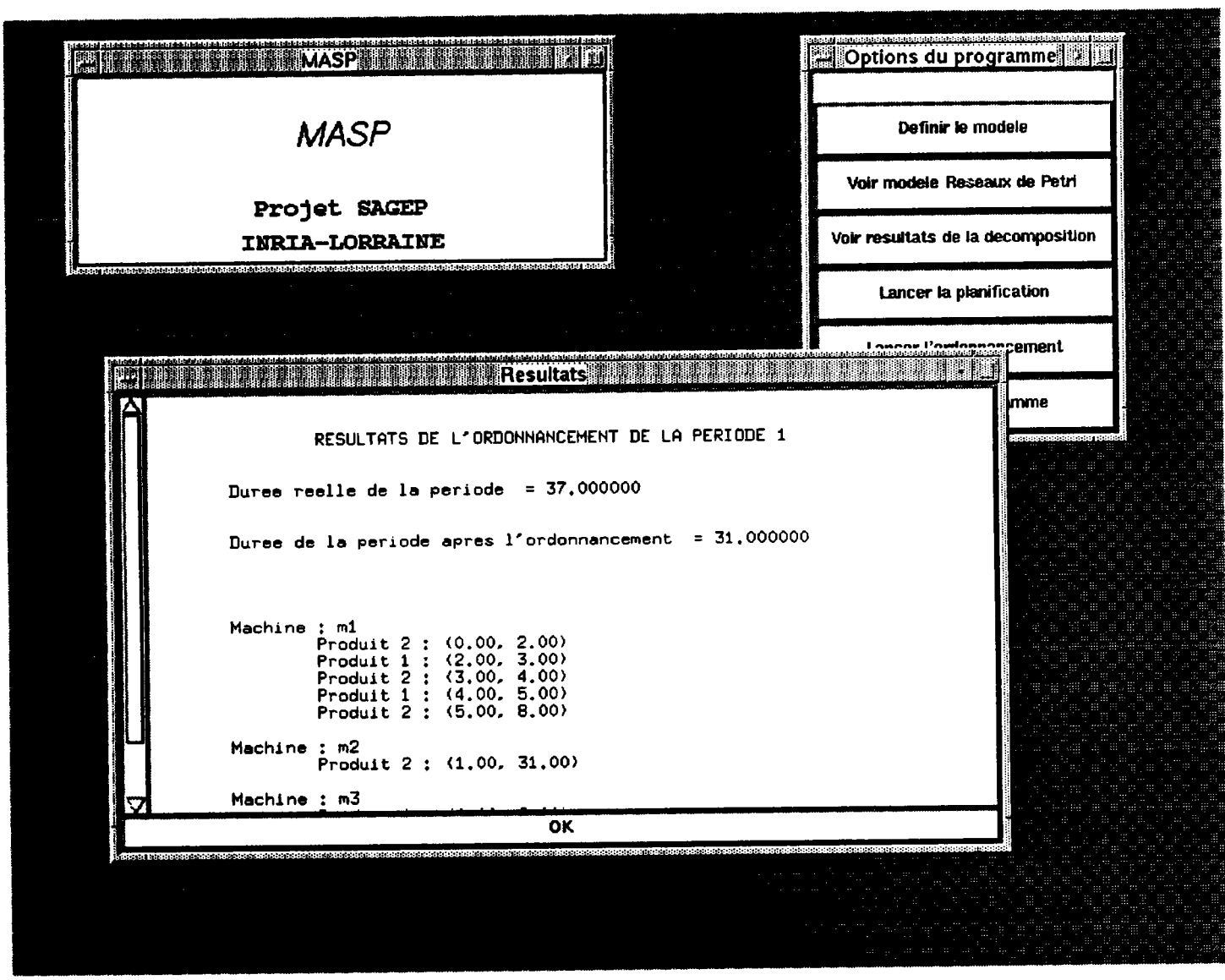


Figure A.11 : Présentation des résultats de l'ordonnancement

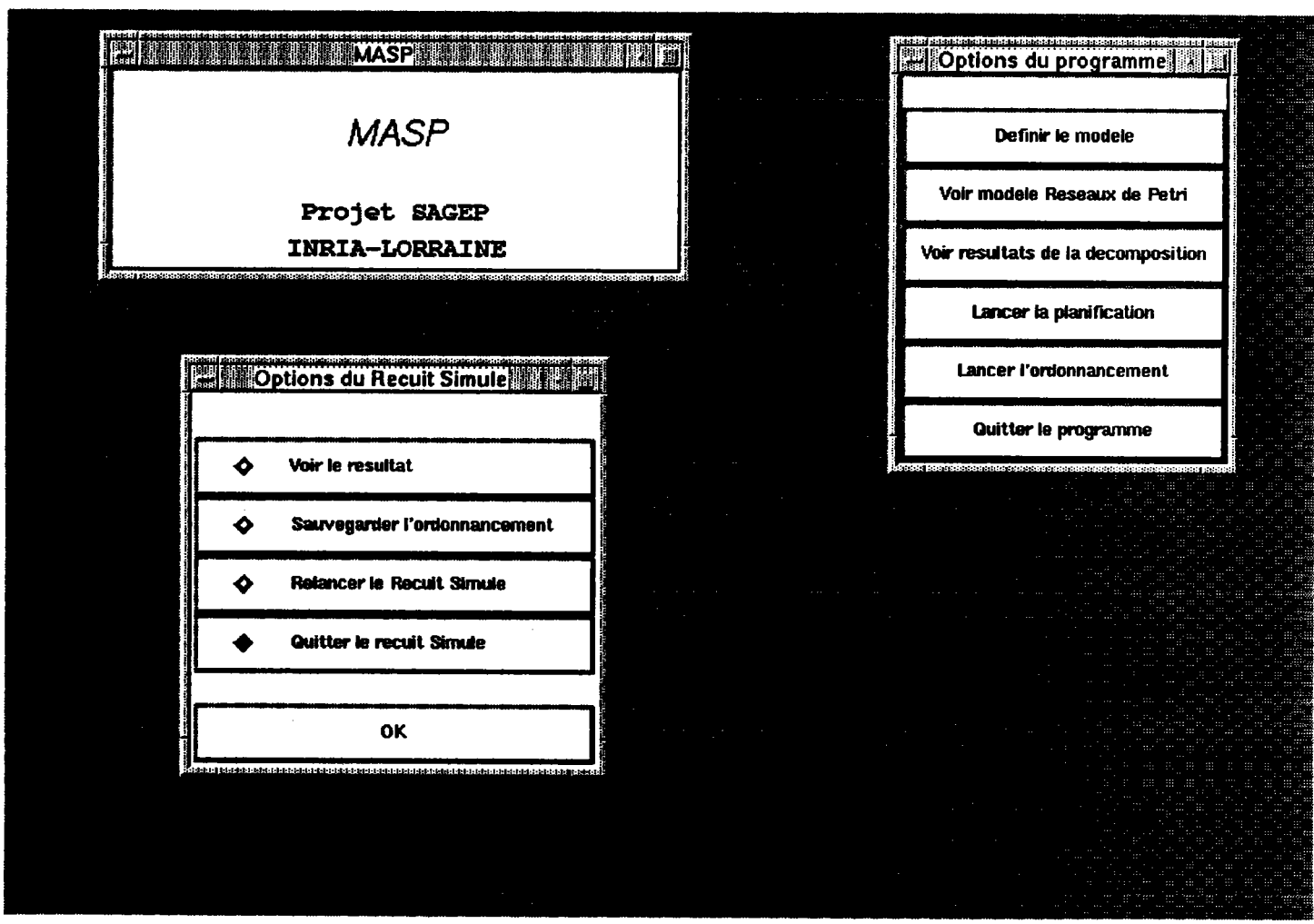


Figure A.12 : Option de la méthode basée sur le recuit simulé