N^o D'ORDRE :

UNIVERSITE PARIS XI
UFR SCIENTIFIQUE D'ORSAY

THESE

Présentée

Pour obtenir

LE GRADE DE DOCTEUR EN SCIENCES

DE L'UNIVERSITE PARIS XI ORSAY

PAR

Liliana Cucu

Sujet : **Ordonnancement non préemptif et condition d'ordonnançabilité pour systèmes embarqués à contraintes temps réel**

Soutenue le 28 Mai 2004 devant la Commission d'examen

M. Philippe Chrétienne	Rapporteurs
M. Yvon Trinquet	
M. Francis Cottet	Examinateurs
M. Joël Goossens	
M. Alain Mérigot	
M. Yves Sorel	

À mes parents.

Remerciements

Je remercie d'abord Yves SOREL, mon directeur de thèse pour m'avoir proposé un sujet de thèse passionnant et pour avoir eu la patience de m'encadrer pendant tout ce temps, en trouvant les bons mots pour me motiver dans les moments difficiles.

Je remercie également Philippe CHRETIENNE et Yvon TRINQUET d'avoir accepté la lourde tâche d'être rapporteurs de cette thèse, leurs commentaires ont augmenté mon recul par rapport aux domaines traités. Je remercie de même Francis COTTET, Joël GOOSSENS et Alain MERIGOT de m'avoir fait l'honneur de faire partie de mon jury. Leurs commentaires sur le manuscrit m'ont permis de le rendre plus explicite.

Mes remerciements s'adressent plus particulièrement à Thierry GRANDPIERRE et à Rémy KOCIK pour m'avoir aidé au début de cette thèse à découvrir la recherche. Je remercie également Antoine VIEL et Arnaud ROUANET qui m'ont aidé à comprendre beaucoup de choses sur l'informatique et l'amitié. Le temps n'aurait pas passé si vite s'il n'y avait pas eu Christophe MACABIAU, Frédéric GAGER, Julien FORGET, Nicolas PERNET, Cyril FAURE et Olivier MARCHETTI. Que ceux avec qui j'ai partagé le bureau, Claire MORISSET et Frédéric ROLIN, trouvent ici un peu de silence.

Je voudrais remercier aussi mes amis qui m'ont soutenu, pour tout ce que je leur ai fait subir pendant tout ce temps, chacun à sa façon m'a aidé à traverser ces quelques trois ans de ma vie.

Je tiens remercier mes colocataires, Adriana et Lucian, qui à part la chaleur que seule une famille peut donner m'ont apporté leur soutien pendant la rédaction du manuscrit.

Enfin, je remercie Samuel qui m'a soutenu, encouragé et supporté. Sans lui, cette thèse n'aurait été qu'une thèse et certainement la vie n'aurait pas été si belle.

Table des matières

Introduction

Contexte

Les systèmes temps réel sont de plus en plus utilisés dans le monde contemporain. Longtemps réservés aux équipements industriels lourds (centrale nucléaire, chaîne de fabrication, avionique, systèmes d'armes), leurs domaines d'utilisation sont aujourd'hui très variés, on peut les trouver dans des produits grand public (automobile, téléphone, domotique) pour lesquels les temps de développement conditionnant la mise sur le marché doivent être minimisés.

Ces systèmes ne doivent pas seulement réaliser correctement des actions mais aussi les réaliser dans un temps borné. Dans un système de pilotage automatique d'un avion, par exemple, celui-ci doit prendre la bonne décision de contourner un obstacle en respectant certaines contraintes temporelles. Ces contraintes, dont le non-respect peut avoir des conséquences catastrophiques, s'appellent « contraintes temps réel dures ». En revanche, dans un système tel qu'un réseau de téléphonie mobile, le non-respect des contraintes temporelles peut entraîner des conséquences moins graves comme la perte de signal ou un décalage dans la conversation. Ces contraintes s'appellent « contraintes temps réel souples ».

Un système temps réel embarqué est composé d'un calculateur qui exécute un ensemble de programmes en interagissant avec son environnement physique dont les changements d'états sont perçus par le calculateur au moyen de capteurs. Ces programmes forment le logiciel du calculateur, ils décrivent les algorithmes applicatifs à réaliser. Chaque algorithme définit un ordre total sur l'exécution d'un ensemble d'opérations arithmétiques et logiques. Le calculateur réagit aux changements d'états de l'environnement pour le maintenir dans un état déterminé au moyen d'actionneurs. Un système temps réel est qualifié de réactif puisqu'il est en permanente interaction avec l'environnement. Enfin, la plupart de ces systèmes (automobiles, avions, téléphone, etc) sont aussi embarqués dans le sens où on les trouve dans des équipements qui sont mobiles et pour lesquels les aspects d'encombrement, de poids, de consommation doivent être pris en compte. L'aspect embarqué impose de garantir le respect des contraintes temps réel tout en minimisant le coût et la taille de l'architecture matérielle.

Les programmes d'un système temps réel reposent principalement sur des algorithmes applicatifs de contrôle-commande, de traitement du signal et des images qui nécessitent d'importantes quantités de calculs. Lorsque ces derniers doivent être effectués en un temps borné, il faut utiliser des calculateurs de fortes puissances. Les limites technologiques et le coût des calculateurs puissants basés sur un unique processeur, diminuent souvent l'intérêt

de leur utilisation. Il faut alors utiliser des calculateurs multiprocesseur basés sur le fonctionnement en parallèle de plusieurs processeurs. Enfin, les calculateurs utilisés sont qualifiés d'hétérogènes car les types des processeurs et des média de communication qui les composent sont souvent différents.

Le problème général des systèmes temps réel consiste à distribuer les programmes sur les processeurs composant l'architecture matérielle et pour chaque processeur, celui-ci étant une machine séquentielle, il faut ordonnancer l'ensemble des programmes qui y ont été distribués.

Objectifs

Cette thèse s'inscrit dans le cadre des recherches menées dans le projet OSTRE (Optimisation de Systèmes Temps Réel Embarqués) de l'INRIA Rocquencourt, plus précisément menées sur la méthodologie adéquation algorithme-architecture (AAA) pour les systèmes temps réel distribués embarqués hétérogènes. Cette méthodologie permet d'améliorer la réalisation de ces systèmes principalement fondée sur la distribution et l'ordonnancement, en assurant une bonne sécurité de fonctionnement et en réduisant la durée de cette réalisation. Pour cela on a besoin de modèles mathématiques capables de prendre en compte toutes les caractéristiques ainsi que la diversité des systèmes temps réel.

Tout d'abord, un système temps réel est un système réactif qui réagit de manière permanente avec l'environnement physique à travers des capteurs et des actionneurs. Des stimuli sont consommées par le système à travers des capteurs et le système produit des réactions vers l'environnement à travers des actionneurs. Ces stimuli et réactions correspondent à une répétition infinie d'opérations et ce comportement est en général décrit en imposant des « contraintes de périodicité ». Notre modèle doit pouvoir exprimer des contraintes de périodicité.

En outre, un délai est en général imposé entre la consommation d'un stimulus par le système et la production de la réaction correspondant à ce stimulus, car le temps de réaction d'un système avec contraintes temps réel dures est borné afin d'éviter des conséquences catastrophiques dues au mauvais contrôle de l'environnement. Notre modèle doit pouvoir exprimer ce délai. Pour cela on propose une nouvelle contrainte appelée « contrainte de latence » mieux adaptée que d'autres contraintes pour décrire ce délai.

Les systèmes temps réel reposent le plus souvent sur des algorithmes de contrôle-commande, de traitement du signal et des images dont les opérations peuvent avoir des « contraintes de précédence » l'une par rapport l'autre. Notre modèle doit pouvoir aussi exprimer des contraintes de précédence.

Nous cherchons donc à proposer un modèle qui prend en compte toutes les contraintes que nous venons d'exposer afin de l'utiliser pour poser le problème d'ordonnancement monoprocesseur et le problème de distribution et d'ordonnancement multiprocesseur.

Nous cherchons ensuite à résoudre ces deux problèmes dans le cas où l'ordonnancement est de type hors ligne non préemptif car cette approche est bien adaptée aux systèmes embarqués réactifs reposant sur des algorithmes de contrôle-commande, de traitement du signal et des images. En effet, comme nous nous intéressons aux systèmes avec contraintes

temps réel dures les choix de distribution et d'ordonnancement doivent être déterministes. Pour cela nous ne traitons pas l'apparition dans le système d'opérations non prévues, ce qui permet d'utiliser des approches hors ligne aussi bien pour l'ordonnancement que pour la distribution. Ces approches ont de plus l'avantage d'être les moins coûteuses en termes de surcoût. Cela correspond parfaitement à la caractéristique « embarqué » de ces systèmes. Par ailleurs, nous choisissons l'approche non préemptive parmi les trois approches possibles pour l'ordonnancement du point de vue de la préemption : non préemptive, préemptive sans prendre en compte le coût de la préemption et préemptive avec prise en compte du coût de la préemption. En effet, seulement le premier et le troisième choix permettent de traiter des cas réalistes sans risquer de ne pas respecter une contrainte temps réel [1], et il est logique de commencer à traiter le premier. C'est ce que nous faisons dans cette thèse, en ayant comme perspective de traiter ensuite le troisième choix.

On est ici dans le cas des contraintes temps réel dures et dorénavant quand il n'y a pas de confusion possible on parlera de contraintes temps réel, tout court.

Plan de la thèse

La thèse contient, après une introduction qui décrit le contexte et les objectifs, trois chapitres, suivis par une conclusion et des perspectives.

Le premier chapitre présente les résultats existants afin de familiaser le lecteur aux notations utilisées par la suite. Ces notations sont propres à cette thèse et chaque fois qu'il y a une notation différente de celle utilisée par le reste de la communauté temps réel pour désigner une notion commune, on explique les équivalences ou les différences qu'il peut y avoir entre ces notations. Dans la présentation des notations, cette thèse définit certaines notions qui ont été utilisées par la communauté temps réel, mais pour lesquelles il manquait des définitions précises. Cela est une tentative de supprimer l'ambiguité sur des notions comme dynamique/statique ou en ligne/hors ligne. Nous avons choisi de présenter parmi les résultats existants ceux qui nous ont aidés à mieux comprendre notre problème ou ceux qui ont servi d'exemple de raisonnement afin d'obtenir nos résultats. Ces choix ont motivé nos travaux. La fin du premier chapitre justifie les choix importants faits pour poser notre problème de distribution et d'ordonnancement, c'est-à-dire la prise en compte de contraintes de précédences et de périodicités dans le cas non-préemptif et l'introduction d'une contrainte de latence. D'une part ces choix reprennent des contraintes qui existent déjà et d'autre part on explique le choix d'une nouvelle contrainte de latence qui est mieux adaptée que d'autres contraintes, appelées contraintes relatives, qui décrivent des délais. On explique aussi que même si les contraintes de périodicités et de précédences existaient déjà, il n'y a aucun résultat jusqu'à présent qui permet de prendre en compte des opérations qui ont des contraintes de précédence et ont des valeurs différentes pour leur périodes. Cette thèse fait aussi une étude complète sur l'interaction entre les contraintes de précédences et de périodicités.

Le deuxième chapitre est le plus important de la thèse. Il présente des résultats obtenus dans le cas monoprocesseur. L'ordre de présentation des deux premiers sous-chapitres sur l'ordonnancement avec contraintes de précédences et de périodicités et sur l'ordonnancement

avec contraintes de précédences et de latences, est indifférent. En revanche le troisième sous-chapitre sur l'ordonnancement avec contraintes de précédences, de périodicités et de latences utilise les résultats des deux sous-chapitres précédents.

Pour le problème de l'ordonnancement des systèmes avec contraintes de précédences et de périodicités, les études d'ordonnançabilité nécessitaient un algorithme optimal d'ordonnancement qui n'introduisait pas de temps creux. On a donc proposé un algorithme qui a la particularité de rendre périodique toutes les opérations non-périodiques qui n'ont aucun successeur périodique si on arrive à les ordonnancer au moins une fois. Cette particularité nous a permis d'effectuer ensuite l'étude d'ordonnançabilité et d'obtenir un test d'ordonnançabilité. En conséquence l'ordre de présentation des résultats est un ordre naturel puisque l'étude d'ordonnançabilité utilise les résultats de l'algorithme d'ordonnancement. Un important défi comme suite de cette thèse consistera à introduire la préemption qui impose la modification du modèle proposé actuellement, mais qui on l'espère permettra de trouver aussi une condition nécessaire d'ordonnancement pour les cas où on prend en compte la préemption.

Si pour le premier problème on a pu s'inspirer des travaux existants, la nouveauté de la latence nous a rendu les choses plus difficiles pour le problème d'ordonnancement des systèmes avec contraintes de précédences et de latences. L'algorithme d'ordonnancement pour ce dernier problème utilise les résultats de l'étude d'ordonnançabilité. Celle-ci nous a permis d'identifier les opérations qui interviennent dans une contrainte relative et l'influence réciproque de ces opérations dans le cas où l'on a plusieurs contraintes relatives. On a formalisé cela à l'aide de trois relations entre des paires d'opérations sur lesquelles une contrainte de latence a été imposée, ce qui nous a permis d'obtenir une condition d'ordonnançabilité. Cette étude est la première du genre et étant donné que la latence généralise les autres contraintes relatives, on envisage d'obtenir par la suite de nouveaux résultats intéressants.

On traite ensuite le problème d'ordonnancement qui prend en compte les trois contraintes : précédences, périodicités et latences. L'étude d'ordonnançabilité étant nécessaire pour l'algorithme d'ordonnancement, on commence par celle-ci. En combinant l'étude d'ordonnançabilité et l'algorithme d'ordonnancement, on obtient un test d'ordonnançabilité. Ces résultats ont eu comme conséquence de mettre en évidence un lien entre notre problème et le problème classique temps réel d'ordonnancement d'opérations avec contraintes de périodicités et d'échéances. Ce dernier est un cas particulier de notre problème et on donne un algorithme qui construit à partir d'une instance du problème classique une instance de notre problème. Ce résultat pourra être utilisé par la suite pour prendre en compte la gigue (variations faibles de périodicité) dans notre problème.

Le troisième chapitre traite les trois problèmes précédents mais dans le cas multiprocesseur pour lequel on prend en compte les échanges de données entres les opérations lorsqu'elles sont distribuées sur des processeurs différents. On prouve que ces problèmes sont NP-difficiles. En conséquence, on propose des heuristiques qui utilisent les résultats monoprocesseur. Chaque heuristique est comparée à un algorithme exact et on montre que l'heuristique est plus rapide que l'algorithme exact. Par la suite on pourrait utiliser le lien entre notre modèle et le modèle classique afin de proposer de nouvelles heuristiques pour le cas multiprocesseur du problème classique.

Chapitre 1

État de l'art et motivations de la thèse

1.1 Notions importantes en ordonnancement

On définit tout d'abord de manière informelle les notions d'« ordonnancement classique » et d'« ordonnancement temps réel » [2], puis d'ordonnançabilité, notions utilisées dans ce chapitre pour différencier les diverses approches possibles et indiquer celles que nous avons plus particulièrement étudiées dans le cadre de cette thèse. On donnera par la suite au chapitre 2 une définition plus formelle de ces notions.

Effectuer un ordonnancement classique monoprocesseur consiste à déterminer pour un ensemble d'opérations dans quel ordre relativement à une date, chacune d'elles doit être exécutée sur une unique ressource. Une opération peut s'exécuter quand toutes les informations dont elle a besoin sont disponibles, elle est alors elle-même dite « disponible ». Le terme opération est pris au sens large, il peut par exemple correspondre à différentes actions à réaliser sur une pièce mécanique (découpe, usinage, finition), ou bien un ensemble de calculs nécessaires pour réaliser un algorithme de traitement du signal (filtrage, évaluation de l'erreur résiduelle, gradient adaptatif).

Un ordonnancement temps réel revient à déterminer pour chacune de ces opérations une date de début d'exécution de l'opération permettant de respecter des contraintes relatives à l'écoulement du temps appelées « contraintes temps réel ». Un ensemble d'opérations sur lequel on impose des contraintes temps réel est habituellement appelé « système temps réel » que l'on notera par la suite « système d'opérations ». Il peut aussi avoir des contraintes relatives à d'autres critères, par exemple des contraintes de précédence. Un ensemble d'opérations est ordonnançable s'il y a au moins un ordonnancement qui satisfait toutes les contraintes. Il est souvent possible et utile de trouver des conditions d'ordonnançabilité permettant d'assurer qu'un ensemble d'opérations est ordonnançable sans avoir à examiner tous les ordonnancements possibles.

Quand on parle d'ordonnancement classique on fait référence aux résultats obtenus en théorie de la complexité et en recherche opérationnelle. Quand on parle d'ordonnancement temps réel on fait référence aux résultats obtenus pour l'ordonnancement de systèmes temps réel. Etant donné que dans cette thèse on emploie beaucoup plus souvent la notion d'ordon-

nancement temps réel que la notion d'ordonnancement classique par la suite on fera référence implicitement à l'ordonnancement temps réel quand on parle d'ordonnancement (tout court).

Dans l'état de l'art qui suit on utilisera fréquemment les notions d'ordonnancement en ligne ou hors ligne, d'algorithme d'ordonnancement dynamique ou statique, d'ordonnancement préemptif ou non-préemptif, et enfin d'algorithme d'ordonnancement qui introduit ou non des temps creux. On définit tout d'abord les trois premières notions. Pour bien comprendre quelles sont leurs différences on va s'appuyer sur les réponses à trois questions, détaillées ci-dessous :

- **Quand est réalisé l'ordonnancement ?**
  La réponse à cette question fait la différence entre un ordonnancement en ligne et un ordonnancement hors ligne.
  Un ordonnancement obtenu pendant l'exécution de l'ensemble d'opérations s'appelle « ordonnancement en ligne ». L'algorithme d'ordonnancement appliqué en ligne a un coût d'exécution en temps réel qui s'ajoute au coût d'exécution des opérations elles-mêmes, mais il a l'avantage de permettre l'ordonnancement d'opérations non prévues au départ dans l'ensemble d'opérations.
  Un ordonnancement obtenu avant l'exécution de l'ensemble d'opérations s'appelle « ordonnancement hors ligne». L'algorithme d'ordonnancement appliqué hors ligne n'a pas de coût d'exécution en temps réel, et l'ensemble d'opérations doit être entièrement connu pour construire l'ordonnancement. Ce dernier est construit avant l'exécution et aucun choix d'ordonnancement n'est fait pendant l'exécution.
- **Quel est l'algorithme d'ordonnancement ?**
  La réponse à cette question fait la différence entre un algorithme dynamique et un algorithme statique d'ordonnancement.
  La notion d'algorithme dynamique/statique d'ordonnancement est souvent confondue avec celle d'ordonnancement en ligne/hors ligne et elle n'a de sens que pour les tâches périodiques. Les notions de dynamique et de statique décrivent des propriétés de l'algorithme d'ordonnancement, alors que les notions de en ligne et de hors ligne précisent le moment où l'algorithme d'ordonnancement est utilisé, ce qui est très différent.
  Un algorithme statique d'ordonnancement détermine la date d'exécution d'une opération choisie parmi les opérations disponibles en fonction de l'ensemble total des opérations du système. Donc le choix d'ordonnancer une opération disponible et non pas une autre opération disponible n'est pas fait en prenant en compte les opérations disponibles, mais en prenant en compte l'ensemble total des opérations au début de l'ordonnancement. Cela implique que l'ordonnancement d'une opération ne dépend pas du temps écoulé depuis le début de l'ordonnancement, qu'il soit réalisé en ligne ou hors ligne.
  En conséquence même utilisé en ligne, un algorithme statique ne sait pas prendre en compte les opérations non prévues car il doit connaître l'ensemble total des opérations avant de commencer l'ordonnancement. Dans le cas où les opérations peuvent avoir des durées variables dues à des choix de séquence d'instructions différentes, la connaissance

totale de l'ensemble d'opérations permet d'évaluer plus facilement le pire cas d'une opération c'est à dire le temps maximum nécessaire à l'exécution de l'opération.
Un algorithme dynamique d'ordonnancement détermine la date d'exécution d'une opération choisie parmi les opérations disponibles à un instant donné en fonction de l'ensemble des opérations disponibles à cet instant. Donc le choix d'ordonnancer une opération disponible et non pas une autre opération disponible n'est pas fait en prenant en compte toutes les opérations du système, mais en prenant en compte les opérations disponibles à l'instant auquel on fait ce choix. Cela implique que l'ordonnancement d'une opération dépend du temps écoulé depuis le début de l'ordonnancement et reste valable même si l'ordonnancement est fait hors ligne.
Faire le choix d'ordonnancer une opération en utilisant seulement l'ensemble d'opérations disponibles permet de prendre en compte des opérations non prévues. En effet, quand apparaît une telle opération, elle devient disponible et le choix de l'ordonnancer peut être fait. Evidemment un algorithme dynamique doit être utilisé en ligne pour prendre en compte des opérations non prévues. Le désavantage de ce type d'algorithme reste la détermination du pire cas pour une opération.

- **L'ordonnancement est-il préemptif?**
  La préemption découle de la capacité qu'ont les processeurs à être sensibles à des interruptions. Ces dernières ont comme origine des événements extérieurs au processeur qui peuvent être périodiques ou apériodiques ou des événements intérieurs au processeur. Ces derniers peuvent correspondre par exemple au passage par une valeur particulière d'un compteur incrémenté par son horloge interne pouvant définir une période ou la disponibilité d'une opération devenue prioritaire.
  Dans un ordonnancement préemptif, les opérations ont le droit d'être interrompues pendant leur exécution par d'autres opérations qui sont dites plus prioritaires, alors que ce n'est pas le cas d'un ordonnancement non-préemptif. Cela permet de réaliser des ordonnancements plus fins dans le sens où la préemption permet de découper une opération en morceaux afin d'y intercaler une autre opération ou un autre morceau d'opération, et donc plus généralement d'avoir plus de chances de trouver un ordonnancement respectant les contraintes. Cependant l'utilisation de la préemption induit systématiquement un coût supplémentaire lors de l'exécution temps réel car il faut sauver et restaurer le contexte avant et après l'interruption, ce qui conduit à ajouter des opérations d'écriture et de lecture. Ce principe a aussi un coût en termes d'occupation mémoire puiqu'il faut mémoriser les informations à sauver et restaurer. Le coût de la préemption doit impérativement être pris en compte au risque de conduire à des ordonnancements qui s'avèreront faux en temps réel car le coût de la préemption a été supposé négligeable alors qu'il ne l'était pas [1].

L'énumération des notions différentiant les approches possibles pour obtenir un ordonnancement montre que seul un ordonnancement en ligne utilisant un algorithme d'ordonnancement dynamique permet de prendre en compte les opérations non prévues. Cela est récapitulé dans de tableau 1.1 qui décrit tous les cas possibles et leurs comportements par rapport aux opérations non prévues. On note par X la possibilité de traiter les opérations

non prévues, par O le cas contraire. On note par EL l'ordonnancement en ligne, par HL l'ordonnancement hors ligne, par S l'algorithme statique et par D l'algorithme dynamique.

	EL/ S	EL/ D	HL/ S	HL/ D
P	X	X	X	X
NP	O	X	O	O

FIG. 1.1 – *Tableau des approches possibles d'ordonnancement*

Dans l'état de l'art qui suit on utilisera aussi la notion d'algorithme d'ordonnancement qui introduit ou non des temps creux [3]. Un algorithme d'ordonnancement qui introduit des temps creux est un algorithme qui peut choisir de ne pas ordonnancer des opérations alors qu'elles sont disponibles. On appelle "ordonnancement avec des temps creux" un ordonnancement obtenu en utilisant un algorithme d'ordonnancement qui introduit des temps creux.

Après avoir passé en revue ces différentes notions, on peut maintenant présenter les résultats existant dans le cas monoprocesseur et dans le cas multiprocesseur pour ce que nous appelons un ensemble d'opérations soumises à des contraintes temps réel. Afin de ne pas perturber le lecteur on utilisera dans cet état de l'art les notions classiques utilisées habituellement dans la littérature, de "système de tâches temps réel" et de "tâches temps réel".

1.2 Résultats existant dans le cas monoprocesseur

Ce chapitre est composé de trois sous-chapitres. Le premier présente les résultats pour les systèmes de tâches avec des contraintes de périodicité. Cela nous permet de rappeler quelques modèles existant pour ce type de système temps réel. Le deuxième traite des systèmes de tâches avec contraintes de précédence et la plupart des résultats existant viennent de l'ordonnancement classique. Les résultats sont présentés en suivant les principales classes de critères utilisés pour ces problèmes. Le troisième présente les quelques résultats qui existent pour les systèmes avec contraintes de précédences et de périodicités.

1.2.1 Contraintes de périodicité

Dans la littérature sur le temps réel, on trouve deux types de contraintes temps réel : les contraintes temps réel absolues et les contraintes temps réel relatives. Une contrainte absolue est définie en utilisant les caractéristiques d'une seule tâche. C'est, par exemple, le cas de l'échéance. Une contrainte relative est définie en utilisant les caractéristiques d'au moins deux tâches. C'est, par exemple, le cas d'une contrainte de bout-en-bout.

On commence par présenter des résultats obtenus pour le cas de contraintes absolues.

Le modèle introduit par Liu et Layland [4] pour les systèmes temps réel périodiques est le plus utilisé dans le monde des systèmes temps réel. Dans ce modèle une tâche A possède

une date de réveil r_A à laquelle elle devient disponible, une échéance D_A définie par rapport à la date de réveil, une période T_A et une durée d'exécution C_A qui est égale à la pire durée d'exécution de la tâche sur le processeur c'est à dire le temps maximum nécessaire pour l'exécution de la tâche. Si le premier réveil de la tâche se fait à r_A, le i^{me} réveil se fait à $r_A + (i-1)T_A$, où T_A est la période de la tâche. L'ordonnancement est un ordonnancement préemptif.

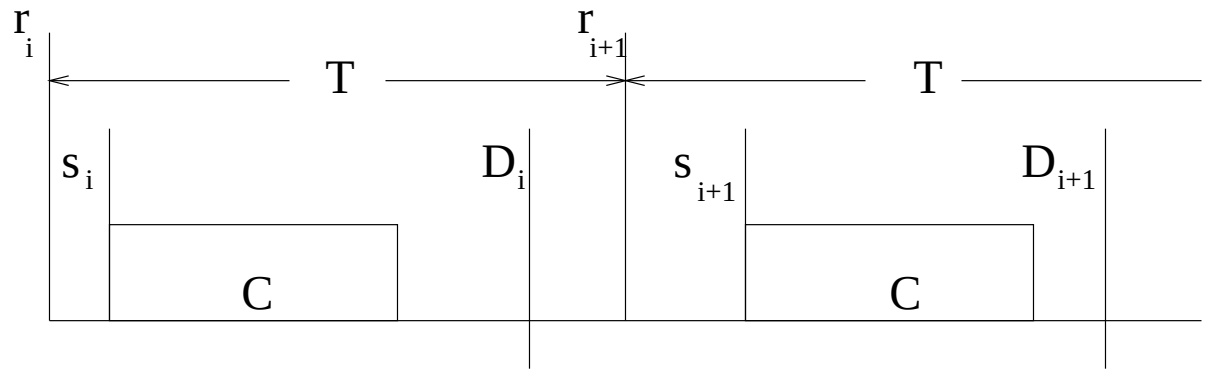

FIG. 1.2 – *Modèle proposé par Liu and Layland*

Dans le même article [4], pour ce modèle, Liu et Layland donnent deux algorithmes d'ordonnancement, un algorithme statique et un algorithme dynamique et ils font en outre une étude d'ordonnançabilité.

L'algorithme statique appelé algorithme "rate monotonic" traite le cas où la période est égale à l'échéance. L'algorithme donne aux tâches une priorité statique inverse à leurs périodes. Alors, les tâches avec la plus petite période sont les plus prioritaires. L'étude d'ordonnançabilité donne une condition suffisante formulée par le théorème suivant :

Théorème 1 *[4] Un système de tâches périodiques peut être ordonnancé en utilisant l'algorithme "rate monotonic" si*

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1)$$

Pour le cas des tâches avec les échéances plus petites que les périodes $D < T$, l'algorithme "rate monotonic" n'est plus optimal. Dans ce cas, les deux auteurs présentent un algorithme dynamique appelé "EDF" (earliest deadline first) qui donne la plus grande priorité aux tâches avec l'échéance la plus proche. L'algorithme est prouvé optimal dans le sens où si un système de tâches peut être ordonnancé en utilisant une politique quelconque d'affectation de priorités, alors le système peut être ordonnancé aussi avec l'algorithme EDF. L'étude d'ordonnançabilité donne une condition nécessaire et suffisante formulée par le théorème suivant :

Théorème 2 *[4] Un système de tâches périodiques peut être ordonnancé en utilisant l'algorithme EDF si et seulement si*

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

En utilisant le même modèle avec préemption, Leung et Merrill [5] prouvent l'existence d'un intervalle de temps borné nécessaire pour décider l'ordonnançabilité d'un système de tâches. Ce résultat est donné par le théorème suivant :

Théorème 3 *[5] Un système de tâches périodiques est ordonnançable si et seulement si toutes les échéances de tâches sont satisfaites dans l'intervalle de temps* $[0,s+2T]$*, où* s *est la plus grande date de réveil de l'ensemble de tâches et* T *le plus petit multiple commun des périodes des tâches.*

Pour prouver ce résultat, les auteurs donnent un lemme qui prouve l'existence d'un motif dans l'ordonnancement à partir de $s+P$. Dans la preuve, ils utilisent aussi l'optimalité de l'algorithme EDF. Ce résultat est amélioré ensuite par Grolleau et autres [6] qui prouvent l'existence d'un motif dans l'ordonnancement à partir du dernier temps creux acyclique de l'ordonnancement. De cette manière, les auteurs prouvent que $s+2P$ est une borne supérieure. La notion de temps creux est définie en détails dans [3].

On a vu que Liu et Layland donnent une condition d'ordonnançabilité suffisante qui n'est pas nécessaire pour qu'un système de tâches soit ordonnancé avec l'algorithme "rate monotonic". Cette étude d'ordonnançabilité a été complétée par Lehoczky et autres [7] qui donnent un test nécessaire et suffisant pour qu'un système de tâches soit ordonnancé avec l'algorithme "rate monotonic". Le test contient deux parties : la première partie permet d'étudier d'une manière itérative l'ordonnançabilité de chaque tâche et cela dans l'ordre croissant des périodes, la deuxième partie est une généralisation du premier test et il permet de conclure si le système de tâches peut être ordonnancé en utilisant l'algorithme "rate monotonic". Dans le même article, les auteurs font une étude stochatisque qui présente la distribution de performances de l'algorithme pour des tâches générées d'une manière aléatoire.

En utilisant l'algorithme "rate monotonic", Orozco et autres [8] proposent un algorithme d'ordonnancement pour les systèmes de tâches qui ont deux parties : une obligatoire et une facultative. L'algorithme a deux parties : une partie utilisée pour chaque ordonnancement et une deuxième partie facultative qui sert à améliorer les ordonnancements obtenus lors de la première partie.

Jusqu'à maintenant on a parlé de systèmes de tâches périodiques avec des échéances plus petites ou égales aux périodes. Quand aucune relation n'est précisée entre les périodes et les échéances des tâches, on parle d'échéances arbitraires. Pour ce problème Lehoczky [9] donne un critère général pour qu'un système de tâches périodiques puisse être ordonnancé en utilisant un algorithme statique, donc avec priorités fixes pour les tâches. Il généralise les résultats obtenus par Liu et Layland concernant le pire cas du taux d'utilisation $\sum_{i=1}^{n} \frac{C_i}{T_i}$ d'un processeur.

Les tâches qui n'ont pas de dates de réveil périodiques sont des tâches non-périodiques. Il y a deux types de tâches non-périodiques : apériodiques et sporadiques. Une tâche apériodique est une tâche pour laquelle il n'y a aucune relation entre ses dates de réveils. Pour une tâche sporadique on connaît le temps minimum m écoulé entre deux dates de réveil consécutives. Pour ce dernier type de tâche, Audsley et autres [10] prouvent qu'en utilisant l'algorithme EDF leurs échéances sont garanties. Pour cela ils prouvent que dans le pire des cas une tâche

sporadique se comporte comme une tâche périodique avec la période m et l'échéance D, où D est l'échéance de la tâche sporadique.

Les tâches peuvent aussi avoir, lorsqu'elles deviennent disponibles, des durées différentes dûes par exemple à la présence d'instructions conditionnelles dans les corps des tâches. Dans ce cas, Pailler et autres [11] utilisent les réseaux de Petri pour modéliser ces systèmes et ils donnent une méthode d'ordonnancement hors ligne pour ce type de systèmes.

On peut aussi imposer des contraintes sur les dates de réveil d'un sous-ensemble de tâches qui ont la même période, c'est à dire qu'il y a un intervalle fixé entre les dates de réveil de tâches appartenant au sous-ensemble. De cette manière les dates de réveil sont liées entre elles et forment un motif de dates de réveil. Ce type de contraintes s'appellent "time-offsets". Tindell fait une étude d'ordonnançabilité qui permet d'introduire les nouvelles contraintes et de déterminer dans quel cas un système est ordonnançable [12]. L'auteur continue l'étude de systèmes "time-offsets", en donnant un test d'ordonnançabilité [13].

Un autre cas particulier concerne les systèmes de tâches "offset free", c'est à dire des tâches dont les dates de réveil peuvent varier. Dans ce cas, Goossens et autres prouvent que l'algorithme "rate monotonic" n'est plus optimal parmi les algorithmes statiques d'ordonnancement [14].

Dans le cas monoprocesseur les tâches peuvent aussi partager des ressources. Pour le cas de tâches avec des dates de réveil $s_A = 0, \forall A \in V$, donc le cas de tâches disponibles à l'instant $t = 0$, Choquet et autres [15] proposent un modèle utilisant les réseaux de Petri qui est ensuite utilisé pour donner un algorithme d'ordonnancement pour les systèmes de tâches qui partagent des ressources et qui communiquent par message. L'algorithme permet de trouver tous les ordonnancements possibles en utilisant des techniques afin de réduire le temps de construction des ordonnancements.

Une autre technique pour trouver un ordonnancement en minimisant les blocages dus à l'accès aux ressources est donné par Baker [16]. Cette méthode appelée "stack-based scheduling" est prouvée optimale dans le sens où elle est moins restrictive que d'autres méthodes et elle prévient les blocages pour un ordonnancement obtenu avec l'algorithme "rate monotonic" et l'algorithme EDF.

Dans le cas où les tâches partagent des ressources, une tâche peut attendre un message d'une autre tâche afin d'être "réveillée". On appelle ce type de flexibilité de la date de réveil, la gigue de date de réveil. Audsley et autres proposent l'utilisation de l'analyse holistique pour déterminer l'ordonnançabilité d'un système temps réel qui peut avoir de la gigue de date de réveil pour certaines tâches [17], [18].

Un cas particulier de contraintes temps réel sont les contraintes temps réel faibles-dures proposées par Bernat et Burns [19]. Les tâches doivent satisfaire m échéances sur toutes les k répétitions consécutives. Des résultats d'ordonnancement sont obtenus par les mêmes auteurs en [20] et Pogi et autres proposent une version modifiée de ces résultats [21]. L'article présente des études de performances.

Tous les résultats présentés jusqu'à maintenant utilisent la préemption. Jeffay et autres [1] expliquent aussi l'importance de l'ordonnancement sans préemption. Tout d'abord, pour des raisons pratiques (la partie physique ou logiciel du système temps réel ne le permet pas),

l'utilisation de la préemption n'est pas toujours possible. Les algorithmes non préemptifs sont plus faciles à implanter et ils peuvent avoir un surcoût lors de l'exécution réelle inférieur à un algorithme préemptif. Le surcoût des algorithmes préemptifs est plus difficile à caractériser et à prévoir. Comme le surcoût d'ordonnancement est souvent ignoré dans les modèles théoriques, l'implantation d'un algorithme non préemptif est plus proche du modèle théorique. L'ordonnancement non préemptif garantit un accès exclusif aux ressources, qui élimine le besoin de synchronisation. Le problème d'ordonnancer des tâches sans préemption forme une base théorique pour des modèles de tâches plus généraux. Le même article présente une étude d'ordonnançabilité pour un système de tâches périodiques ou sporadiques avec dates de réveil arbitraires sans préemption et sans temps creux ayant comme conclusions les deux théorèmes suivants :

Théorème 4 *Si un système de tâches est ordonnançable alors on a :*
- $\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$
- $\forall i,\ 1 < i \leq n,\ \forall L, T_1 < L < T_i$ *:*

$$L \geq C_i + \sum_{j=1}^{i-1} \frac{L-1}{T_j} C_j$$

Théorème 5 *Si un système de tâches périodiques ou sporadiques avec dates de réveil arbitraires satisfait les condition du théorème 4, alors l'algorithme d'ordonnancement EDF non préemptif va ordonnancer tous les systèmes de tâches avec des dates de réveil données obtenues de premier système de tâches.*

Un système de tâches avec des dates de réveil données s'appelle un système de tâches concrètes. Les auteurs prouvent que le problème de décision pour qu'un système de tâches périodiques concrètes soit ordonnançable est NP-complet au sens fort, mais le même problème pour des systèmes de tâches sporadiques concrètes est pseudo-polynômial.

Toujours dans le cas non-preemptif, Yuan et Agrawala présente une méthode arborescente de recherche des ordonnancements qui satisfont les contraintes temps réel [22]. La méthode contient deux parties, une partie qui génère une séquence de tâches et une deuxième partie qui ordonnance les tâches.

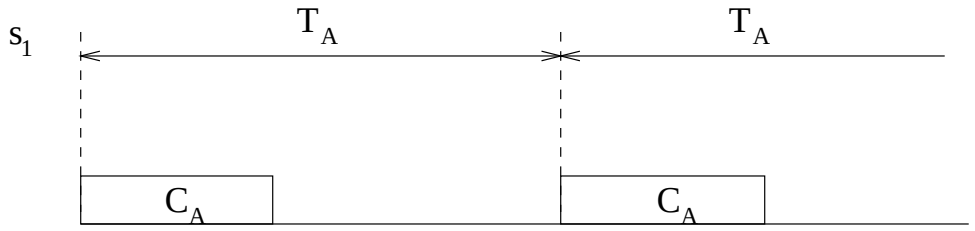

FIG. 1.3 – *Modèle de tâches périodiques strictes sans date de réveil*

Korst et autres [23] traitent un cas particulier de tâche périodique qui est la tâche périodique stricte sans date de réveil (voir figure 1.3). Cela signifie que le temps écoulé entre

deux ordonnancements consécutifs d'une tâche est le même pour tous les ordonnancements de la tâche. Ce type de problème est rencontré dans le contexte des modélisations de circuits intégrés qui implantent des algorithmes de traitement du signal. Les tâches utilisées par un tel algorithme sont en général élémentaires, c'est à dire elles ont la durée égale à 1, elles doivent être ordonnancées régulièrement pour des répétitions successives d'un signal et elles ne peuvent pas être préemptées à cause de leurs nature élémentaire.

Les auteurs prouvent que le problème de décision pour qu'un système de tâches périodiques strictes soit ordonnançable est NP-complet au sens fort, mais ils montrent aussi que le problème est polynômial pour le cas particulier avec périodes et durées d'exécution divisibles entre elles.

Les mêmes auteurs prouvent dans un autre article [24] que le problème de décision pour qu'un système de tâches périodiques soit ordonnançable est NP-complet au sens fort.

Sur les contraintes relatives appelées contraintes de bout en bout il y a peu de résultats dans la littérature et pourtant ce type de contrainte est très important pour les systèmes temps réel. Les contraintes de bout en bout sont imposées sur une entrée et une sortie. Par exemple, soit A la lecture de la valeur de la pression et B la modification de la température. Si on veut imposer la modification de la température 10 secondes après que la pression soit modifiée, on doit définir une contrainte de bout en bout [25]. Même si les contraintes de bout en bout sont souvent peu nombreuses dans un système temps réel, elles se propagent dans les composantes d'un système temps réel. L'algorithme proposé pour résoudre ce type de contraintes dans [25] définit des contraintes de périodicité pour un système qui ne possède à l'origine que des contraintes de précédence. Une autre particularité est que toutes les sorties peuvent avoir maximum un seul prédécesseur et cette hypothèse empêche d'avoir des précédences entre deux tâches quelconques.

Une approche statistique [26] est utilisée pour résoudre le problème avec contraintes de bout en bout mais sur un graphe de précédences particulières : les composantes connexes du graphe sont des chaînes. Chaque chaîne est formée d'une liste de tâches et possède des contraintes de bout en bout entre l'entrée et la sortie d'une tâche et on compte le nombre de tâches qui satisfont leurs contraintes.

La connaissance de la durée d'exécution des tâches d'un système n'est pas toujours possible avant l'exécution, donc les durées exactes d'exécution ne sont connues que lors de l'exécution du système. Gerber et autres étudient les contraintes de bout en bout dans ce cas [27]. Leur algorithme contient deux parties : une partie hors-ligne et une partie en-ligne. La partie hors-ligne de l'algorithme est une étude d'ordonnançabilité qui établit si les contraintes peuvent être satisfaites. Si c'est le cas, un ordonnancement provisoire est obtenu et l'algorithme en-ligne peut générer des limites supérieures et inférieures pour les dates de début des tâches à partir des dates de début et des durées d'exécution (qui sont connues à cette étape) de tâches précédentes.

Pour le cas des contraintes temps réel faibles-dures Lindsay étudie aussi des contraintes de bout en bout [28].

La liste des résultats présentés n'est pas exhaustive, elle sert à introduire le lecteur dans le monde de systèmes temps réel avec contraintes de périodicité dans le cas monoprocesseur.

1.2.2 Contraintes de précédence

Les principaux résultats concernant les problèmes d'ordonnancement de systèmes de tâches avec précédences viennent de l'ordonnancement classique. Les contraintes de précédence définissent un ordre partiel sur l'ensemble de tâches. Par exemple, si deux tâches A et B ont une contrainte de précédence, cela implique que B doit s'ordonnancer après que A ait été ordonnancée.

L'ordre partiel peut être facilement représenté à l'aide d'un graphe orienté ayant comme sommets les tâches et comme arcs les précédences entre les tâches. Dans la figure 1.4 où les contraintes de précédence sont modélisées à l'aide d'un graphe, A, B et D doivent être ordonnancées dans cet ordre et de même A, C et D. Par contre, il n'y pas d'ordre imposé entre B et C.

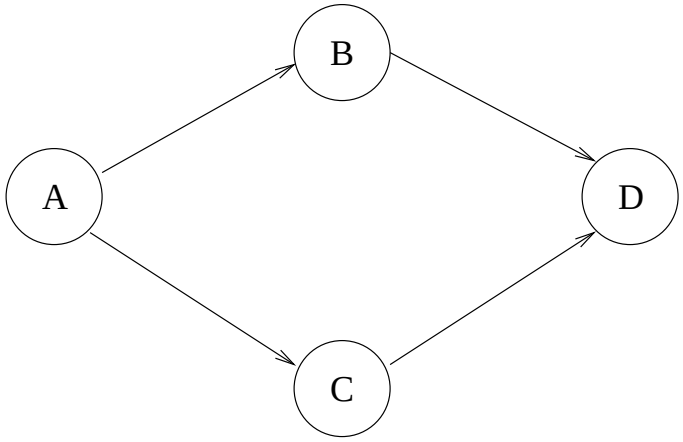

FIG. 1.4 – *Graphe de précédences*

Les problèmes d'ordonnancement des systèmes de tâches avec contraintes de précédence sont parfois soumis à certains critères d'optimalité. Dans ce cas, un ordonnancement ne doit pas seulement satisfaire les contraintes de précédences des tâches, mais il doit optimiser la solution par rapport à un critère donné. Ces critères appartiennent aux classes suivantes :

- critères de minmax : dans ce type de problème, il faut trouver un ordonnancement qui optimise (minimise ou maximise) une fonction monotone des dates de début de tâches;
- critères concernant le retard maximum : dans ce type de problème, les tâches possèdent des échéances et il faut trouver un ordonnancement qui minimise le retard maximum qui peut être égal au plus grand retard de toutes les tâches.

Un premier problème avec critère de type minmax est étudié par Lawler en 1973 [29]. Pour chaque tâche A, on définit une fonction de coût $f_A(s_A)$ qui varie par rapport à la date de fin de la tâche. Le critère d'optimalité est constitué par la fonction $f_{max} = \max_{j=1}^{n} f_j(C_j)$. L'algorithme donné, ordonnance les tâches en commençant par celles sans prédécesseur et en avançant vers les tâches sans prédécesseur. L'article prouve que l'algorithme construit

un ordonnancement qui est solution pour le problème. Il faut remarquer que l'algorithme ne prend pas en compte les durées d'exécution de tâches pendant l'ordonnancement. Après avoir construit un ordonnancement, les auteurs utilisent les durées d'exécution pour vérifier s'il satisfait les contraintes. Si l'ordonnancement obtenu ne satisfait pas les contraintes, alors il n'y a pas d'ordonnancement qui le fait.

Un cas particulier de ce problème est le cas des tâches avec contraintes de précédences et échéances. Pour cela, l'auteur formule le problème comme un problème de minmax. Il redéfinit les échéances des tâches par rapport aux précédences et aux anciennes échéances.

Au problème précédent on ajoute des dates de réveil. Le problème obtenu ainsi est prouvé NP-difficile [30]. Il suffit de considérer les durées d'exécution égales à une unité de temps et le problème devient facile et les dates de réveil sont redéfinies par rapport aux précédences. Aussi pour le problème d'ordonnancement préemptif dans le même cas, on obtient un problème facile résolu par un algorithme de complexité $\mathcal{O}(n^2)$, où n est le nombre de tâches.

Minimiser le retard maximum pour des systèmes de tâches avec contraintes de précédence, avec échéances et avec dates de réveil est connu comme étant NP-difficile. Il y a trois cas spéciaux de problèmes qui sont polynômiaux :

- les dates de réveil sont égales pour toutes les tâches. Après avoir redéfini les échéances par rapport aux précédences, les tâches sont ordonnancées dans l'ordre croissant des échéances. L'algorithme est une conséquence de l'algorithme de Lawler;
- les échéances sont égales pour toutes les tâches. Après avoir redéfini les dates de réveil par rapport aux précédences, les tâches sont ordonnancées dans l'ordre croissant des dates de réveil;
- les durées d'exécution de tâches sont toutes égales à 1. Après avoir redéfini les échéances par rapport aux précédences, à tout moment on ordonnance la tâche avec la plus petite échéance [31]. Les trois algorithmes ont une complexité de $\mathcal{O}(nlogn)$.

La plupart des résultats pour les systèmes de tâches avec contraintes de précédence sont obtenus pour des cas particuliers de précédences. Des algorithmes polynômiaux sont donnés pour le cas des précédences définies par des "in-trees" (chaque tâche a au plus un successeur), par des "out-trees" (chaque tâche a au plus un prédécesseur) ou par des graphes séries-parallèle. Un graphe séries-parallèle est un graphe dont la fermeture transitive ne contient pas un Z-graphe (voir figure 1.5) [32]. On remarque qu'un "intree" et un "outtree" sont des cas particuliers de graphes séries-parallèle.

Avant de présenter les résultats concernant les graphes séries-parallèle, on définit la notion de fonction qui permet un changement d'arguments. Une fonction f qui permet un changement d'arguments est une fonction si pour $f(\alpha) \leq f(\beta)$ on a : $f(a \ldots \alpha\beta \ldots b) \leq f(a \ldots \beta\alpha \ldots b)$.

Dans le cas où les précédences sont définies par un graphe séries-parallèle il y a un résultat général qui prouve l'existence de solutions pour des problèmes avec des critères d'optimalité donnés par des fonctions qui permettent des changements d'arguments, résultat donné par

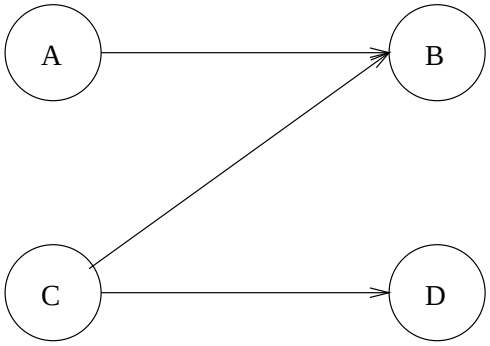

FIG. 1.5 – *Z-graphe*

le théorème suivant [33] :

Théorème 6 *Le problème d'ordonnancement d'un système de tâches avec contraintes de précédence définies par un graphe séries-parallèle et avec un critère d'optimalité donné par une fonction qui permet un changement d'arguments a toujours une solution.*

La solution à ce type de problème est en général trouvée en utilisant un algorithme qui ordonnance d'abord soit les tâches sans successeur, soit les tâches sans prédécesseur.

Un problème d'ordonnancement classique peut ne pas posséder un critère d'optimalité et dans ce cas il faut trouver un ordonnancement qui satisfait les contraintes des tâches. C'est le cas d'un autre problème important qui utilise les précédences pour modéliser les contraintes des tâches : le problème central d'ordonnancement. Dans ce problème, les tâches ont des contraintes de type potentiel : $s_A - s_B \geq a_{AB}$, où a_{AB} est un nombre réel donné. Une solution est constituée par un ordonnancement de durée minimale qui satisfait les contraintes de type potentiel. Ces contraintes permettent de modéliser les dates de réveil et les échéances des tâches d'un système.

Une modélisation du problème est faite à l'aide des graphes potentiel-tâches $\mathcal{G} = (\mathcal{V},\mathcal{E})$ dans [34]. L'ensemble $\mathcal{V}$ est égale à l'ensemble de tâches plus une tâche fictive de début A_0 et une tâche fictive fin A_{n+1}, où n est le nombre de tâches A_i de système. Les deux tâches fictives ont une durée nulle d'exécution. L'ensemble E contient les arcs (A_0,A_i) valués par 0, les arcs (A_i,A_j) correspondant aux contraintes de type potentiel valués par $a_{A_iA_j}$ et les arcs (A_i,A_n+1) valués par C_{A_i}.

Le problème ainsi obtenu est résolu à l'aide d'une méthode de chemin critique appelée la méthode potentiels-tâches qui consiste à calculer des dates au plus tôt et au plus tard. Cela permet ensuite de déterminer des marges pour chaque tâche, c'est à dire de déterminer le délai dont on peut retarder une tâche dans l'ordonnancement sans qu'elle ne rende l'ordonnancement inacceptable comme solution pour notre problème. Cette méthode est appelé « de chemin critique ».

Blum et autres [35] étudient le problème de p sommets liés par des arcs avec une matrice qui définit les distances entre les sommets. Ils cherchent un circuit qui minimise la somme de

toutes les latences de sommets, où la latence d'un sommet i est la distance parcourue avant qu'on visite i.

Xu et Parnas [36] étudient le problème des tâches non-périodiques, mais ils définissent des relations d'exclusion entre les tâches qui peuvent être ensuite utilisées pour définir des contraintes de périodicité. L'algorithme proposé utilise une technique de retour-arrière qui permet d'améliorer une solution intermédiaire et l'algorithme est prouvé optimal.

On remarque que tous les résultats présentés ne font pas d'étude d'ordonnançabilité, les auteurs préfèrent plutôt prouver l'optimalité de leurs algorithmes. Sur l'ordonnancement obtenu à l'aide de ces algorithmes, on vérifie si les contraintes sont satisfaites.

1.2.3 Contraintes de précédences et de périodicités

Ce chapitre traite le cas des systèmes de tâches avec contraintes de précédences et périodicités, toujours dans le cas monoprocesseur. Ces résultats viennent de l'ordonnancement temps réel, mais aussi de l'ordonnancement classique.

Les résultats sur les systèmes temps réel avec précédences et périodicités sont principalement obtenus pour des cas particuliers de précédences et de périodes. Les quelques modèles proposés ne permettent pas d'imposer des contraintes de précédence entre deux tâches périodiques quelconques. En conséquence, on ne peut pas étudier les liens qui peuvent exister entre précédence et périodicité.

En utilisant le modèle donné par Liu et Layland, Harbour et autres [37] traitent le problème d'ordonnançabilité d'un système de tâches, dont les tâches sont décomposées en sous-tâches soumises à des contraintes de précédence. Les auteurs considèrent le cas des ordonnancements à priorité variable. Mais l'article utilise un cas particulier des contraintes de précédence : les chaînes. Donc les contraintes de précédence imposent un ordre total sur l'ensemble des sous-tâches. Cela signifie qu'il n'y a aucun choix à faire lors de l'ordonnancement entre les sous-tâches de la même tâche. Les sous-tâches possèdent la même contrainte de périodicité que la tâche à laquelle elles appartiennent et elles ont des échéances cohérentes par rapport à l'échéance de la tâche à laquelle elles appartiennent. L'étude d'ordonnançabilité est faite tâche par tâche. Le même type d'étude d'ordonnançabilité est faite pour le même problème sans la possibilité de faire varier des priorités des tâches [38].

Toujours en utilisant le modèle donné par Liu et Layland, Chetto et autres [39] traitent le problème d'ordonnancement dynamique des systèmes avec tâches périodiques et sporadiques. Les tâches sporadiques ont des contraintes de précédence et l'étude d'ordonnançabilité concerne ce type de tâches. En revanche l'étude est faite sur des ordonnancements obtenus en utilisant l'algorithme EDF.

Le problème d'ordonnançabilité consiste à décider en ligne si un ensemble de tâches sporadiques dépendantes qui doit être ordonnancé peut être accepté ou non. Un ensemble est accepté si toutes les tâches de l'ensemble peuvent être ordonnancées avant leurs échéances, tout en satisfaisant les contraintes des tâches périodiques et des autres tâches sporadiques déjà acceptées. Les auteurs présentent un algorithme polynômial qui permet de prendre cette décision.

Spuri et autres [40] définissent des contraintes de précédence pour un système de tâches donné à l'aide du modèle de Liu et Layland. Ils imposent une condition importante sur les dates de réveil et sur les échéances : elles doivent satisfaire des conditions de cohérence par rapport à l'ordre partiel. Cela revient à redéfinir les dates de réveil et les échéances des tâches par rapport aux contraintes de précédence des tâches. Mais cette redéfinition impose des contraintes très fortes sur les contraintes de périodicité des tâches.

Un modèle plus général qui prend en compte les répétitions des tâches et des contraintes de précédence, sans que les tâches soient périodiques, a été proposé par Lee [41]. Ce modèle qui s'appelle SDF (Synchronous Data Flow) permet d'exprimer des rythmes de production et de consommation de données entre les tâches.

Baruah et autres [42] étudient les propriétés temps réel du modèle du SDF. Ils prouvent que le problème de décision est NP-complet si les instances sont ordonnançables sur un seul processeur et ils donnent des cas particuliers pour lesquels le problème devient facile.

Sur le même modèle, Goddard [43] définit des contraintes appelées latences qui bornent le temps écoulé entre une entrée et la sortie obtenue en utilisant cette entrée. Ces latences sont donc imposées seulement sur les entrées et les sorties et elles correspondent aux temps de réponse.

Korst traite dans sa thèse [44] l'ordonnancement périodique strict avec précédences. Dans ce cas il formule les précédences comme des contraintes linéaires et il prouve que l'ordonnançabilité d'un tel système est de complexité $O(nq)$ où n est le nombre des tâches et q est le nombre de précédences. Un corollaire de ce résultat permet de donner une condition nécessaire et suffisante d'ordonnançabilité.

Sun et Liu [45] présentent trois algorithmes d'ordonnancement pour un système avec contraintes de périodicités et de précédences, mais les précédences sont des chaînes, donc il y a toujours un ordre total entre les tâches qui ont des contraintes de précédence.

Baccelli et autres [46] utilisent l'algébre (max, plus) pour vérifier l'ordonnançabilité des systèmes avec contraintes de périodicités et de précédences modélisées à l'aide de reseaux de Pétri. Les tâches possèdent des priorités fixées à l'avance et la méthode permet de vérifier l'exactitude de ce choix.

Un résultat différent de ceux qu'on a présentés jusqu'à maintenant est donné par van Beek et Wilken [47]. Les auteurs considèrent le problème d'un graphe dont les arcs sont répétitifs, c'est à dire s'il y a une précédence de A vers B valuée avec la valeur n alors A doit se faire n fois afin que B devienne disponible. Les auteurs définissent des contraintes de latence : pour A et B, si l est une contrainte de latence, alors B doit être ordonnancée au moins après l unités de temps après que A soit finie. L'algorithme donné pour résoudre le problème utilise la programmation par contraintes.

Serafini et Ukovich proposent un modèle de précédences généralisées, qui ressemble au modèle proposé par van Beek. Dans ce cas, les tâches ne sont pas répétées afin de rendre leurs successeurs disponibles, mais un successeur doit attendre un temps depuis la fin de son prédécesseur afin qu'il soit disponible. Un autre modèle proche de celui proposé par van Beek est proposé par Munier dans [48] qui est une extension du problème central répétitif [34]. Les tâches sont répétées et les différences de répétition entre deux tâches liées par une

contrainte de précédence sont données par une fonction linéaire. L'auteur prouve que le plus grand chemin dans un graphe valué associé au graphe d'origine donne les valeurs maximales du nombre de répétitions des tâches.

On remarque le peu de résultats existant pour le cas de systèmes avec contraintes de périodicités et de précédences et les limites des modèles existant qui ne permettent pas d'avoir des contraintes de périodicité différentes entre deux opérations liées par une contrainte de précédence.

1.3 Résultats existant dans le cas multiprocesseur

Effectuer un ordonnancement temps réel multiprocesseur consiste à déterminer pour un ensemble d'opérations à quelle date chacune d'elles doit être exécutée et quelle ressource (processeur) est utilisée par chaque opération. Étant donné que tout problème d'ordonnancement de tâches sur n processeurs est prouvé NP-difficile [49], les résultats existant dans le cas multiprocesseur sont des heuristiques ou des algorithmes pseudo-polynômiaux. Puisque en général les systèmes temps réel utilisent des valeurs bornées [50], on peut trouver dans la littérature des algorithmes pseudo-polynômiaux.

Dans le cas multiprocesseur on traite non seulement un problème d'ordonnancement, mais aussi de distribution. On ne doit pas seulement imposer un ordre total sur l'ensemble des tâches, mais aussi décider sur quel processeur on ordonnance quelle tâche. La durée d'exécution d'une tâche peut être différente d'un processeur à un autre.

Il peut y avoir deux types d'approche :

- résoudre le problème d'ordonnancement d'abord et puis celui de distribution ou vice-versa. Donc l'heuristique a deux niveaux : un qui ordonnance les tâches et un deuxième qui distribue les tâches sur les processeurs [51], [52], [53], [54];
- résoudre les deux problèmes dans le même temps. Donc l'heuristique ordonnance et distribue les tâches en même temps [55], [56], [57].

Oh et Baker étudient l'ordonnançabilité d'un système en utilisant l'algorithme "rate monotonic" modifié pour le cas de n processeurs [58]. Leur méthode assure une bonne utilisation de chaque processeur. Ils donnent une limite supérieure et une limite inférieure pour le taux total d'utilisation de n processeurs. Ils prouvent aussi que la limite inférieure est utile dans la pratique. Cela est dû au fait qu'elle peut être obtenue en utilisant une méthode de type "first fit" (première tâche qui convient est ordonnancée).

Un autre résultat pour les systèmes avec tâches périodiques est donné par Hong et Leung [59]. Ils prouvent qu'il n'y a pas d'algorithme appliqué en-ligne qui est optimal pour le cas de processeurs hétérogènes. Ils prouvent aussi, que l'algorithme EDF n'est pas optimal dans le cas multiprocesseur, même appliqué hors-ligne [60]. Mais Goossens et autres [61] prouvent que EDF reste un bon algorithme dans le sens où ils donnent des conditions d'ordonnançabilité qui permettent de déterminer si un système de tâches satisfait ces contraintes en utilisant l'algorithme EDF.

Tindell et Clark [62] proposent une nouvelle technique appelée analyse holistique pour des systèmes de tâches qui ont des échéances arbitraires (il n'y a aucune relation entre l'échéance d'une tâche et sa période). Ils proposent une méthode qui permet de borner les retards dus aux communications et aux ressources partagées par les tâches. Ainsi, ils déterminent le pire cas pour chaque tâche et ils vérifient si les temps de réponse satisfont les contraintes temps réel du système. L'article étudie aussi des contraintes de bout en bout des messages c'est à dire le temps écoulé entre le départ d'un message depuis un processeur-source et son arrivé au processeur-destinataire. Toujours pour des contraintes de bout en bout, Richard et autres utilisent l'analyse holistique [63].

Il y a d'autres critères d'optimalité qui intéressent le monde temps réel comme par : exemple optimiser le nombre de processeurs [64], minimiser le temps de réponse des tâches [65], [66], [67] ou le temps de communication [68].

Une autre technique est d'énumérer en utilisant les solutions possibles [69], [70], [71], [65]. Pour réduire la taille des arbres, Richard et autres utilisent l'analyse holistique [56].

Korst traite dans sa thèse [44] l'ordonnancement périodique strict avec précédences où les précédences sont dues aux données consommées par les successeurs. Dans ce cas s'il y a une précédence de A à B, B utilise les données produites par A et ces données doivent être mémorisées, donc il y a aussi un problème de mémoire. Il traite d'abord l'ordonnancement et ensuite la distribution. Il donne un résultat théorique qui prouve que pour chaque ordonnancement qui satisfait les contraintes il y a une distribution et une mémoire qui satisfont les contraintes de tâches.

Korst et autres [23] prouvent que le problème de décision pour qu'un système de tâches périodiques strictes soit ordonnançable en minimisant le nombre de processeurs est NP-complet au sens fort, mais ils montrent aussi que le problème est polynômial pour le cas particulier avec périodes et durées d'exécution divisibles entre elles.

Les mêmes auteurs prouvent dans un autre article [24] que le problème de décision pour qu'un système de tâches périodiques soit ordonnançable en minimisant le nombre de processeurs est NP-complet au sens fort.

Shukla et Agrawal [72] proposent un modèle pour l'exécution périodique de graphes flot de données qui permet de prendre en compte les communications. Les auteurs présentent une étude d'ordonnançabilité.

Ramamritham [73] présente un modèle prenant en compte des contraintes de périodicité de tâches et les tâches sont formées de sous-tâches qui ont des contraintes de précédence entre elles. L'algorithme proposé détermine, d'abord la distribution des sous-tâches sur les processeurs et l'ordonnancement des sous-tâches, ensuite il détermine l'ordonnancement des communications. Les simulations prouvent que les heuristiques sont très efficaces.

Martel [74] étudie le problème d'ordonnancement et de distribution de n tâches qui ont des dates de réveil, des échéances et des durées d'exécution sur m processeurs. L'auteur donne un algorithme de complexité $O(m + logn)(m^2n^3 + n^4)$ qui est prouvé optimal dans le sens où s'il y a un ordonnancement, l'algorithme le trouvera. L'algorithme est ensuite utilisé avec des heuristiques de recherche pour trouver un ordonnancement qui minimise le retard maximal de toutes les tâches.

Aggarwal et Chandra étudient un modèle d'architecture avec n processeurs, chaque processeur ayant à sa disposition une mémoire de taille infinie [75]. Le coût de lecture/écriture depuis/dans la mémoire est pris est compte et il est appelé latence. Les auteurs donnent des algorithmes optimaux pour des problèmes particuliers comme la multiplication de matrices, etc.

Dans le cas non-préemptif d'un système de tâches périodiques, Baruah donne [76] une condition suffisante d'ordonnançabilité si le système est ordonnancé en utilisant l'algorithme EDF.

Tindell et autres étudient le problème de distribution de tâches périodiques sur n processeurs [77], en utilisant une technique de recuit simulé. L'algorithme proposé décrit les propriétés d'une solution sans donner le moyen de la trouver.

1.4 Motivations de la thèse

Dans la suite nous préférons la notion générale d'opération à celle de tâche car elle correspond au niveau de la spécification, auquel nous nous intéressons, plutôt qu'au niveau de celui de l'implantation.

On ne fait ainsi aucune hypothèse implicite sur le fait qu'une opération sera implantée par la suite sous la forme d'une tâche gérée par un système d'exploitation temps réel, car cela n'est pas obligatoire. En effet, si on a construit un ordonnancement hors ligne il pourra être exécuté sous la forme d'une séquence d'instructions sans qu'il ne soit besoin d'avoir recours à l'ordonnanceur d'un système d'exploitation temps réel. Pour être cohérent avec le terme d'opération nous utiliserons le terme "opérateur" plutôt que "ressource", "machine", "processeur". Afin de bien faire la différence entre le niveau spécification et implantation, Korst utilise lui aussi le mot "opération" dans l'article [23].

Nous allons successivement passer en revue ce qui nous a amené à étudier les approches hors ligne, sans préemption, pour des systèmes d'opérations soumises à des contraintes de précédences, de périodicités strictes et de latences. Le pluriel des mots précédences, périodicités strictes et latences a son importance car il veut dire que l'on peut avoir plusieurs contraintes de chacun de ces types. La motivation des choix s'appuie sur les résultats présentés dans les deux chapitres précédents.

Les applications temps réel qui nous intéressent se trouvent dans les domaines de l'automobile, du rail, de l'avionique, de la robotique mobile, des télécommunications, etc. Ce sont des applications temps réel dur, c'est-à-dire pour lesquelles le non respect d'une contrainte peut avoir des conséquences catastrophiques. Par exemple le non respect d'une contrainte de latence sur la commande de vol d'un avion peut conduire à ne plus pouvoir contrôler son altitude. Ces applications reposent toutes sur des algorithmes de traitement du signal et d'image et des algorithmes de contrôle commande qui peuvent nécessiter une grande quantité de calculs. Cependant nous savons à l'avance le nombre exact d'opérations qu'il va falloir réaliser. Il n'apparaîtra pas de nouvelles opérations lors de l'exécution de l'application. Ces raisons nous ont fait choisir naturellement l'approche hors ligne qui est la mieux adaptée. Cette approche a en plus l'avantage d'introduire un surcoût dû à l'ordonnancement minimal

lors de l'exécution puisqu'elle consiste tout simplement en une mise en séquence des opérations, et de plus elle est complètement déterministe. On connaît exactement le coût de l'ordonnancement, ce qui est nécessaire pour les systèmes temps réel durs.

Il y a deux raisons pour lesquelles nous avons choisi d'étudier l'ordonnancement non préemptif. La première prolonge la raison qui nous a fait choisir l'approche hors ligne. En effet, l'incertitude du coût de la préemption qui peut amener à penser qu'un ordonnancement sera bon alors qu'il ne l'est pas car une erreur a été commise lors de son estimation, nous ont conduit dans un premier temps à nous placer dans le cas non préemptif. On minimise ainsi le nombre d'opérations à réaliser puisque l'utilisation de la préemption induit un coût supplémentaire de lecture/écriture pour sauvegarder le contexte au moment où est faite chaque préemption. Nous envisageons cependant, comme suite à cette thèse, d'étudier plus précisément le coût de la préemption afin de montrer à partir de quand (en fonction de son coût) elle apporte un réel plus. Il y a peu de résultats dans la littérature à ce sujet, le choix ou non de faire de la préemption est fait en général a priori sans réelle justification, et il y a peu d'études sur l'influence du coût de la préemption sur l'ordonnancement [1]. La seconde raison réside dans le fait qu'une étude complète (avec précédences, périodicité et latences) dans le cas non préemptif nous a semblé être un "bon tremplin" pour obtenir de nouveaux résultats dans le cas préemptif.

En ce qui concerne la prise en compte de contraintes de précédence nous n'avons pas eu de choix à faire. Les algorithmes de traitement du signal et d'image et les algorithmes de contrôle commande sont habituellement spécifiés par les automaticiens à l'aide de "schéma bloc" comportant des boîtes représentant les fonctions (opérations) à réaliser et des flèches représentant des transferts de données entre ces boîtes, imposant un ordre d'exécution entre les fonctions afin d'assurer que le résultat produit à l'instant t sera consommé à l'instant t. Cet aspect est important car il est à la base de la notion d'état permettant de définir le comportement d'un système relativement à son comportement passé et à un état initial. On ne doit donc pas confondre l'instant t avec l'instant $t-1$ et on doit être capable de savoir l'état du système à l'instant initial. Nous travaillons donc naturellement avec un ensemble d'opérations sur lequel il existe une relation d'ordre partielle sur leur exécution qui définit les contraintes de précédence.

Les systèmes temps réel sont avant tout des systèmes réactifs [78], c'est à dire qu'ils doivent impérativement réagir à chaque changement d'état de leur environnement au travers de stimuli reçus via des opérations d'entrée réalisant la fonction de capteur-convertisseur-analogique-numérique échantillonnés. Le résultat de la réaction est produit vers l'environnement via des opérations de sortie réalisant la fonction d'actionneur-convertisseur-numérique-analogique, eux aussi échantillonnés. Cela conduit donc à exécuter infiniment la séquence suivante : réception de données via les capteurs, opérations sur ces données, émission des résultats obtenus via les actionneurs. Cette répétition infinie de chaque opération du système temps réel auquel on associe une durée de temps physique, la période, entre deux répétition, correspond à la notion de périodicité. Il peut bien sûr y avoir plusieurs capteurs possédant des périodes différentes. Comme cela est dit dans [4] la contrainte de périodicité la plus répandue est celle qui est définie par rapport à la date de réveil de la tâche (opération). Comme nos

applications temps réel reposent sur des algorithmes de traitement du signal et d'image et des algorithmes de contrôle commande dont les opérations sont toujours disponibles, la notion de date de réveil n'est pas utile. En effet, celle-ci est nécessaire quand on a en tête d'utiliser un système d'exploitation temps réel dont on doit modéliser le comportement. On sait que la tâche est réveillée à un instant fixe donné par l'horloge temps réel, mais on ne sait pas combien de temps après elle sera disponible. Dans notre cas on parle alors de période stricte comme cela est fait dans l'article [23]. A partir de maintenant, on utilisera toujours la notion de périodicité stricte même si pour simplifier le discours on utilisera le mot "périodicité".

La différence importante entre le modèle classique de Liu et Layland et notre modèle concerne l'incertitude sur la disponibilité et l'exécution des tâches (respectivement des opérations) dans les deux modèles. Dans le modèle de Liu et Layland l'instant auquel une tâche devient disponible est toujours connu (date de réveil), et la date de début de la première répétition de la tâche n'a pas comme conséquence la connaissance des dates de début de toutes les répétitions de la tâche. Dans notre modèle l'instant auquel une opération devient disponible est inconnu uniquement pour la première répétition de l'opération, de même pour la date de début. Cela a comme conséquence la connaissance des dates de début de toutes les autres répétitions de l'opération et donc l'inutilité des autres dates de disponibilité. Cette différence importante est détaillée en comparant les deux modèles pour une opération quelconque A dans la figure 1.6. On marque en pointillé les dates incertaines et en ligne continue les autres dates. L'ordonnancement qui se trouve en haut correspond au modèle classique et celui qui se trouve en bas de la figure correspond à notre modèle.

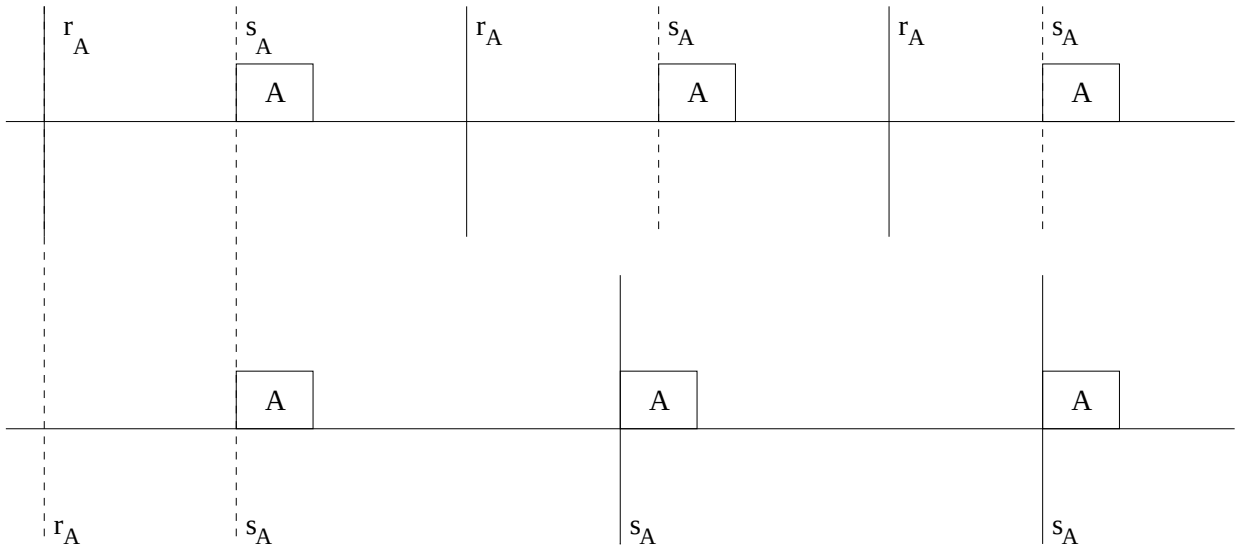

FIG. 1.6 – *Différence importante entre le modèle classique et notre modèle*

Le modèle SDF (Synchronous Data Flow) décrit dans [41] qui est une extension du modèle classique flot de données de Dennis [79] est plus proche de notre modèle que celui de Liu et Layland dans la mesure où les opérations ne possédent pas de date de réveil. Chaque

opération consomme une partie ou la totalité des données produites par d'autres opérations dès qu'elles sont disponibles, et réciproquement. On peut ainsi parler d'une répétition infinie d'une opération relativement à d'autres opérations, mais ce n'est pas une contrainte de périodicité car une durée n'est pas imposée entre deux répétitions d'une même opération.

La notion de périodicité apparaît aussi dans l'ordonnancement classique mais, contrairement à notre modèle, les opérations ont toutes la même période. C'est le cas du modèle utilisé pour décrire le problème central répétitif [34] ou du modèle proposé pour un ensemble d'opérations périodiques par Serafini [80].

Notre modèle permet à chaque opération d'avoir sa propre contrainte de périodicité et donc c'est un modèle multi-contraintes même de ce point de vue.

Dans la littérature sur l'ordonnancement temps réel, le mot "latence" a déjà été utilisé à plusieurs reprises. Par exemple par Hagmann [81] pour exprimer le temps nécessaire pour obtenir une donnée lors de la lecture depuis un disque dur, ou par Aggarwal dans [75] pour exprimer le temps nécessaire à la communication d'une donnée depuis une mémoire vers un processeur. Des définitions plus théoriques ont été données par Goddard [43] dans le cas de graphes flot de données ou par Van Beek [47] dans le cas de précédences répétitives. Les définitions théoriques existant se limitent à des cas particuliers, on ne peut pas les utiliser pour exprimer le temps écoulé entre deux opérations quelconques d'un système d'opérations. Évidemment si on veut que la latence exprime ce temps écoulé entre deux opérations, alors elle fait partie de la classe des contraintes relatives, et l'intérêt des contraintes relatives a très bien été justifié par Gerber [25]. Alors que les exemples donnés par Gerber ne traitent que des contraintes de bout en bout entre une opération d'entrée et une opération de sortie, voici un exemple de freinage pour une voiture équipée d'un régulateur de vitesse qui ne peut être résolu à l'aide d'une contrainte absolue et qui nécessite une contrainte de latence entre deux opérations qui ne sont pas toutes des entrées et des sorties. Dans la figure 1.7 l'opération DEF exprime le déclenchement du frein, l'opération CAL exprime les calculs faits pour déterminer la force de l'appui sur le frein, l'opération DAC exprime la désactivation du régulateur de vitesse, et l'opération DFR exprime le déclenchement du freinage sur les roues. Si on veut limiter le temps entre l'opération CAL et l'opération DFR une contrainte relative plus « forte » que la contrainte de bout en bout (début-fin) s'impose.

Notre définition de la latence généralise la notion de contrainte de bout en bout qui est imposée entre une entrée et une sortie du système temps réel. De plus les études existant traitant de ce type de contrainte [25] [26] [27] sont faites dans des cas particuliers: soit les précédences entre les opérations sont des chaînes, soit les périodes ne font pas partie des hypothèses du problème, mais elles sont calculées par l'algorithme d'ordonnancement.

Pour conclure, nous proposons un nouveau modèle pour les systèmes temps réel dont l'intérêt principal est sa généralité [82]. Il permet de prendre en compte des contraintes de précédence ainsi que plusieurs contraintes de périodicité et plusieurs contraintes de latence. Ces dernières contraintes sont nouvelles et correspondent à un réel besoin pour les utilisateurs. Nous réalisons les études d'ordonnancement et d'ordonnançabilité [83], [84] corresppondantes dans le cas monoprocesseur et dans le cas multiprocesseur.

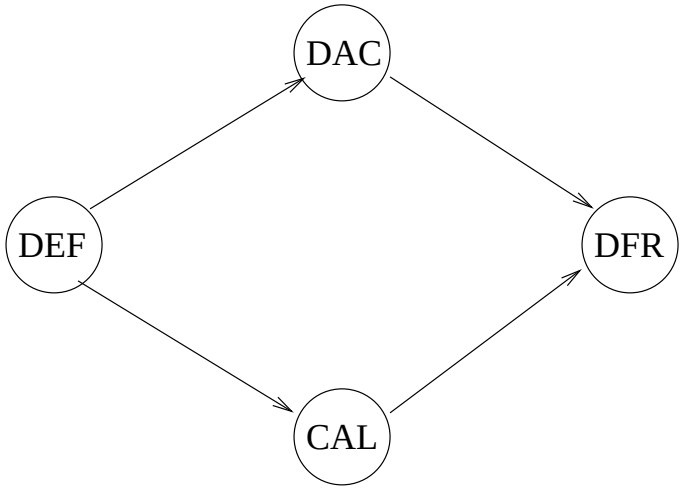

FIG. 1.7 – *Graphe spécifiant une application de freinage*

Chapitre 2

Ordonnancement et ordonnançabilité pour systèmes avec contraintes de précédences, de périodicités et de latences dans le cas monoprocesseur

Le deuxième chapitre de cette thèse est dédié aux résultats obtenus dans le cas monoprocesseur et il contient quatre sous-chapitres. Le premier sous-chapitre étudie les systèmes d'opérations avec contraintes de précédences et de périodicités, le deuxième sous-chapitre étudie les systèmes d'opérations avec contraintes de précédences et de latences, le troisième sous-chapitre étudie les systèmes d'opérations avec contraintes de précédences, de périodicités et de latences, enfin le quatrième sous-chapitre prouve que les systèmes d'opérations avec contraintes de périodicités, dates de réveil et échéances utilisés classiquement en ordonnancement temps réel, sont un cas particulier des systèmes d'opérations avec contraintes de précédences, de périodicités et de latences.

Les trois premiers sous-chapitres présentent le modèle utilisé, puis une condition d'ordonnançabilité et un algorithme d'ordonnancement, prouvé optimal, dans le sens où s'il y a un ordonnancement l'algorithme le trouvera. Cependant l'ordre de présentation de ces résultats est différent selon les sous-chapitres. En effet, le premier sous-chapitre présente d'abord l'algorithme d'ordonnancement puis la condition d'ordonnançabilité, car la condition d'ordonnançabilité s'appuie sur des propriétés particulières des ordonnancements obtenus avec l'algorithme d'ordonnancement proposé. En revanche, le deuxième sous-chapitre présente d'abord la condition d'ordonnançabilité puis l'algorithme d'ordonnancement, car l'algorithme d'ordonnancement utilise les conclusions de l'étude d'ordonnançabilité faite lors de la preuve de la condition d'ordonnançabilité. De même le troisième sous-chapitre présente d'abord la condition d'ordonnançabilité puis l'algorithme d'ordonnancement pour la même raison que précédemment.

Les quatre sous-chapitres utilisant les notions d'ordonnancement et d'ordonnançabilité qui ont été définies d'une manière informelle dans le chapitre 1.1, on donne maintenant des

définitions formelles de ces deux notions.

Définition 1 *pour un système d'opérations, un ordonnancement S est un ordre total sur l'ensemble des opérations qui associe à chaque opération A une date de début s_A qui exprime l'instant auquel l'opération sera exécutée. Cela signifie qu'un ordonnancement S est égal à l'ensemble $\{s_A \in \mathbb{N}, A \in \mathcal{V}\}$. On note par $\mathcal{S}$ l'ensemble de tous les ordonnancements possibles pour un système d'opérations.*

Définition 2 *un système d'opérations est appelé ordonnançable s'il y a au moins un ordonnancement qui satisfait toutes ses contraintes.*

Une autre définition qui est commune aux trois premiers sous-chapitres concerne la notion de chemin.

Définition 3 *[85] un chemin orienté de l'opération A_1 à l'opération A_n est la séquence alternée des sommets et arcs différents $A_1(A_1,A_2)A_2 \ldots A_{n-1}(A_{n-1},A_n)A_n$ où $A_i \in \mathcal{V}$ et $(A_i,A_{i+1}) \in \mathcal{E}$, $\forall i \in \{1,2,\ldots,n\}$.*

Puisqu'on utilise seulement des graphes orientés (voir chapitre 1.2.2), pour simplifier on utilisera dorénavant le terme « chemin » au lieu de « chemin orienté ».

On note par $\mathcal{P}$ l'ensemble de tous les chemins du graphe $\mathcal{G}$ et on dit que $P(A,B) \in \mathcal{P}$ s'il y a au moins un chemin de A à B. Si $\nexists P(A,B) \in \mathcal{P}$, alors il n'y a pas de chemin de A à B. S'il y a au moins un chemin de A à B, alors on note par $\mathcal{M}(A,B)$ l'ensemble des opérations appartenant à tous les chemins de A à B, $\mathcal{M}(A,B) = \{C \in \mathcal{V}$ avec $P(A,C) \in \mathcal{P}$ et $P(C,B) \in \mathcal{P}\} \bigcup \{A,B\}$. Pour illustrer ces notations on donne l'exemple suivant.

Exemple 1 *On considère le graphe G donné dans la figure 2.1. L'ensemble $\mathcal{M}(A,G) = \{A,B,D,C,G\}$ n'est pas vide puisqu'il y a au moins un chemin de A à G c'est-à-dire $P(A,G) \in \mathcal{P}$. Par exemple, de A à G il y a 4 chemins : $A(A,D)D(D,G)G$, $A(A,B)B(B,D)D(D,G)G$, $A(A,D)D(D,C)C(C,G)G$ et $A(A,B)B(B,D)D(D,C)C(C,G)G$. En revanche, par exemple, puisqu'il n'y aucun chemin de C à E et l'ensemble $\mathcal{M}(C,E)$ est vide.*

2.1 Contraintes de précédences et de périodicités

Ce chapitre s'intéresse aux systèmes temps réel avec contraintes de précédences et de périodicités. Même si séparément les deux domaines sont riches en résultats, les études où les deux aspects sont traités ensembles ne concernent que des cas particuliers. Le premier sous-chapitre présente un modèle original pour les systèmes temps réel avec contraintes de précédences et de périodicités. Ce modèle permet d'imposer des contraintes de précédence entre deux opérations quelconques et une contrainte de périodicité sur une opération quelconque. Il met en évidence la cohérence entre ces deux contraintes et la pertinence de la périodicité par rapport aux précédences et vice-versa. Le sous-deuxième chapitre présente un algorithme optimal (s'il y a un ordonnancement, l'algorithme le trouvera) et le sous-dernier chapitre fait une étude d'ordonnançabilité.

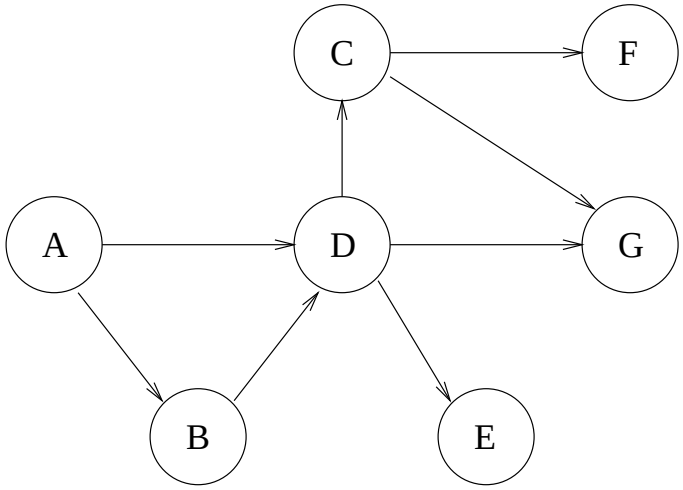

FIG. 2.1 – *Ensemble des opérations appartenant à tous les chemins allant d'une opération A à une opération G*

2.1.1 Modèle

Notre modèle permet de spécifier des systèmes temps réel qui, on le rappelle, sont avant tout des systèmes réactifs interagissant de manière permanente avec l'environnement au travers de capteurs et d'actionneurs. Les capteurs correspondent aux opérations du système qui n'ont aucun prédécesseur et les actionneurs correspondent aux opérations du système qui n'ont aucun successeur. Ces opérations s'appellent opérations d'entrée, respectivement, opérations de sortie. Une opération de sortie est obtenue à partir d'une ou plusieurs opérations d'entrée après plusieurs opérations intermédiaires qui sont liées à des opérations d'entrée et de sortie par des contraintes de précédence. Ces contraintes de précédence sont définies dans notre modèle par un graphe orienté acyclique $\mathcal{G} = (\mathcal{V},\mathcal{E})$, où $\mathcal{V}$ est l'ensemble d'opérations et $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ l'ensemble d'arcs qui représentent les contraintes de précédence. Chaque opération peut avoir une contrainte de précédence relativement à une autre opération et cette contrainte impose un ordre d'exécution entre les deux opérations, le successeur ne peut être ordonnancé que si son prédécesseur l'a été. Evidemment, si une opération a plusieurs prédécesseurs, elle doit être ordonnancée après tous ses prédécesseurs.

Deux opérations sur lesquelles aucune contrainte de précédence n'est imposée possèdent un « parallélisme potentiel » [55]. Cette expression vient du fait que lorsqu'on est dans le cas multiprocesseur les deux opérations peuvent être potentiellement ordonnancées en parallèle (en même temps) sur deux processeurs différents. Dans le cas monoprocesseur traité dans ce chapitre, deux opérations qui n'ont aucune contrainte de précédence entre elles peuvent s'ordonnancer dans un ordre quelconque. Ainsi, la relation d'ordre définie par le graphe est une relation d'ordre partiel.

L'interaction permanente du système temps réel avec l'environnement implique une répétition infinie des opérations d'entrée et des opérations de sortie. Les répétitions de ces

opérations impliquent une répétition des autres opérations intermédiaires due aux précédences du graphe. Ainsi, il n'y a pas seulement les opérations d'entrée et les opérations de sortie qui sont répétées infiniment mais toutes les opérations du graphe $\mathcal{G}$. On a donc un graphe infini $\mathcal{G}_\infty = (\mathcal{V}_\infty,\mathcal{E}_\infty)$. On appellera par la suite le graphe $\mathcal{G}$ « motif de la répétition infinie ». On définit $\mathcal{V}_\infty$ à l'aide d'une bijection $Ind : \mathcal{V} \times \mathbb{N} \rightarrow \mathcal{V}_\infty$ où $Ind(A,i) = A_i$ avec A_i la i^{me} répétition de A correspondant à l'opération A appartenant à la i^{me} répétition du motif. Les opérations appartenant à des motifs différents peuvent avoir des contraintes de précédence appelés « arcs inter-motifs » et de même les arcs appartenant au même motif s'appellent « arcs intra-motif ». L'ensemble infini de ces arcs est $\mathcal{E}_\infty = \{(A_i,B_i),\forall i \in \mathbb{N} \text{ si } \exists(A,B) \in \mathcal{V}\} \bigcup \{(A_i,B_j) \text{ t.q. } i,j \in \mathbb{N},\ i < j,\ A_i,B_j \in \mathcal{V}_\infty\}$.

La répétition infinie du motif permet de définir une « contrainte de périodicité » pas seulement pour une opération d'entrée ou de sortie mais aussi pour une opération quelconque de $\mathcal{G}_\infty$. Cette contrainte de périodicité impose une relation d'ordre total entre les répétitions consécutives d'une opération et de plus ces répétitions doivent être ordonnancées à des intervalles de temps physique égaux dont la valeur numérique s'appelle « période ». Comme expliqué au chapitre 1.4, cette contrainte de périodicité est une contrainte de périodicité stricte définie au chapitre 1.2.1.

Définition 4 *une opération A a une contrainte de périodicité de période $T_A \in \mathbb{N}$ si $s_{A_{i+1}} - s_{A_i} = T_A,\forall i \geq 1$, où A_i,A_{i+1} sont les répétitions consécutives i et $i+1$ de l'opération A et $(A_i,A_{i+1}) \in \mathcal{E}_\infty$. On note par A_1 la première répétition de A.*

Sans perdre de généralité [86], on suppose que les contraintes de périodicité et les durées d'exécution des opérations sont des multiples d'un tic τ (le temps est discret). A partir de maintenant, toutes les valeurs de périodes et de durées d'exécution sont multipliées implicitement par τ.

Remarque 1 *Un système d'opérations avec contraintes de précédences et de périodicités est ordonnançable (définition 2) s'il y a au moins un ordonnancement qui satisfait toutes ses contraintes de précédences et de périodicités. Trouver un tel ordonnancement est le problème que nous cherchons à résoudre dans ce chapitre.*

Remarque 2 *Sans perdre de généralité, on suppose qu'il y a au moins une opération avec une contrainte de périodicité dans chaque composante connexe du graphe d'opérations. Dans le cas contraire la composante connexe qui ne contiendrait aucune opération avec une contrainte de périodicité n'aurait aucune contrainte à satisfaire.*

Exemple 2 *La notion de contrainte de périodicité est illustrée dans la figure 2.2, où on a les contraintes de périodicité $T_A = 10$ et $T_B = 30$ avec $C_A = C_B = C_C = 1$. On peut définir une contrainte de périodicité pour A puisque cette opération a une contrainte de précédence répétée infiniment entre ses répétitions consécutives, de même pour B. En revanche, on ne peut pas définir une contrainte de périodicité pour C. Un ordonnancement qui satisfait les contraintes de précédences et de périodicités est donné dans la figure 2.3.*

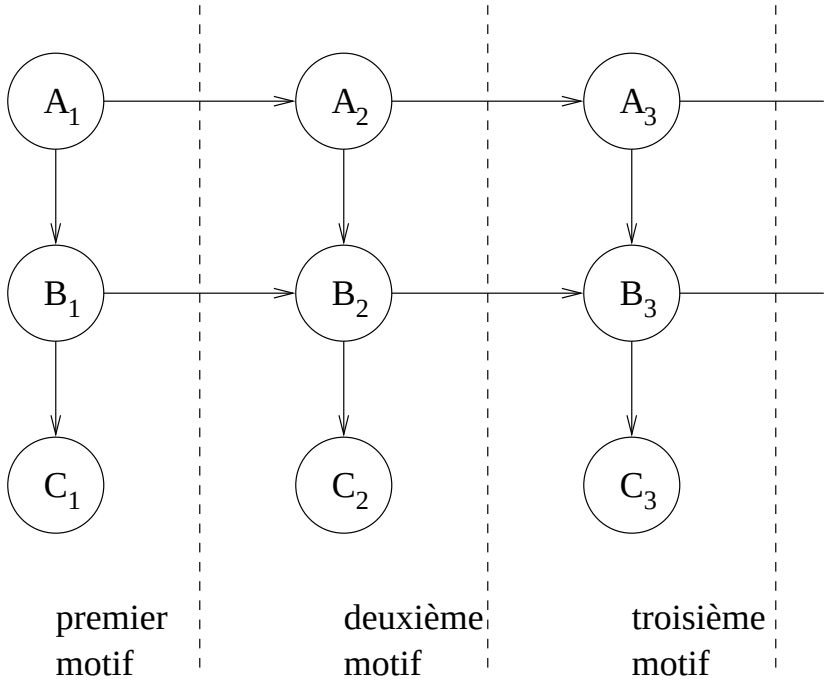

FIG. 2.2 – *Graphe contenant des opérations périodiques et des opérations non-périodiques*

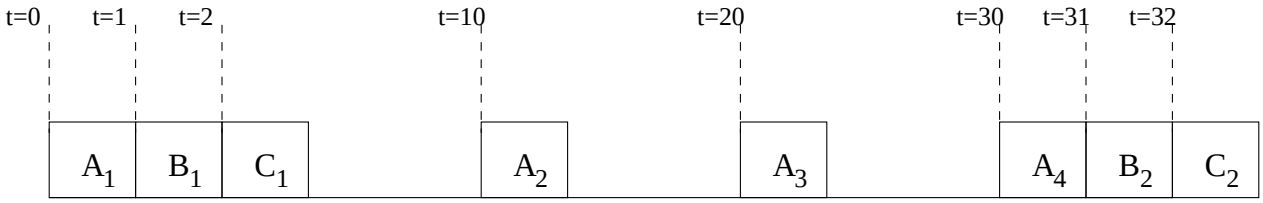

FIG. 2.3 – *Ordonnancement d'un système contenant des opérations périodiques et des opérations non-périodiques*

Les arcs intra-motif du graphe infini permettent d'exprimer des graphes comme celui de la figure 2.4 où il y a des contraintes de précédence entre des opérations différentes qui ont le même indice de répétition infinie. Pour imposer des contraintes de précédence entre des opérations différentes qui n'ont pas le même indice, il faudrait utiliser des arcs inter-motifs, mais ceux-ci ne permettent pas d'imposer des contraintes de précédence de A_i à B_j où $i > j$. En conséquence si on veut comme dans l'exemple de la figure 2.5 imposer une contrainte de A_2 à B_1, on ne peut utiliser que les arcs intra-motif, en répétant l'opération A d'une manière finie à l'intérieur du motif du graphe. On appellera par la suite les répétitions d'une opération à l'intérieur du motif du graphe « répétitions finies ».

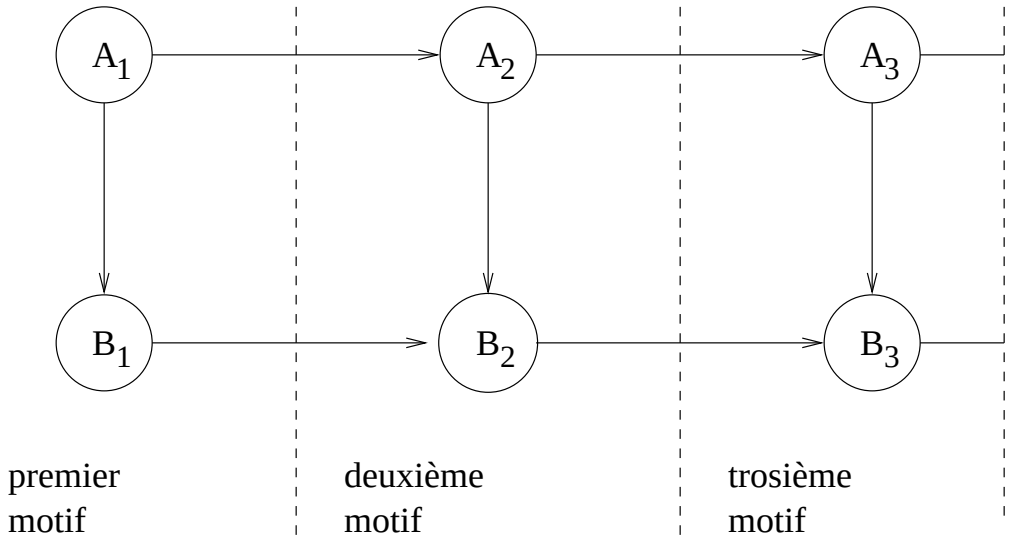

FIG. 2.4 – *Graphes d'opérations contenant seulement des opérations avec contraintes de précédence entre les répétitions de mêmes indices*

On considère pour les deux graphes des figures 2.4 et 2.5 les contraintes de périodicité $T_A = 10$ et $T_B = 30$ avec $C_A = C_B = 1$. L'ordonnancement de la figure 2.6 satisfait les contraintes du système défini par le graphe de la figure 2.4 et de même l'ordonnancement de la figure 2.7 satisfait les contraintes du système défini par le graphe de la figure 2.5. Il est important de noter que même si dans les deux ordonnancements (celui de la figure 2.6 et celui de la figure 2.7) on a un même motif $A - A - A - B$ qui se répète dans l'ordonnancement, ce motif s'installe à partir d'instants différents ; à l'instant $t = 10$ pour la figure 2.6 et à l'instant $t = 20$ pour la figure 2.7. On appellera par la suite ce motif « motif de l'ordonnancement ». La partie de l'ordonnancement qui s'étend du début de l'ordonnancement jusqu'à l'installation du motif s'appelle en général « régime transitoire » et la suite, qui se répète, « régime permanent ».

En conséquence, les répétitions finies permettent d'imposer des contraintes de précédence entre des opérations différentes qui n'ont pas le même indice. De cette manière, on peut spécifier des opérations qui doivent être ordonnancées plusieurs fois avant l'ordonnancement

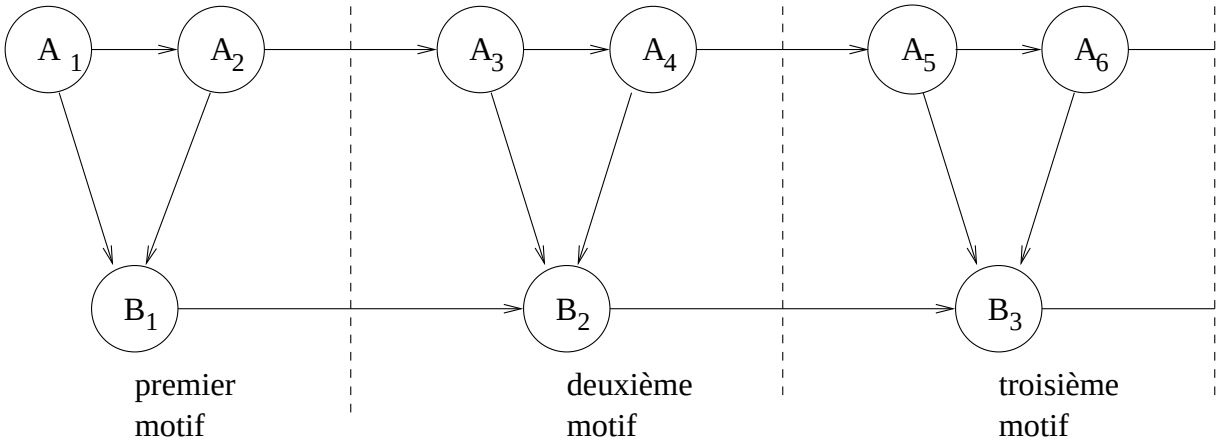

FIG. 2.5 – *Graphes d'opérations contenant des opérations avec contraintes de précédence entre les répétitions d'indices différents*

de leurs successeurs.

Remarque 3 *Il faut noter que le motif de l'ordonnancement et le motif de la répétition sont différents tout en étant évidemment liés. Le motif de l'ordonnancement correspond à un des ordres totaux obtenus à partir de l'ordre partiel auquel correspond le motif de la répétition.*

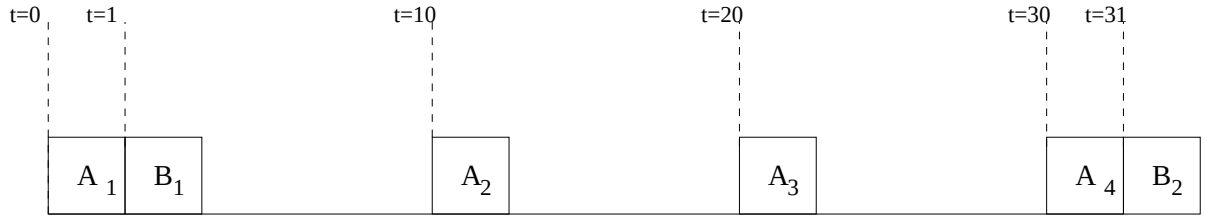

FIG. 2.6 – *Ordonnancement satisfaisant les contraintes de précédences et de périodicités du graphe de la figure 2.4*

La présence ou l'absence de précédences entre les répétitions différentes d'une même opération permettent d'avoir deux types de répétitions. Une répétition est appelée « répétition temporelle » quand les répétitions d'une même opération ont des contraintes de précédence entre elles, donc il y a un ordre total imposé entre les différentes répétitions. Une répétition est appellée « répétition spatiale » quand les répétitions d'une même opération sont sans contrainte de précédence entre elles, donc elles peuvent être ordonnancées dans un ordre quelconque. Il faut noter qu'on peut définir une contrainte de périodicité uniquement pour des opérations répétées temporellement.

Par exemple, dans la figure 2.8 l'opération A est répétée temporellement trois fois et l'opération C est répétée spatialement deux fois, A_1 est une opération d'entrée et C_1 et C_2

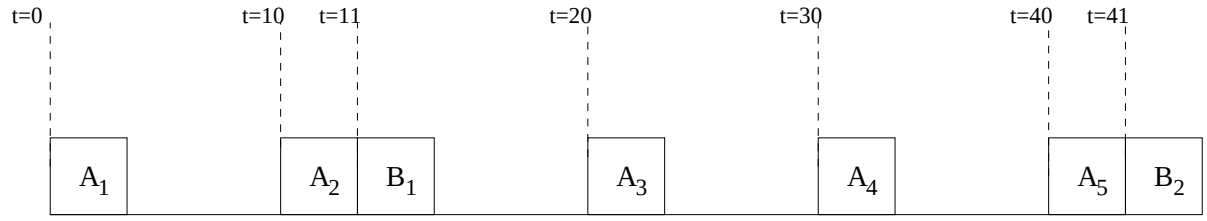

FIG. 2.7 – *Ordonnancement satisfaisant les contraintes de précédences et de périodicités du graphe de la figure 2.5*

sont des opérations de sortie. L'arc de A_3 du i^{me} motif à A_1 du $(i+1)^{me}$ motif est un arc inter-motif et l'arc de B à C_1 est un arc intra-motif. Il faut bien noter que la répétition infinie du motif induit une répétition infinie de toutes les opérations.

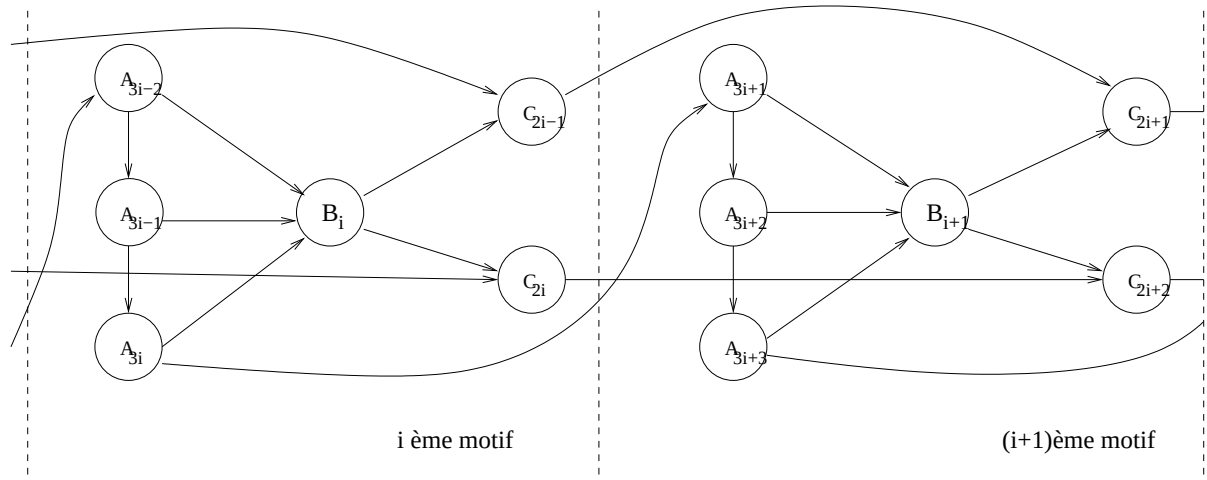

FIG. 2.8 – *Graphe non factorisé*

Afin de simplifier ce modèle de graphe complexe que l'on va appeler *graphe non factorisé*, on définit une contrainte de précédence *répétée finiment* quand les opérations répétées (spatialement ou temporellement) sont en relation de précédence et ces précédences correspondent à des arcs intra-motif. Cela permet au motif de contenir une seule répétition de chaque opération qui est répétée finiment et de décrire à l'aide d'une fonction *code* les arcs existant entre deux opérations dont au moins une est répétée finiment. Pour deux opérations A et B répétées temporellement ou spatialement, on définit $codeAB : \{1,2,..,n\} \times \{1,2,..,m\} \rightarrow \{0,1\}$ tel que :

$$codeAB(i,j) = \begin{cases} 1, & \text{s'il y a un arc de la } i^{me} \text{ répétition de } A \\ & \text{à la } j^{me} \text{ répétition de } B \\ 0, & \text{sinon} \end{cases}$$

où n est le nombre de répétitions de A et m le nombre de répétitions de B. Il est évident

que le nombre d'arcs possibles entre toutes les répétitions de A et de B est borné par nm (on a au plus m arcs pour chaque répétition de A).

Pour le cas particulier des précédences entre les différentes répétitions d'une opération A répétée temporellement n fois, on a la fonction $codeAA$: $\{1,2,..,n\} \times \{1,2,..,n\} \rightarrow \{0,1\}$ telle que :

$$codeAA(i,j) = \begin{cases} 1, & \text{pour } j = i+1 \text{ et } i < n \\ 0, & \text{sinon} \end{cases}$$

S'il n'y a aucun arc entre les répétitions consécutives de A, la fonction $codeAA$ n'est pas spécifiée. Quand la fonction $codeAA$ est spécifiée, le nombre d'arcs possibles entre les répétitions consécutives de A est $n-1$, sinon il est 0.

Deux opérations d'entrée ou deux opérations de sortie appartenant à deux motifs consécutifs peuvent être en relation de précédence appelée contrainte de précédence *répétée infiniment* et cette précédence est représentée par un arc ∞. Tous ces arcs sont des arcs inter-motif.

Il y a des opérations qui sont à la fois répétées finiment et infiniment. Pour une opération d'entrée (respectivement une opération de sortie) A qui est répétée temporellement infiniment et aussi répétée temporellement finiment n fois (à l'intérieur du motif), on a un arc ∞ dans le graphe factorisé correspondant à la contrainte infinie de précédence dans le graphe non factorisé. Si $codeAA$ a été défini pour A alors on a un arc entre la dernière répétition A_n et la première répétition A_1 appartenant au prochain motif. Si aucune fonction $codeAA$ n'a été définie alors on a un arc entre A_i et la répétition A_i appartenant au prochain motif pour toutes les répétitions A_i, $i \in \{1,2,\ldots,n\}$.

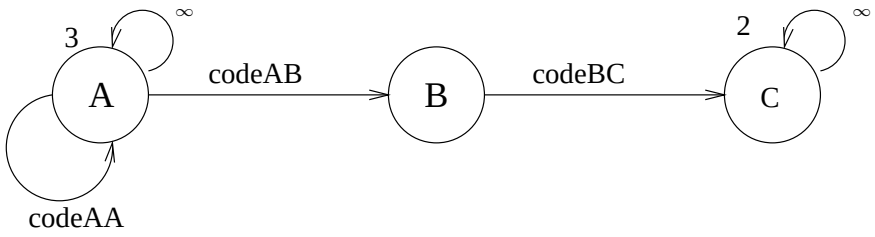

FIG. 2.9 – *Graphe factorisé*

Le graphe obtenu en utilisant les contraintes de précédence répétées finiment et les contraintes de précédence répétées infiniment s'appelle *factorisé*.

A partir du graphe non factorisé de la figure 2.8 on obtient le graphe factorisé de la figure 2.9 avec les fonctions *code* suivantes :

$codeAB$: $\{1,2,3\} \times \{1\} \rightarrow \{0,1\}$ avec $codeAB(i,1)=1$, $\forall i \in \{1,2,3\}$, $codeBC$: $\{1\} \times \{1,2\} \rightarrow \{0,1\}$ avec $codeBC(1,i)=1$ $\forall i \in \{1,2\}$ et la fonction $codeAA$ donnée par la définition générale d'une fonction *code* pour une opération temporellement répétée n fois (dans ce cas $n=3$). Les opérations A et C ont un arc ∞.

Remarque 4 *Dans le cas où on a dans le graphe factorisé deux opérations A et B avec le même nombre n de répétitions et la fonction $codeAB$:* $\{1,2,..,n\} \times \{1,2,..,n\} \rightarrow \{0,1\}$ *telle*

que

$$codeAB(i,j) = \begin{cases} 1, & si\ i = j \\ 0, & sinon \end{cases}$$

afin de simplifier le modèle, la fonction code ne sera pas spécifiée pour ce cas particulier de contrainte répétitive de précédence. Aussi, dans le cas où le nombre de répétitions d'une opération est égal à 1, on ne le mentionne pas dans le graphe factorisé.

Avant de passer à la présentation des résultats, on donne une première relation qui est la conséquence de la présence des précédences entre opérations périodiques.

Lemme 1 *Si deux opérations périodiques A, B ayant une contrainte de précédence $(A,B) \in \mathcal{E}$ appartiennent à un système défini par un graphe $\mathcal{G} = (\mathcal{V},\mathcal{E})$ alors*

$$T_A \leq T_B \tag{2.1}$$

Preuve En effet, la première répétition de A doit être ordonnancée avant la première répétition de B due à la contrainte de précédence. La contrainte de précédence répétée infiniment entre A et B implique que A doit être ordonnancée toujours avant B, ainsi on obtient l'équation (2.1). Le lemme est prouvé ⊡

De la même façon, si les opérations A et B sont répétées finiment, respectivement, n et m fois dans le motif répété infiniment et s'il y a au moins une contrainte de précédence d'une répétition de A à une répétition de B, alors

$$nT_A \leq mT_B \tag{2.2}$$

Cela permet de conclure que l'ordre partiel correspondant du point de vue ordonnancement à l'ordre croissant des contraintes de périodicité d'opérations d'un chemin dans un graphe, c'est-à-dire un ordonnancement satisfaisant les contraintes de précédence satisfait aussi les contraintes de périodicité.

L'exemple suivant illustre l'inégalité entre les périodes des opérations qui ont des précédences entre elles.

Exemple 3 *Soit un graphe $\mathcal{G} = (\mathcal{V},\mathcal{E})$ donné dans la figure 2.10 tel que la durée de chaque opération $C_A = 1, \forall A \in \mathcal{V}$. On présente deux exemples : un exemple avec les contraintes de périodicité qui satisfont l'inégalité 2.2 et un deuxième exemple avec les contraintes de périodicité qui ne satisfont pas l'inégalité 2.1.*

Si on considère que $T_A = 2$ et $T_B = 4$ alors $2T_A = T_B$ et il y a un ordonnancement qui satisfait les contraintes de périodicité données dans la figure 2.11. Si on considère que $T_A = 4$ et $T_B = 4$ alors $2T_A > T_B$ et on remarque dans la figure 2.12 qu'à l'instant $t = 5$ de l'ordonnancement l'opération B doit être ordonnancée avant A_1 qui est ordonnancée à $t = 8$ (les deux répétitions appartiennent au même motif). Donc la contrainte de périodicité ne permet pas de satisfaire les contraintes de précédence.

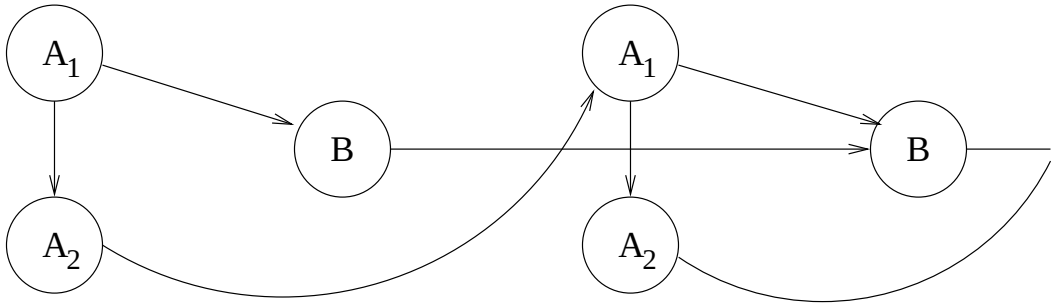

FIG. 2.10 – *Graphe contenant des opérations avec contraintes de précédence et ayant des périodes différentes*

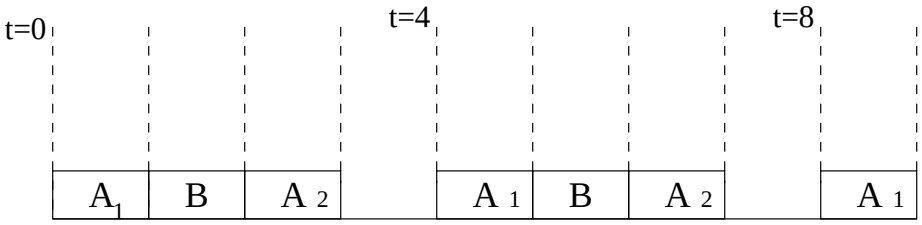

FIG. 2.11 – *Ordonnancement satisfaisant des contraintes de précédences et de périodicités*

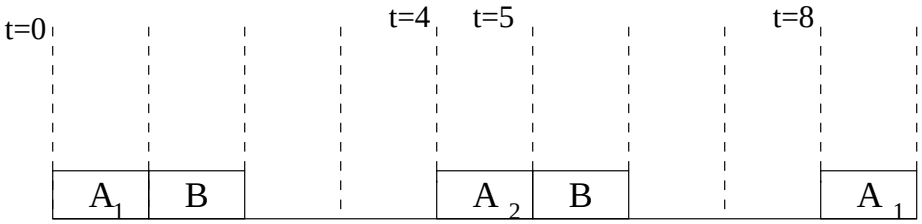

FIG. 2.12 – *Ordonnancement ne satisfaisant pas des contraintes de précédences et de périodicités*

On s'intéresse à la complexité de notre problème d'ordonnancement, d'un système d'opérations avec contraintes de précédences et de périodicités, car le problème de décision associé au même problème mais sans précédence a été prouvé NP-complet par Korst et autres [23]. On utilise donc son résultat afin de prouver que notre problème est aussi NP-complet. Ensuite on utilise son autre résultat qui prouve que dans le cas où les périodes et les durées d'exécution forment une séquence de valeurs divisibles le problème est polynômial, pour prouver que cette propriété reste valable pour notre problème aussi.

Le problème d'ordonnancement avec valeurs quelconques de périodes et de durées d'exécution, pour des systèmes d'opérations avec contraintes de périodicités et sans précédence est le suivant : on cherche un ordonnancement des opérations sur m processeurs identiques avec $m \leq k$ qui satisfait les contraintes de périodicités des opérations appartenant à $\mathcal{V}$, où $\mathcal{V}$ est un ensemble de n opérations périodiques $A_1, \ldots, A_n$ avec T_i la période et $C_i \leq T_i$ la durée d'exécution de l'opération A_i pour tout $1 \leq i \leq n$, et un entier k. Le problème de décision associé à ce problème est noté par PSP. Dans le même article, le théorème suivant prouve que le problème de décision PSP associé à ce problème d'ordonnancement est NP-complet au sens fort.

Théorème 7 *[23] PSP est NP-complet au sens fort pour $k = 1$.*

Il faut noter que pour $k = 1$ puisque $m \leq k$ on obtient $m = 1$ et on se trouve ainsi dans le cas monoprocesseur qui nous intéresse. Dans le même article, Korst prouve que le problème avec des valeurs particulières de périodes et de durées d'exécution est polynômial. Les auteurs notent par $PSPD$ le problème de décision PSP avec les périodes et les durées d'exécution formant une séquence de valeurs divisibles : $a_1, \ldots a_l$ avec $a_i / a_{i+1}, \forall i \in \{1, \ldots, l-1\}$. Ils donnent le théorème suivant :

Théorème 8 *[23] $PSPD$ est de complexité $\mathcal{O}(nl)$.*

Pour pouvoir utiliser ces résultats, on donne le théorème suivant qui prouve l'équivalence entre le problème sans précédence présenté par Korst pour $k = 1$ et notre problème. Pour cela on formule le problème de décision associé à notre problème d'ordonnancement noté par PP (précédences, périodicités) comme étant le suivant : y a-t-il un ordonnancement qui satisfait les contraintes de périodicités et de précédences où les contraintes de précédences sont définies par le graphe $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ pour lequel $\mathcal{V}$ est l'ensemble de n opérations périodiques $A_1, \ldots, A_n$ avec T_i la période pour tout $1 \leq i \leq n$?

Théorème 9 *La réponse est oui pour le problème PSP avec $k = 1$ si et seulement la réponse est oui pour le problème PP.*

Preuve On prouve ce théorème par double implication. Tout d'abord, l'implication directe. On prouve que la réponse pour le problème PP est oui en supposant que la réponse est oui pour le problème PSP avec $k = 1$. Cela signifie qu'il y a au moins un ordonnancement qui satisfait les contraintes de périodicité des opérations. On note par $\mathcal{S}_{PSP}$ l'ensemble des ordonnancements qui satisfont les contraintes de périodicité. On étudie le sous-ensemble

$\mathcal{S}_<$ des ordonnancements obtenus en ordonnançant les opérations dans l'ordre croissant des périodes. Evidemment l'ensemble $\mathcal{S}_<$ n'est pas vide. Soit $\mathcal{W} = \mathcal{V}$ un ensemble de travail.

Dans un ordonnancement quelconque $S \in \mathcal{S}_<$ on observe que les opérations avec les périodes égales à la plus petite période T_1 forment un motif qui a une période T_1. Si dans ce motif l'ordre partiel défini par le graphe $\mathcal{G}$ n'était pas satisfait on peut changer les opérations entre elles afin qu'elles satisfassent l'ordre partiel. On ne modifie pas la période du motif, donc les opérations du motif satisfont toujours leurs contraintes de périodicité. Aucune opération appartenant au motif ne peut avoir un prédécesseur avec une période plus grande que la sienne (inégalité 2.1), donc en modifiant le motif on satisfait toutes les contraintes de précédence entre les opérations avec les périodes égales à la plus petite période et leurs prédécesseurs. On modifie $\mathcal{W} = \mathcal{W} \backslash \{A \in \mathcal{V}$ avec la période égale à $T_1\}$.

On passe aux motifs des opérations avec les périodes égales à T_2, la plus petite période des opérations appartenant à $\mathcal{W}$. De la même manière, si dans les motifs l'ordre partiel défini par le graphe n'était pas satisfait on peut changer les opérations entre elles afin qu'elles satisfassent l'ordre partiel. On ne modifie pas la période des motifs, donc les opérations des motifs satisfont toujours leurs contraintes de périodicité. On modifie $\mathcal{W} = \mathcal{W} \backslash \{A \in \mathcal{V}$ avec la période égale à $T_2\}$. On répète ce pas de manière récursive jusqu'au moment où l'ensemble $\mathcal{W}$ est vide. On a modifié l'ordonnancement S afin qu'il satisfasse les contraintes de précédence définies par le graphe $\mathcal{G}$ tout en gardant les contraintes de périodicité satisfaites. Donc la réponse pour le problème PP est aussi oui.

On prouve l'implication inverse. On prouve que la réponse pour le problème PSP avec $k = 1$ est oui en supposant que la réponse est oui pour le problème PP. Il y a donc au moins un ordonnancement qui satisfait les contraintes de précédences et de périodicités. Étant donné que l'ordonnancement satisfait les contraintes de périodicité alors la réponse est oui aussi pour le problème PSP avec $k = 1$. Le théorème est prouvé $\boxdot$

L'exemple suivant illustre la preuve du dernier théorème. Il faut noter que bien que les deux systèmes n'aient pas les mêmes solutions du point de vue de l'ordonnancement, ils sont cependant ordonnançables de telle manière que l'un est ordonnançable si et seulement si l'autre est ordonnançable.

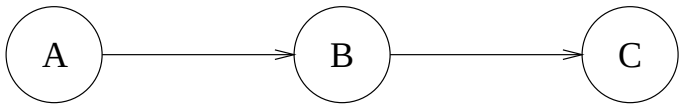

FIG. 2.13 – *Graphe illustrant la preuve du théorème 9*

Exemple 4 *On considère un premier système avec trois opérations A, B et C avec les durées d'exécution $C_A = C_C = 2$ et $C_B = 4$ et les périodes $T_A = T_C = T_B = 8$ et aucune contrainte de précédence n'est définie entre deux opérations différentes. On considère un deuxième système avec trois opérations A, B et C avec les durées d'exécution $C_A = C_C = 2$ et $C_B = 4$ et les périodes $T_A = T_C = T_B = 8$ et on impose des contraintes de précédence définies par le graphe donné dans la figure 2.13. Pour le premier système on trouve un ordonnancement*

donné dans la figure 2.14 et pour le deuxième système on trouve un ordonnancement donné dans la figure 2.15. On remarque que les deux ordonnancements ont le même motif qui se répète, donc si il y a un ordonnancement pour un des sytèmes, l'ordonnancement existe aussi pour l'autre système. En conséquence l'ordonnançabilité d'un système implique l'ordonnançabilité de l'autre système.

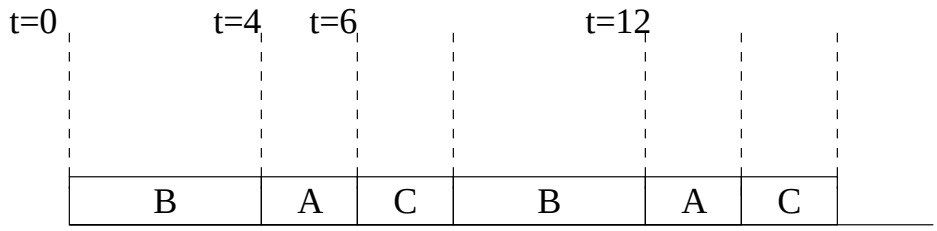

FIG. 2.14 – *Ordonnancement satisfaisant les contraintes du premier système*

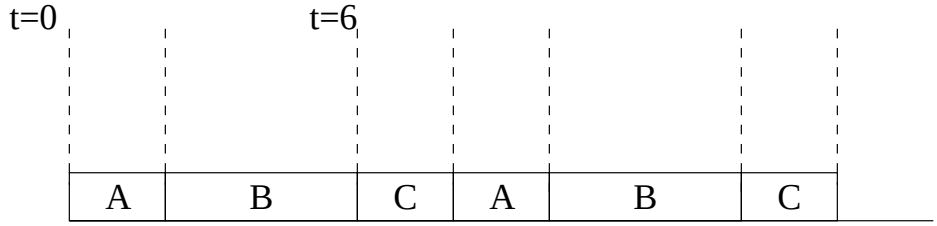

FIG. 2.15 – *Ordonnancement satisfaisant les contraintes du deuxième système*

Corollaire 1 *Le problème PP est NP-complet au sens fort.*

En effet, dûs aux théorèmes 7 et 9 le problème de décision PP associé à notre problème d'ordonnancement a la même complexité que le problème de décision PSP.

On note par PPD le problème de décision associé à notre problème d'ordonnancement des opérations avec des périodes et de durées d'exécution formant une séquence de valeurs divisibles.

Corollaire 2 *La réponse est oui pour le problème $PSPD$ avec $k = 1$ si et seulement la réponse est oui pour le problème PPD.*

En effet, étant donné que dans la preuve du théorème 9 on n'a pas imposé de restriction sur les valeurs des périodes et des durées d'exécution, le théorème 9 se particularise pour le cas avec des périodes et des durées d'exécution formant une séquence de valeurs divisibles et on obtient le corollaire 2.

Ce dernier résultat implique que le problème de décision PPD est polynômial. Par conséquence, dans la suite on choisira les valeurs de périodes et de durées d'exécution comme étant une séquence de valeurs divisibles.

2.1.2 Algorithme optimal d'ordonnancement

La relation d'ordre imposée par le graphe qui définit les contraintes de précédence est une relation d'ordre partiel. Un algorithme d'ordonnancement transforme l'ordre partiel en un (un des ordres possibles) ordre total qui satisfait les contraintes de périodicité des opérations. Une fois que la première répétition d'une opération avec contrainte de périodicité est ordonnancée, on connaît les dates de début des répétitions suivantes de l'opération.

Remarque 5 *Il s'agit d'ordonnancer un ensemble infini d'opérations donc l'algorithme aura nécessairement un nombre infini de pas à exécuter ce qui est contradictoire avec la notion classique d'algorithme. Cependant nous utiliserons par la suite par abus de language la notion d'algorithme pour indiquer la manière de trouver un ordonnancement.*

Pour le déroulement de l'algorithme d'ordonnancement, on note par $\mathcal{W}$ l'ensemble d'opérations disponibles, par s_T la date de début de la dernière opération ordonnancée, par C_T sa durée d'exécution, par $\mathcal{P}er$ l'ensemble des opérations qui ont une contrainte de périodicité et par $Prec(A)$ l'ensemble des prédécesseurs de l'opération A. Pendant le déroulement de l'algorithme, l'ensemble $\mathcal{W}$ contient les opérations disponibles, c'est-à-dire les opérations dont les prédécesseurs sont déjà ordonnancés. Aussi, pendant le déroulement de l'algorithme, chaque fois qu'une opération est ordonnancée, s_T (respectivement C_T) est remplacé par la date de début (respectivement la durée d'exécution) de cette opération.

Algorithme 1

Initialisation : $\mathcal{W} = \bigcup_{A \in \mathcal{V} \ \textit{et}\ Prec(A)=\emptyset} \{A\}$ et $s_T = 0, C_T = 0$.

Pas 1 *(opération sans contrainte de périodicité)* :
si $\exists A \in \mathcal{W}$ t.q. $A \notin \mathcal{P}er$, alors $s_A = s_T + C_T$ et on va au Pas 1. Sinon, on passe au Pas 2.

Pas 2 : on cherche une opération $A \in \mathcal{W} \bigcap \mathcal{P}er$ avec A_1 déjà ordonnancée tel que :

$$s_{A_i} + T_A - s_T - C_T = min_{B_i \in \mathcal{W} \bigcap \mathcal{P}er \ \textit{et}\ B_1 \ \textit{déjà ordonnancée}} \{s_{B_i} + T_B - s_T - C_T\}$$

où A_i est la dernière répétition ordonnancée de A. Si on trouve plusieurs opérations A alors le système n'est pas ordonnançable et l'algorithme s'arrête.

Pas 3 *(opération sans contrainte de périodicité)* :
si $\exists C \in \mathcal{W} \backslash \{\{A\} \bigcap \mathcal{P}er\}$ t.q. $s_T + C_T + C_C \leq s_{A_i} + T_A$ pour l'opération A trouvée précédemment, alors on a $s_C = s_T + C_T$ et on va au Pas 2. Sinon, on va au Pas 4.

Pas 4 *(opération avec contrainte de périodicité mais sans la première répétition déjà ordonnancée)* :
si $\exists C \in \mathcal{W} \bigcap \mathcal{P}er$ avec C_1 pas ordonnancée et $s_T + C_T + C_C \leq s_{A_i} + T_A$ avec A étant l'opération trouvée au Pas 2, alors on a $s_C = s_T + C_T$ avec $T_C = min_{D \in \mathcal{W} \bigcap \mathcal{P}er \ \textit{et}\ C_D \leq T_A} \{T_D\}$, et avec C ayant la plus grande durée d'exécution, on enlève l'opération de $\mathcal{W}$, toutes les

opérations qui deviennent ordonnançables sont rajoutées à $\mathcal{W}$ et on va au Pas 2. Sinon, on va au Pas 5.

Pas 5 *(opération avec contrainte de périodicité)*:
on a $s_A = s_T + C_T$ t.q. $s_{A_{i+1}} = s_{A_i} + T_A$ et on va au Pas 2.

Remarque 6 *l'algorithme d'ordonnancement ne s'arrête jamais, sauf si les contraintes de périodicité ne peuvent pas être satisfaites au Pas 2 et dans ce cas le système n'est pas ordonnançable.*

Avant de prouver l'optimalité de l'algorithme d'ordonnancement 1, on donne un exemple d'un système ordonnancé en utilisant l'algorithme.

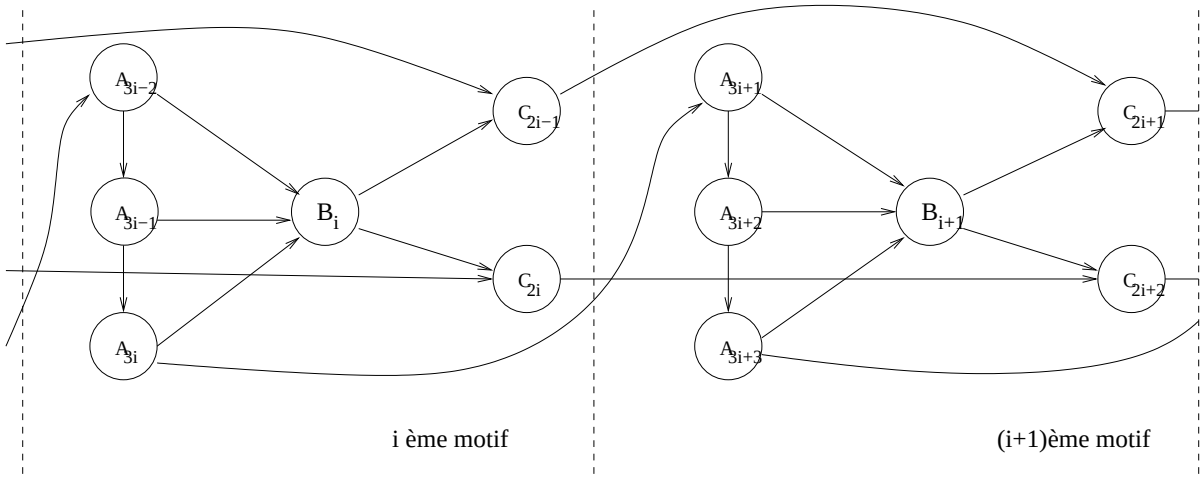

FIG. 2.16 – *Graphe illustrant l'utilisation de l'algorithme d'ordonnancement 1*

Exemple 5 *Soit $\mathcal{G} = (\mathcal{V},\mathcal{E})$ donné dans la figure 2.16 avec les durées d'exécution $C_A = C_B = C_C = 1$ et les périodes $T_A = 5$, $T_B = 15$. Le tableau 2.1 donne le déroulement de l'algorithme 1.*

Théorème 10 *L'algorithme d'ordonnancement 1 est optimal (s'il y a un ordonnancement, l'algorithme le trouvera).*

Preuve On utilise un raisonnement par l'absurde. On suppose qu'il y un ensemble $\mathcal{S}$ de tous les ordonnancements qui satisfont les contraintes de précédences et de périodicités et qu'après avoir utilisé l'algorithme on n'a trouvé aucun ordonnancement. Cela signifie qu'aucun des ordonnancements appartenant à $\mathcal{S}$ n'est obtenu en ordonnançant les opérations dans l'ordre croissant des périodes. Mais les précédences impliquent que les opérations disponibles sont celles avec leurs prédécesseurs déjà ordonnancés, donc leurs prédécesseurs ont des périodes plus petites que leurs propres périodes et ils sont ordonnancés avant elles. Donc les

$\mathcal{W}$	s_T	C_T	Pas utilisé	Décision
$\{A_1\}$	0	0	Pas 4	$s_{A_1} = 0$
$\{A_2\}$	0	1	Pas 2	A choisi
$\{A_2\}$	0	1	Pas 5	$s_{A_2} = 5$
$\{A_3\}$	5	1	Pas 2	A choisi
$\{A_3\}$	5	1	Pas 5	$s_{A_3} = 10$
$\{A_1,B\}$	10	1	Pas 2	A choisi
$\{A_1,B\}$	10	1	Pas 4	$s_B = 11$
$\{A_1,C_1,C_2\}$	10	1	Pas 2	A choisi
$\{A_1,C_1,C_2\}$	11	1	Pas 3	$s_{C_1} = 12$
$\{A_1,C_2\}$	10	1	Pas 2	A choisi
$\{A_1,C_2\}$	12	1	Pas 3	$s_{C_2} = 13$
$\{A_1\}$	13	1	Pas 2	A choisi
$\{A_1\}$	13	1	Pas 4	$s_{A_1} = 15$
$\{A_2\}$	15	1	Pas 2	A choisi
$\cdots$				

TAB. 2.1 – *Ordonnancement obtenu en utilisant l'algorithme 1*

ordonnancements appartenant à $\mathcal{S}$ ne sont pas obtenus en ordonnançant les opérations qui ne sont pas liées par un chemin dans l'ordre croissant des périodes. Sans perdre de généralité on considère deux opérations A et B avec $T_A < T_B$ qui ne sont pas liées par un chemin et B est ordonnancée après A dans tous les ordonnancements appartenant à $\mathcal{S}$. Étant donné que $T_A < T_B$ et que A et B ne sont pas liées par un chemin à partir de la répétition n (avec n suffisamment grand) A va être toujours ordonnancée avant B. En faisant tendre n vers l'infini, A va être toujours ordonnancée avant B, donc il est possible que A soit ordonnancée avant B depuis le début d'un ordonnancement sans que les autres contraintes soient "pénalisées". En appliquant ce raisonnement de manière récursive on obtient qu'il y a des ordonnancements qui ordonnancent les opérations dans l'ordre croissant des périodes et qui satisfont les contraintes, donc $\mathcal{S}$ ne contient pas tous les ordonnancements qui satisfont les contraintes de précédences et de périodicités. On a trouvé une contradiction avec notre supposition, donc l'algorithme d'ordonnancement trouve toujours un ordonnancement s'il y en a un. Le théorème est prouvé $\boxdot$

2.1.3 Condition d'ordonnançabilité

Une condition d'ordonnançabilité pour un système d'opérations avec des contraintes de précédences et de périodicités est la conséquence directe de l'existence d'une hyper-période, c'est-à-dire de l'existence d'un motif répété infiniment dans l'ordonnancement. Ce motif inclut ou est égal au motif du graphe qui définit les contraintes de précédence du système. Quelles que soient les opérations A et B répétées finiment, respectivement, n et m fois dans le motif répété infiniment avec au moins une contrainte de précédence d'une répétition de A

à une répétition de B, si l'égalité $nT_A = mT_B$ est satisfaite alors l'hyperpériode est égale au motif du graphe .

L'existence d'une hyper-période est donnée en prouvant qu'une opération non-périodique hérite de la période de ses successeurs et que dans un ordonnancement obtenu avec l'algorithme 1 les opérations non-périodiques sont ordonnancées avec la période de ses prédécesseurs. Tout d'abord on prouve que pour le cas de deux opérations A et B avec une contrainte de précédence entre elles, B étant le seul successeur de A avec une contrainte de périodicité, en forçant A à être ordonnancée d'une manière périodique entre deux répétitions successives de l'opération B, le système reste ordonnançable s'il était ordonnançable sans cette contrainte supplémentaire.

Théorème 11 *Soit A et B avec $(A,B) \in \mathcal{E}$ deux opérations appartenant à un système défini par un graphe $\mathcal{G} = (\mathcal{V},\mathcal{E})$. On suppose que l'opération A n'a aucune contrainte de périodicité et B est le seul successeur de A qui a une contrainte de périodicité T_B. Soit $\mathcal{G}' = (\mathcal{V},\mathcal{E}')$ le graphe obtenu à partir de $\mathcal{G}$ en rajoutant une contrainte de précédence entre l'opération B du i^{me} motif et l'opération A du $(i+1)^{me}$ motif, $\forall i \in \mathbb{N}$. Si le système défini par $\mathcal{G}$ est ordonnançable alors le système défini par $\mathcal{G}'$ est aussi ordonnançable et A hérite la contrainte de périodicité de B (les deux ordonnancements satisfont la même contrainte de périodicité pour B et l'opération A, respectivement A et B).*

Preuve Soit $S = \{s_A | A \in \mathcal{V}\}$ un ordonnancement quelconque pour un système défini par $\mathcal{G}$. Alors S satisfait les contraintes de précédence de A et B et la contrainte de périodicité de B. On exprime la contrainte de périodicité de B :

$$s_{B_i} - s_{B_{i-1}} = T_B, \forall i \geq 1 \tag{2.3}$$

où s_B est la date de début de B et B_i est la i^{me} répétition de B. On exprime les contraintes de précédence :

$$s_{A_i} + C_A \leq s_{B_i}, \forall i \geq 1 \tag{2.4}$$

L'ordonnancement S peut contenir plusieurs répétitions de A avant la première répétition de B. Soit n_0 le nombre de répétitions de A ordonnancées avant la première répétition de B. Donc, on a $s_{A_1} + n_0 C_A \leq s_{B_1}$. L'inégalité (2.4) implique que A_{n_0+1} doit être ordonnancée avant B_{n_0+1}. On a au moins une répétition de A entre deux répétitions consécutives de B, c'est-à-dire $\exists i_0 \leq n_0$ t.q. :

$$s_{B_{i_0}} + C_B \leq s_{A_{n_0+1}} \tag{2.5}$$

$$s_{A_{n_0+1}} + C_A \leq s_{B_{i_0+1}} \tag{2.6}$$

En additionnant les inégalités (2.5) et (2.6), on obtient $s_{B_{i_0+1}} - s_{B_{i_0}} \geq C_A + C_B$ et en utilisant (2.3), on a $T_B \geq C_A + C_B$. Cette inégalité montre qu'entre deux répétitions consécutives de B, il y a un temps suffisant pour ordonnancer une répétition de A.

Sans perdre de généralité, on peut supposer que $s_{A_{n_0}}$ est égal à $s_{B_{i_0}} + \mathcal{C}$, où $\mathcal{C} < T_B - C_B$ est une constante entière. Puisque les périodes forment une séquence de valeurs divisibles alors elles ne sont pas premières entre elles et comme A a été ordonnancée à $s_{B_{i_0}} + \mathcal{C}$, il n'y

a aucune opération périodique qui doit être ordonnancée à $s_{B_{i_0}} + \mathcal{C} + iT_B$. Cela signifie qu'il y a un ordonnancement S' dont les dates de début de A et B satisfont :

$$S' = \begin{cases} s_{B_i} = s_{A_i} + T_B - \mathcal{C}, \forall i \geq 1 \\ s_{A_{i+1}} = s_{A_i} + T_B, \forall i \geq 1 \end{cases} \tag{2.7}$$

Les équations de (2.7) montrent que S' est un ordonnancement qui satisfait les contraintes définies par $\mathcal{G}$ ainsi que la contrainte de précédence entre B_i à A_{i+1}. Donc, l'ordonnancement S' est un ordonnancement pour le système défini par $\mathcal{G}'$. En conséquence, toutes les dates de début de A dans l'ordonnancement S' satisfont une contrainte de périodicité $T = T_B$. Donc, A hérite la contrainte de périodicité de B. Le théorème est prouvé ⊡

On généralise ce résultat pour le cas d'une opération A avec n successeurs avec contraintes de périodicité. Le théorème suivant prouve que dans ce cas A hérite d'une contrainte de périodicité égale à la plus petite période des successeurs de A.

Théorème 12 *(généralisation du théorème 11)*
Soit $A,B_i,\forall i \in \{1,2,\ldots,n\}$ *avec* $(A,B_i) \in \mathcal{E},\forall i \in \{1,2,\ldots,n\}$ $n+1$ *opérations appartenant à un système défini par un graphe* $\mathcal{G} = (\mathcal{V},\mathcal{E})$. *On suppose que* A *n'a aucune contrainte de périodicité et* $B_i,\forall i \geq 1$ *sont les seuls successeurs périodiques de* A *(*B_1 *a la plus petite contrainte de périodicité parmi toutes le opérations* B_i*). Soit* $\mathcal{G}' = (\mathcal{V},\mathcal{E}')$ *le graphe obtenu à partir du graphe* $\mathcal{G}$ *en rajoutant une contrainte de précédence de* B_1 *du* j^{me} *motif à* A *du* $(j+1)^{me}$ *motif, alors le système défini par* $\mathcal{G}'$ *est aussi ordonnançable.* A *hérite la contrainte de périodicité* $T_{B_1} = min_{i\in\{1,\ldots,n\}}\{T_{B_i}\}$ *(les deux ordonnancements satisfont la même contrainte de périodicité pour* $B_i,\forall i \geq 1$ *et ils utilisent les mêmes durées d'exécution pour* A *et* $B_i,\forall i \geq 1$*).*

Preuve Sans perdre de généralité on suppose que les opérations sont ordonnées dans l'ordre croissant de leurs périodes, donc on a $T_{B_1} = min_{i\in\{1,\ldots,j\}}\{T_{B_i}\}$.

Le choix de la plus petite période est prouvé en utilisant un raisonnement par l'absurde. On suppose que A hérite une autre période T_i alors $T_i > T_1$. Alors on a $T_A > T_{B_1}$ et comme il y a une précédence entre A et B on a une contradiction avec l'inégalité 2.1. Donc A ne peut hériter d'une autre période que T_1 et on a vu dans le théorème 11 que si A hérite de la période d'un successeur le système reste ordonnançable. Le théorème est prouvé ⊡

La propriété d'héritage de la plus petite période de ses successeurs donnée par le théorème 12 est obtenue en étudiant un ordonnancement quelconque. Les théorèmes suivants qui prouvent l'héritage de périodes de ses prédécesseurs d'une opérations sont obtenus en étudiant des ordonnancements particuliers obtenus à l'aide de l'algorithme optimal d'ordonnancement 1. On traite directement le cas d'une opération avec n prédécesseurs périodiques.

Théorème 13 *Soit* B *une opération appartenant à un système défini par un graphe* $\mathcal{G} = (\mathcal{V},\mathcal{E})$. *On suppose que* B *n'a aucune contrainte de périodicité et* $\exists n$ *prédécesseurs* A_i *périodiques. Si le système est ordonnançable, alors* B *hérite de la contrainte de périodicité* $T = max_{i\in\{1,\ldots,n\}}\{T_{A_i}\}$.

Preuve Sans perdre de généralité, on considère que A_n est l'opération avec la plus grande contrainte de périodicité parmi toutes les $A_i, \forall i \in \{1,\ldots,n\}$. Puisque A_n a la plus grande contrainte de périodicité, elle sera ordonnancée la dernière parmi les opérations $A_i, \forall i \in \{1,\ldots,n\}$ et la prochaine répétition de A_n sera aussi la dernière parmi les opérations $A_i, \forall i \in \{1,\ldots,n\}$. Puisque le système est ordonnançable, B est ordonnancée entre deux répétitions consécutives de A_n, une fois que la première répétition de A_n est ordonnancée. Puisque les contraintes de périodicité ne sont pas premières entre elles et B a été ordonnancée à s_B, il n'y a aucune opération périodique qui doit être ordonnancée à $s_B + iT_{A_n}, \forall i \geq 1$. Cela implique que B hérite la contrainte de périodicité de A_n Le théorème est prouvé ⊡

Après avoir étudié séparément l'héritage de la périodicité des successeurs d'une opération et l'héritage de la périodicité des prédécesseurs d'une opération, le corollaire suivant traite le cas général d'une opération qui a des prédécesseurs avec contraintes de périodicité et des successeurs avec contraintes de périodicité.

Corollaire 3 *Soit B une opération appartenant à un système défini par un graphe $\mathcal{G} = (\mathcal{V},\mathcal{E})$. On suppose que B n'a aucune contrainte de périodicité. On suppose que $\exists n$ opérations périodiques A_i t.q. $\exists P(A_i,B) \in \mathcal{P}$ et $\exists m$ opérations périodiques C_j t.q. $\exists P(B,C_j) \in \mathcal{P}$. Si le système est ordonnançable alors B hérite la contrainte de périodicité $T = min_{j\in\{1,\ldots,m\}}\{T_{C_j}\}$.*

Preuve Sans perdre de généralité on suppose que les opérations sont ordonnées dans l'ordre croissant des périodes. Donc on a $T_{C_1} = min_{j\in\{1,\ldots,m\}}\{T_{C_j}\}$ et $T_{A_n} = min_{i\in\{1,\ldots,n\}}\{T_{A_i}\}$. En plus on a $T_{A_n} \leq T_{C_1}$, puisque $\exists P(A_i,B) \in \mathcal{P}$ et $\exists P(B,C_j) \in \mathcal{P}$.

En regardant les théorèmes 12 et 13 on observe que l'héritage des périodes des successeurs est imposé pour avoir un ordonnancement et que l'héritage des périodes des prédécesseurs est un choix. Donc la satisfaction de la contrainte de périodicité d'un successeur dépend de l'opération, mais la satisfaction de la contrainte de périodicité d'un prédécesseur ne dépend pas de l'opération. De plus la période d'un prédécesseur est plus petite que la période d'un successeur, donc elle est plus forte. En conséquence on fait le choix de rendre l'opération B périodique avec la période de ses successeurs, sans que le système devienne non-ordonnançable. On applique ce raisonnement d'une manière récursive en utilisant la transitivité de l'ordre partiel défini par le graphe. Le corollaire est prouvé ⊡

Remarque 7 *L'opération non-périodique A dont les prédécesseurs sont périodiques est une opération apériodique et l'opération non-périodique B dont les successeurs sont périodiques est une opération sporadique. En effet, dans le premier cas les contraintes de périodicité des prédécesseurs ne donnent aucune information concernant la date de début de l'opération A, en revanche dans le deuxième cas les contraintes de périodicité des successeurs permettent de placer la date de début de l'opération B entre deux répétitions consécutives de prédécesseurs avec la plus petite période.*

Exemple 6 *On présente un exemple d'un système défini par le graphe $\mathcal{G}$ donné dans la figure 2.17. On a $T_A = 5$ et $T_D = 10$. En utilisant le corollaire 3, C hérite les contraintes de périodicité de A et de D, E hérite la contrainte de périodicité de A, et B hérite la contrainte de périodicité de C. Donc, on a $T_A = T_B = T_C = T_E = 5$ et $T_D = 10$. Pour $C_A = C_B = C_C = C_D = C_E = 1$ on obtient l'ordonnancement S (voir figure 2.18).*

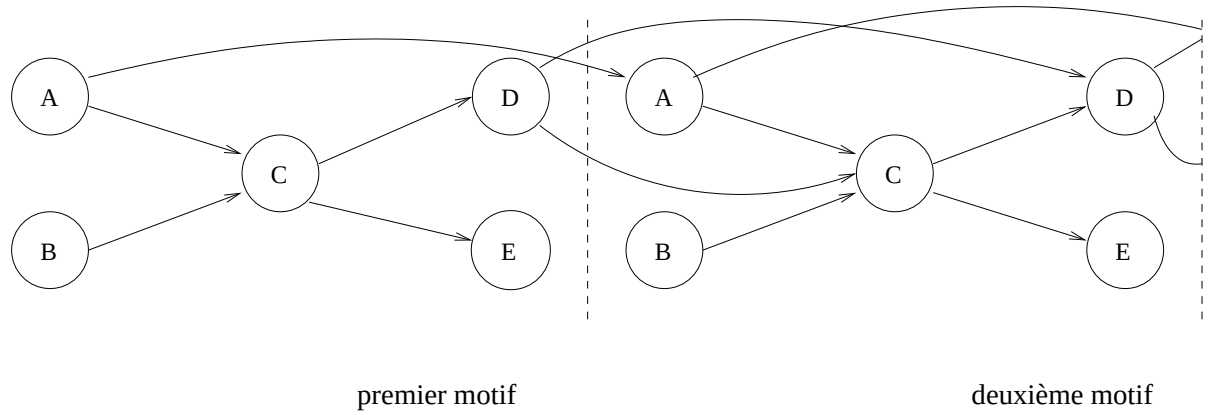

FIG. 2.17 – *Graphe illustrant l'héritage de périodes*

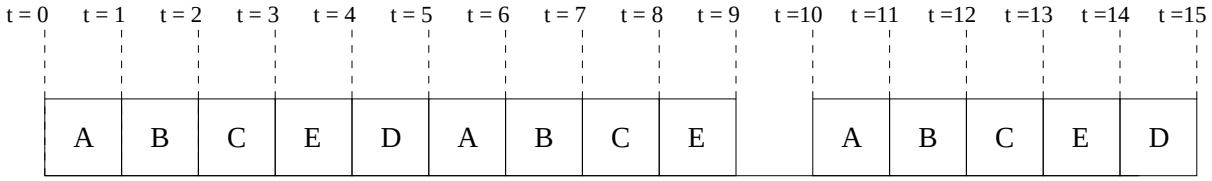

FIG. 2.18 – *Ordonnancement pour le système défini à l'aide du graphe de la figure 2.17*

Etant donné qu'on a fait la supposition qu'il y a au moins une opération périodique dans chaque composante connexe du graphe et à cause de la transitivité des précédences, l'héritage des périodes dans un graphe rend toutes les opérations du graphe périodiques. En conséquence lors de l'ordonnancement il n'y a plus d'opération sans contrainte de périodicité à ordonnancer. Donc l'algorithme d'ordonnancement 1 devient plus simple, les pas qui ordonnancent les opérations sans contrainte de périodicité vont disparaître. On obtient l'algorithme suivant :

Algorithme 2

Initialisation : $\mathcal{W} = \bigcup_{A \in \mathcal{V} \ \textit{et}\ Prec(A)=\emptyset} \{A\}$ et $s_T = 0, C_T = 0$.

Pas 1 :
on ordonnance l'opération $A \in \mathcal{W}$ tel que $T_A = min_{D \in \mathcal{W}} T_D$, A ayant la plus grande durée d'exécution. On a $s_A = 0$, on enlève l'opération de $\mathcal{W}$, toutes les opérations qui deviennent ordonnançables sont rajoutées à $\mathcal{W}$ et on va au Pas 2.

Pas 2 : on cherche une opération $A \in \mathcal{W} \bigcap \mathcal{P}er$ avec A_1 déjà ordonnancée tel que :

$$s_{A_i} + T_A - s_T - C_T = min_{B_i \in \mathcal{W} \bigcap \mathcal{P}er \ \textit{et}\ B_1 \ \textit{déjà ordonnancée}} \{s_{B_i} + T_B - s_T - C_T\}$$

où A_i est la dernière répétition ordonnancée de A. Si on trouve plusieurs opérations A alors le système n'est pas ordonnançable et l'algorithme s'arrête.

Pas 4 *(opération avec contrainte de périodicité mais sans la première répétition déjà ordonnancée)* :

si $\exists C \in \mathcal{W}$ avec C_1 pas ordonnancée et $s_T + C_T + C_C \leq s_{A_i} + T_A$ avec A étant l'opération trouvée au Pas 2, alors on a $s_C = s_T + C_T$ avec $T_C = min_{D \in \mathcal{W}\ et\ C_D \leq T_A}\{T_D\}$, et avec C ayant la plus grande durée d'exécution, on enlève l'opération de $\mathcal{W}$, toutes les opérations qui deviennent ordonnançables sont rajoutées à $\mathcal{W}$ et on va au Pas 2. Sinon, on va au Pas 5.

Pas 5 :
on a $s_A = s_T + C_T$ t.q. $s_{A_{i+1}} = s_{A_i} + T_A$ et on va au Pas 2.

Avant de prouver l'optimalité de l'algorithme d'ordonnancement 2 on donne un exemple de l'utilisation de l'algorithme.

Exemple 7 *On reprend l'exemple 6 avec le graphe $\mathcal{G} = (\mathcal{V},\mathcal{E})$ donné dans la figure 2.17 avec les durées d'exécution $C_A = C_B = C_C = C_D = C_E = 1$ et les périodes $T_A = T_B = T_C = T_E = 5$ et $T_D = 10$. L'ordonnancement obtenu en utilisant l'algorithme d'ordonnancement 2 est donné dans le tableau 2.2.*

$\mathcal{W}$	s_T	C_T	Pas utilisé	Décision
$\{A,B\}$	0	0	Pas 1	$s_A = 0$
$\{A,B\}$	0	1	Pas 2	A choisi
$\{A,B\}$	0	1	Pas 3	$s_B = 1$
$\{A,B,C\}$	1	1	Pas 2	A choisi
$\{A,B,C\}$	1	1	Pas 3	$s_C = 2$
$\{A,B,E,D\}$	2	1	Pas 2	A choisi
$\{A,B,E,D\}$	2	1	Pas 3	$s_E = 3$
$\{A,B,D\}$	3	1	Pas 2	A choisi
$\{A,B,D\}$	3	1	Pas 3	$s_D = 4$
$\{A,B\}$	4	1	Pas 2	A choisi
$\{A,B\}$	4	1	Pas 4	$s_A = 5$
$\cdots$				

TAB. 2.2 – *Ordonnancement obtenu en utilisant l'algorithme 2*

Corollaire 4 *L'algorithme 2 est optimal (s'il y a un ordonnancement, l'algorithme le trouvera).*

Preuve Evidemment l'algorithme 2 est l'algorithme 1 appliqué à une classe particulière de systèmes avec contraintes de périodicité, dont on a enlevé les pas qui ne sont pas utilisés. Alors en utilisant le théorème 10, l'algorithme 2 reste optimal. Le corollaire est prouvé $\boxdot$

On note par T le PPCM de toutes les contraintes de périodicité, qui du fait de la relation entre les périodes, est égal à la plus grande période des opérations. Pour un ordonnancement S et deux valeurs quelconques $t_1 < t_2$ du temps écoulé depuis le début de l'ordonnancement, on note par $S(t_1,t_2)$ la séquence de dates de début des opérations ordonnancées de t_1 à t_2.

Le théorème suivant prouve l'existence d'une hyper-période de l'ordonnancement en montrant qu'il y a un motif qui se répéte dans l'ordonnancement.

Théorème 14 *pour un système avec des contraintes de précédences et de périodicités, on a*

$$S(s_{max}+iT,s_{max}+iT+t) = S(s_{max}+(i+1)T,s_{max}+(i+1)T+t),\forall 0 < t \leq T \text{ et } i \geq 0$$

où s_{max} est la date de début de la dernière opération ordonnancée parmi les opérations du premier motif.

Preuve La preuve est faite par double récurrence selon les entiers i et t. Soit $P(i),\forall i \geq 0$ la proposition $S(s_{max}+iT,s_{max}+iT+t) = S(s_{max}+(i+1)T,s_{max}+(i+1)T+t),\forall 0 < t \leq T$. D'abord, on vérifie que $P(0)$ est vraie par récurrence selon t. Soit $Q(t) = P(0)$, $Q(t)$ est la proposition $S(s_{max},s_{max}+t) = S(s_{max}+T,s_{max}+T+t),\forall 0 < t \leq T$. On vérifie que $S(s_{max},s_{max}+1) = S(s_{max}+T,s_{max}+T+1)$. En effet, $S(s_{max},s_{max}+1)$ contient seulement la première opération ordonnancée qui a: soit sa propre contrainte de périodicité, soit la périodicité héritée de ses successeurs qui est la plus petite période. Puisque T est le PPCM de toutes les contraintes de périodicité, la première opération est ordonnancée, aussi, à $s_{max}+T$. En conséquence, $S(s_{max}+T,s_{max}+T+1)$ contient exactement la même opération que $S(s_{max},s_{max}+1)$ (on rappelle qu'une contrainte de périodicité est donnée par un entier). Donc $S(s_{max},s_{max}+1) = S(s_{max}+T,s_{max}+T+1)$ est vraie.

En appliquant la récurrence, on suppose que $Q(t)$ est vraie et on vérifie que $Q(t+1)$ est aussi vraie ($Q(t)$ vraie implique $Q(t+1)$ vraie). On a $S(s_{max},s_{max}+t) = S(s_{max}+T,s_{max}+T+t)$ et on veut prouver que $S(s_{max},s_{max}+t+1) = S(s_{max}+T,s_{max}+T+t+1)$, ce qui signifie que la même opération a été ordonnancée à $s_{max}+t$ et à $s_{max}+T+t$. On a deux possibilités: soit la dernière opération ordonnancée se finit avant $s_{max}+t$, soit non.

Dans le premier cas, puisqu'on a les mêmes opérations ordonnancées de s_{max} à $s_{max}+t$ comme de $s_{max}+T$ à $s_{max}+T+t$, on va avoir le même ensemble d'opérations à $s_{max}+t$ et à $s_{max}+T+t$ (on suppose qu'on utilise un algorithme statique d'ordonnancement, par exemple notre algorithme d'ordonnancement). Cela implique que la même opération est ordonnancée à $s_{max}+t$ et à $s_{max}+T+t$.

Dans le deuxième cas, puisque l'ordonnancement est non-préemptif, la dernière opération va utiliser la ressource à $s_{max}+t$ et à $s_{max}+T+t$.

Donc, on a prouvé que $Q(t+1)$ est vraie si $Q(t)$ était vraie. Puisque $Q(t)$ est vraie, on obtient $P(0)$ vraie.

Suivant la récurrence, on suppose que $P(i)$ est vraie et on vérifie que $P(i+1)$ est aussi vraie ($P(t)$ vraie implique $P(i+1)$ vraie). La preuve se fait exactement comme dans le cas $Q(t)$ vraie implique $Q(t+1)$ vraie.

Donc, $P(i)$ est vraie, $\forall i \geq 0$. Le théorème est prouvé $\boxdot$

L'existence d'une hyper-période nous permet de donner une condition nécessaire d'ordonnançabilité en étudiant le nombre de répétitions de chaque opération dans une hyper-période.

Corollaire 5 *Si un système défini par* $\mathcal{G} = (\mathcal{V},\mathcal{E})$ *est ordonnançable, alors* $\sum_{A\in\mathcal{V}} \frac{C_A}{T_A} \leq 1$.

Preuve Le théorème 14 prouve qu'un ordonnancement d'un système avec contraintes de précédences et de périodicités a une hyper-période T égale au PPCM de toutes les périodes. Donc, si un système est ordonnançable alors il y a un ordonnancement de 0 à T.

Puisque toutes les périodes ne sont pas premières entre elles, deux dates de début ne peuvent pas être égales. Une opération A avec une contrainte de périodicité T_A va être ordonnancée $n_A = \frac{T}{T_A}$ fois de 0 à T. Donc, si un système est ordonnançable alors toutes les opérations sont ordonnancées de 0 à T et on a $\sum_{A\in\mathcal{V}} n_A C_A \leq T$. Puisque $n_A = \frac{T}{T_A}$, on obtient que l'ordonnançabilité d'un système implique $\sum_{A\in\mathcal{V}} \frac{C_A}{T_A} \leq 1$. Le corollaire est prouvé ⊡

Le fait qu'on étudie des ordonnancements sans préemption ne permet pas à cette condition nécessaire d'être aussi une condition suffisante et le contre-exemple suivant sert à prouver cela.

FIG. 2.19 – *Graphe illustrant le corollaire 5*

Exemple 8 *On considère le graphe* $\mathcal{G} = (\mathcal{V},\mathcal{E})$ *donné dans la figure 2.19 avec les durées d'exécution* $C_A = 2$ *et* $C_B = 4$ *et les périodes* $T_A = 2$ *et* $T_B = 4$. *L'inégalité* $\sum_{A\in\mathcal{V}} \frac{C_A}{T_A} \leq 1$ *est satisfaite, mais aucun ordonnancement non-préemptif existe puisque la période de A empêche la première répétition de B de s'ordonnancer. Dans le cas préemptif [4] la condition est suffisante et il y a au moins un ordonnancement qui satisfait les contraintes (figure 2.20).*

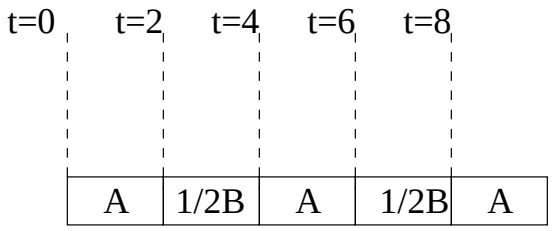

FIG. 2.20 – *Ordonnancement préemptif*

Une autre application de l'existence d'une hyper-période concerne l'algorithme d'ordonnancement 1. Cet algorithme construit un ordonnancement infini, mais l'existence d'une hyper-période permet de trouver un ordonnancement en bornant l'algorithme borné en nombre de pas. On est donc ainsi dans le cas d'un algorithme classique. En plus, son utilisation permet de tester l'ordonnançabilité d'un système.

Corollaire 6 *(***Test d'ordonnançabilité***) En appliquant l'algorithme d'ordonnancement précédent de* 0 *à* $s_{max} + T$*, il est possible de décider si un système avec contraintes de précédences et de périodicités est ordonnançable ou non.*

Preuve Comme on avait vu dans le théorème 14, un ordonnancement pour un système avec contraintes de précédences et de périodicités a une hyper-période T égale au PPCM de toutes les contraintes de précédence. Si l'algorithme d'ordonnancement trouve un ordonnancement de 0 à $s_{max} + T$, alors le système est ordonnançable. Au contraire, si l'algorithme d'ordonnancement ne trouve pas un ordonnancement de 0 à $s_{max} + T$, alors le système n'est pas ordonnançable. Le corollaire est prouvé $\boxdot$

Après avoir étudié l'ordonnancement et l'ordonnançabilité des systèmes d'opérations avec contraintes de précédences et de périodicités, on passe aux systèmes d'opérations avec contraintes de précédences et de latences.

2.2 Contraintes de précédences et de latences

Ce chapitre s'intéresse aux systèmes temps réel avec contraintes de précédences et de latences. Le premier sous-chapitre présente un modèle original pour ces systèmes. Ce modèle permet d'imposer des contraintes de précédence entre deux opérations quelconques et une contrainte de latence entre deux opérations quelconques. Il met en évidence la cohérence entre ces deux contraintes et la pertinence de la latence par rapport aux précédences et vice-versa. Le deuxième sous-chapitre fait une étude d'ordonnançabilité et le dernier sous-chapitre présente un algorithme optimal (s'il y a un ordonnancement, l'algorithme le trouvera).

2.2.1 Modèle

Ce chapitre présente le modèle utilisé pour décrire les systèmes temps réel avec contraintes de précédences et de latences. Notre définition de la latence généralise les contraintes relatives existant entre deux opérations devant s'exécuter dans un certain temps.

On a un système avec contraintes de précédence définies par un graphe orienté acyclique $\mathcal{G} = (\mathcal{V},\mathcal{E})$ où $\mathcal{V}$ est l'ensemble des opérations et $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ l'ensemble des arcs qui représentent des précédences entre les opérations. Le système est toujours un graphe infini obtenu par la répétition infinie d'un sous-graphe motif. Le graphe factorisé utilise les arcs ∞ pour factoriser les arcs inter-motif. Par exemple dans la figure 2.21 on a le graphe défactorisé auquel correspond le graphe factorisé de la figure 2.22.

Par rapport au modèle proposé pour les systèmes avec contraintes de précédences et de périodicités, la répétition du motif du graphe n'est pas accompagnée par aucune contrainte

temporelle entre deux répétitions consécutives d'une opération appartenant à deux motifs différents comme la contrainte de périodicité par exemple. Pour cette raison un ordonnancement d'un système avec contraintes de précédences et de latences a le motif qui est égale au motif du graphe et qui contient donc les répétitions du même indice.

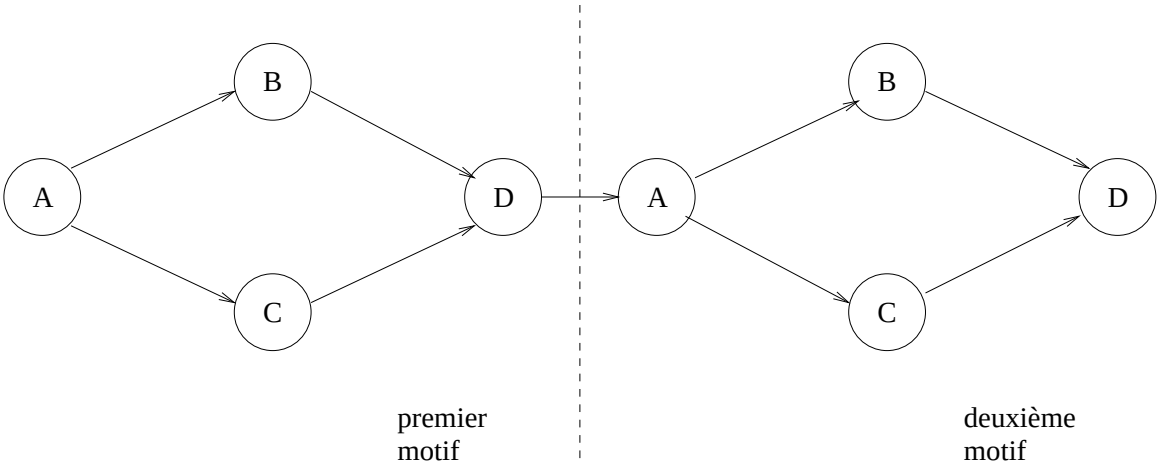

FIG. 2.21 – *Graphe défactorisé*

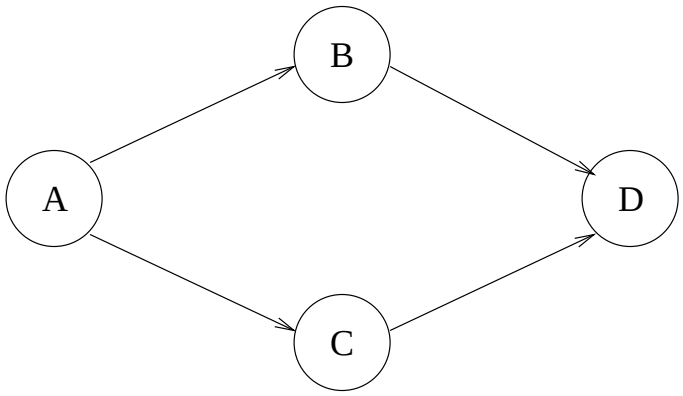

FIG. 2.22 – *Graphe factorisé*

Remarque 8 *On rappelle que pour les systèmes avec contraintes de précédences et de périodicités on ordonnance les opérations disponibles à partir de la date de fin de la dernière opération ordonnancée qui permet ensuite d'ordonnancer toutes les autres répétitions des opérations disponibles. En revanche, pour les systèmes avec contraintes de précédences et de latences on va ordonnancer les opérations disponibles à partir de la date de fin de la dernière opération ordonnancée. Par conséquent, on considère les ordonnancements tels que* $\forall A \in \mathcal{V}$,$\exists B \in \mathcal{V}$ *tel que* $s_A + C_A = s_B$.

Définition 5 *deux opérations différentes A et B appartenant au même motif, telles que $\exists P(A,B) \in \mathcal{P}$, ont une contrainte de latence $L_{AB} \in \mathbb{N}$ si elles sont ordonnancées telles que $s_B + C_B - s_A \leq L_{AB}$. Soit $\mathcal{L}$ l'ensemble de toutes les contraintes de latence définies pour un système.*

Remarque 9 *La condition qui impose sur A et sur B d'appartenir au même motif n'est pas restrictive. En effet si on veut imposer une contrainte de latence sur deux opérations A et B qui n'appartiennent pas au même motif, il suffit alors de choisir un nouveau motif (qui n'est finalement qu'une convention). Evidemmment on ne risque pas d'obtenir un motif de taille infinie puisque la durée d'une latence L_{AB} est supposée avoir une nature finie due au fait qu'elle répresente la limite du temps écoulé entre deux opérations (dans le cas contraire, la latence ne sera plus intéressante comme contrainte temps réel). On donne figure 2.23 un exemple où on veut définir une contrainte de latence $L(C,B)$, C et B appartenant à deux motifs consécutifs. Pour cela on choisit le motif qui est égal à deux anciens motifs consécutifs.*

Remarque 10 *En utilisant la définition 2 donnée au début du chapitre 2, un système d'opérations avec contraintes de précédences et de latences est ordonnançable s'il y a au moins un ordonnancement qui satisfait toutes ses contraintes de précédences et de latences. Trouver un tel ordonnancement est le problème que nous cherchons à résoudre dans ce chapitre.*

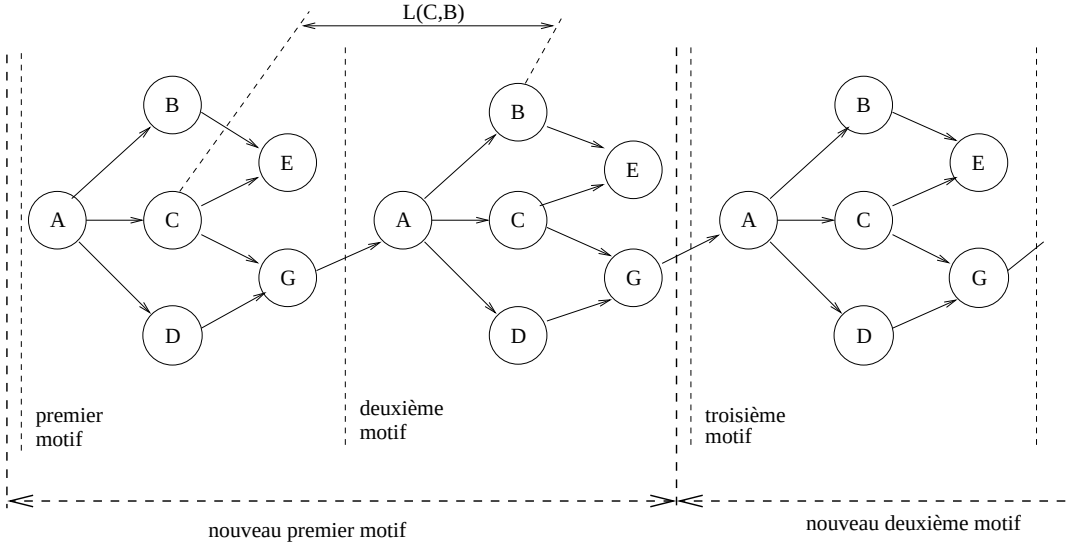

FIG. 2.23 – *Latence définie sur plusieurs motifs*

Comme on l'avait mentionné dans le chapitre 1.4, pour les systèmes réactifs une opération de sortie est une opération sans successeur dans le motif est obtenue en réaction à une opération d'entrée qui est une opération sans prédécesseur dans le motif, après l'exécution de plusieurs opérations liées par des contraintes de précédence. Imposer un délai entre une opération d'entrée et une opération de sortie revient à satisfaire une contrainte de latence

pour ces deux opérations. La définition 5 généralise la notion de latence pour deux opérations quelconques, qui ne sont pas nécessairement des opérations d'entrée et des opérations de sortie.

Les deux prochains chapitres utilisent ce modèle afin d'étudier l'ordonnancement et l'ordonnançabilité des systèmes temps réel avec contraintes de précédences et de latences.

2.2.2 Condition d'ordonnançabilité

On donne une condition nécessaire et suffisante pour qu'un système avec des contraintes de précédences et de latences soit ordonnançable. Puisque le graphe est un motif répété infiniment et que les contraintes de latence sont définies pour des opérations appartenant au même motif, il suffit d'étudier le motif pour obtenir une condition d'ordonnançabilité.

Les contraintes de latence sont définies sur des paires d'opérations liées par un chemin qui impose un ordre total d'ordonnancement sur cette paire d'opérations. Donc, si on veut ordonnancer un système d'opérations avec plusieurs contraintes de latence, il faut étudier l'ordre partiel résultant de la combinaison des n paires ayant des contraintes de latence. Pour cela on étudie les n paires deux à deux. En conséquence dans ce chapitre on présente d'abord les relations entre deux paires d'opérations du point de vue de l'ordonnancement et ensuite on passe à l'étude d'ordonnançabilité en prenant en compte les contraintes de latence.

Pour faciliter la lecture, par « opération appartenant à une paire (X,Y) » on désigne par la suite une opération appartenant à au moins un des chemins allant de l'opération X à l'opération Y de la paire considérée. De même, par « une paire (X,Y) contient une opération Z » on désigne que Z appartient à la paire (X,Y).

Pour deux opérations quelconques X et Y appartenant à un système d'opérations et un ordonnancement S de ce système, on appelle « durée d'ordonnancement », le temps écoulé entre le début de l'opération X et la fin de l'opération Y. On note ce temps $\delta_{XY}(S) = s_Y - s_X + C_Y$, où $s_X,s_Y \in S$. Sans perdre de généralité et afin que $\delta_{XY}(S)$ soit toujours positif, on considère que X est toujours ordonnancée avant Y. Comme on considère que les opérations disponibles sont ordonnancées à partir de la date de fin de la dernière opération ordonnancée (voir remarque 8), on a $\delta_{XY}(S) = \sum_{\forall C, s_X \leq s_C \leq s_Y} C_C$.

Théorème 15 *Soit $\mathcal{G} = (\mathcal{V},\mathcal{E})$ un graphe et (A,B) une contrainte de latence. Si le système est ordonnançable alors la somme des durées d'exécution des opérations appartenant à $\mathcal{M}(A,B)$ est inférieure ou égale à $L_{A,B}$.*

Preuve Si le système est ordonnançable alors il y a un ordonnancement S qui satisfait les contraintes du système. Donc pour l'ordonnancement S on a

$$\delta_{AB}(S) \leq L_{A,B} \tag{2.8}$$

Commes les opérations appartenant à $\mathcal{M}(A,B)$ doivent être ordonnancées entre A et B dû à l'ordre partiel alors on a $\sum_{C \in \mathcal{M}(A,B)} C_C \leq \delta_{AB}(S) \leq L_{A,B}$. Donc $\sum_{C \in \mathcal{M}(A,B)} C_C \leq L_{A,B}$. Le théorème est prouvé ⊡

Le théorème 15 donne une condition nécessaire pour qu'une contrainte de latence soit satisfaite. En imposant cette condition, on garantit qu'on peut toujours ordonnancer au moins les opérations appartenant à la contrainte de latence entre la première et la dernière opération de la contrainte.

Si parmi ces opérations il n'y en a aucune appartenant à une autre paire sur laquelle une contrainte de latence est imposée, alors cette condition est aussi suffisante (voir théorème 16). Cela est dû au fait que tous les ordonnancements satisfaisant l'ordre partiel défini par le graphe, pour lesquels on obtient le plus petit temps écoulé entre la première opération et la dernière opération de la contrainte de latence, contiennent seulement des opérations appartenant à la paire sur laquelle la contrainte est imposée. Tous ces ordonnancements conduisent donc au même ensemble d'opérations ordonnancées entre la première opération de la latence et la dernière opération de la latence, qu'on appelle « ensemble minimal correspondant à la contrainte de latence ».

Si au contraire parmi ces opérations il y a au moins une opération appartenant à une autre paire sur laquelle une contrainte de latence est imposée, alors la condition donnée dans le théorème 15 est seulement nécessaire (voir théorème 19). Cela est dû au fait que tous les ordonnancements, pour lesquels on obtient le plus petit temps écoulé entre la première opération et la dernière opération de la contrainte de latence, ne contiennent entre la première opération de la latence et la dernière opération de la latence que des opérations appartenant à la paire sur laquelle la contrainte est imposée. Tous ces ordonnancements conduisent à des ensembles différents d'opérations ordonnancées entre la première opération de la latence et la dernière opération de la latence, qu'on appelle ensembles minimaux correspondant à la contrainte de latence.

Une première conséquence de ces interprétations du théorème 15 est le corollaire 7 concernant le cas d'un système avec une seule contrainte de latence. On se trouve dans le cas où il n'y a aucune opération appartenant à une autre paire sur laquelle une contrainte de latence est imposée et on a donc une condition nécessaire et suffisante.

Corollaire 7 *Soit $\mathcal{G} = (\mathcal{V},\mathcal{E})$ un graphe et (A,B) la seule contrainte de latence imposée. Le système est ordonnançable si et seulement si la somme des durées d'exécution des opérations appartenant à $\mathcal{M}(A,B)$ est plus petite ou égale que $L_{A,B}$.*

Preuve S'il n'y a qu'une seule contrainte de latence alors l'étude d'ordonnançabilité consiste à considérer les ordonnancements qui satisfont l'ordre partiel et qui contiennent entre la première opération de la latence et la dernière opération de la latence seulement des opérations appartenant à la paire sur laquelle la contrainte est imposée. En effet, pour cet ordonnancement on obtient le plus petit temps écoulé entre la première opération de la latence et la dernière opération de la latence. En conséquence il suffit de comparer la valeur de la latence à la somme de durées d'exécution des opérations appartenant à la paire sur laquelle la contrainte est imposée. Le corollaire est prouvé $\boxdot$

Exemple 9 *Cet exemple illustre le cas d'une seule contrainte de latence. Pour le graphe donné dans la figure 2.24 on considère une seule contrainte de latence $L(A,D)$. On suppose que toutes les opérations sont de durées égales à 1. L'ensemble minimal $\{A,B,C,E,D\}$*

contient entre la première opération de la latence et la dernière opération de la latence seulement des opérations appartenant à la paire sur laquelle la contrainte est imposée. L'opération F n'appartient pas à cet ensemble. Le système est ordonnançable si et seulement si la contrainte de latence est supérieure ou égale à 5.

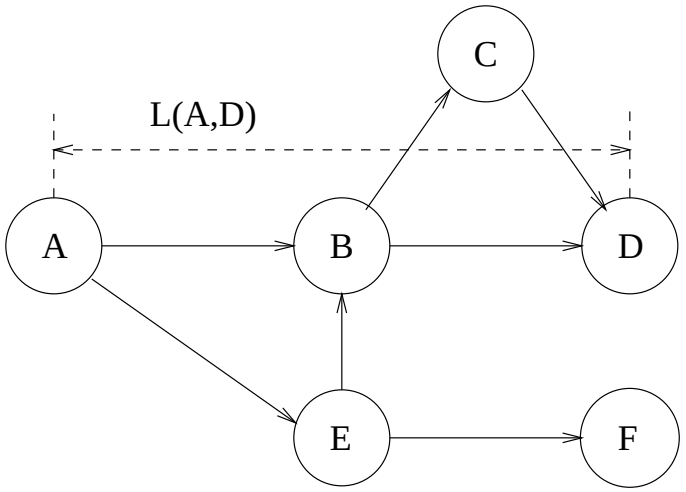

FIG. 2.24 – *Graphe illustrant la remarque 7 concernant une seule contrainte de latence*

Exemple 10 *Cet exemple illustre le cas où il y a plusieurs contraintes de latence. Pour le graphe donné dans la figure 2.25 on considère les contraintes de latence $L(A,C)$ et $L(J,L)$. Les ensembles minimaux pour ces contraintes contiennent seulement les opérations appartenant à la paire sur laquelle la contrainte a été imposée étant donné qu'aucune des deux contraintes de latence ne contient d'opérations appartenant à l'autre contrainte. Dans ce cas les deux ensembles minimaux sont uniques et on a pour la contrainte $L(A,C)$ l'ensemble minimal $\{A,B,C\}$ et pour la contrainte $L(J,L)$ l'ensemble minimal $\{J,K,L\}$.*

Toujours pour le graphe donné dans la figure 2.25 on considère les contraintes de latence $L(D,I)$ et $L(J,L)$. Les ensembles minimaux pour ces contraintes contiennent aussi des opérations appartenant à l'autre paire étant donné que chaque contrainte de latence contient des opérations appartenant à l'autre contrainte. Dans ce cas les ensembles minimaux ne sont pas uniques. Commes les contraintes de précédence (F,L) et (K,H) imposent que K et F soient ordonnancées avant L et H, alors soit F doit être ordonnancée entre les opérations de l'autre paire (J,L), soit K doit être ordonnancée entre les opérations de l'autre paire (D,I). On a donc les deux possibilités suivantes :

- *soit pour la contrainte $L(D,I)$ l'ensemble minimal est $\{D,F,H,I,J,K\}$ et pour la contrainte $L(J,L)$ l'ensemble minimal est $\{J,K,L,H,I\}$,*
- *soit pour la contrainte $L(D,I)$ l'ensemble minimal est $\{D,F,H,I,L\}$ et pour la contrainte $L(J,L)$ l'ensemble minimal est $\{J,K,L,D,F\}$.*

Faire une étude d'ordonnançabilité dans le cas avec contraintes de précédences et de latences revient donc à déterminer, pour chaque contrainte de latence, l'ensemble minimal d'opérations, c'est-à-dire les opérations qui pourraient être ordonnancées à l'extérieur de la contrainte de latence et qui si elles sont ordonnancées à l'intérieur de la latence ne feraient qu'allonger inutilement le temps écoulé entre la première et la dernière opération de la contrainte de latence. Par l'extérieur d'une contrainte de latence on désigne, pour un ordonnancement quelconque, les opérations ordonnancées avant la première opération ou après la dernière opération de la contrainte de latence. Par l'intérieur d'une contrainte de latence on désigne, pour un ordonnancement quelconque, les opérations ordonnancées après la première opération et avant la dernière opération de la contrainte de latence.

Relations entre deux paires d'opérations du point de vue de l'ordonnancement

On prend deux paires d'opérations et on étudie les cas où elles sont liées, ou non, par des chemins. Les chemins qui lient deux paires sont ceux qui vont d'une opération appartenant à une des paires vers une opération appartenant à l'autre paire.

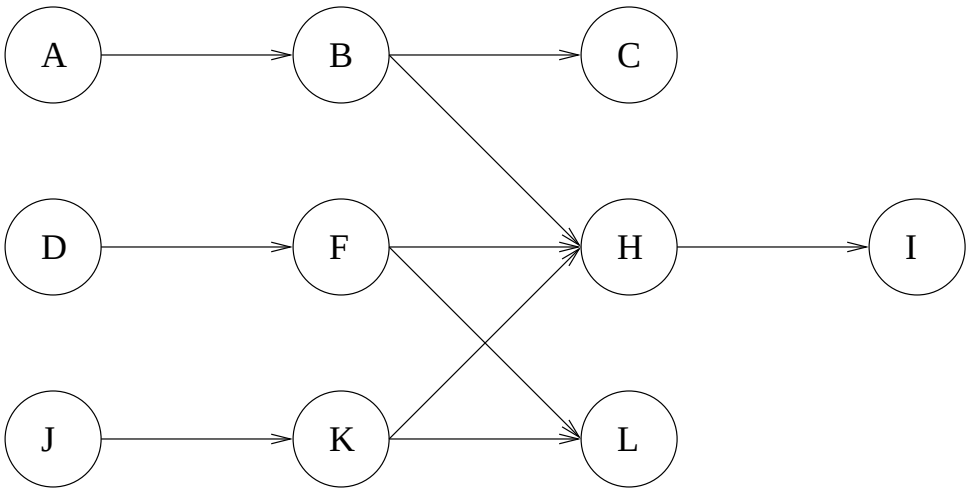

FIG. 2.25 – *Relations entre paires d'opérations du point de vue de l'ordonnancement*

Ainsi on a 2^2 combinaisons possibles de deux paires pour lesquelles il y a des chemins d'une opération d'une paire vers une opération de l'autre paire et/ou vice-versa :

1. il n'y a aucun chemin à partir d'une opération d'une des paires vers une opération de l'autre paire (dans la figure 2.25, les paires (A,C) et (J,L)). Cette relation est appelée ||;
2. il y a au moins un chemin d'une paire vers l'autre paire et il n'y a aucun chemin de la dernière paire vers la première ou vice-versa (dans la figure 2.25, les paires (A,C) et (D,I)). Cette relation est appelée Z. Ce cas couvre deux combinaisons possibles puisque pour deux paires (X,Y) et (V,W) il peut y avoir soit au moins un chemin

d'une opération de (X,Y) vers une opération de (V,W) soit au moins un chemin d'une opération de (V,W) vers une opération de (X,Y). Les deux possibilités correspondent à la définition de la relation Z;

3. il y a au moins un chemin d'une paire vers l'autre paire et vice-versa (dans la figure 2.25, les paires (D,I) et (J,L)). Cette relation est appelée X.

Dans le cas de la relation || de la figure 2.25, aucune précédence n'impose sur une opération appartenant à $\mathcal{M}(A,C)$ d'être ordonnancée avant ou après toutes les opérations appartenant à $\mathcal{M}(J,L)$. Cela signifie que l'ordre partiel défini par les précédences n'impose aucun ordre entre les opérations appartenant à $\mathcal{M}(A,C)$ et les opérations appartenant à $\mathcal{M}(J,L)$. Par exemple les ordonnancements dans lesquels A, B et C sont ordonnancées avant J, K et L, ou bien J, K et L sont ordonnancées avant A, B et C, satisfont l'ordre partiel.

Dans le cas de la relation Z, les précédences imposent sur les opérations appartenant à $\mathcal{M}(A,B)$ d'être ordonnancées avant les opérations appartenant à $\mathcal{M}(H,I)$. Cependant elles n'imposent aucun ordre entre les opérations appartenant à $\mathcal{M}(B,C)\backslash\{B\}$ et les opérations appartenant à $\mathcal{M}(D,H)\backslash\{H\}$. Par exemple A, B et C peuvent être ordonnancées avant toutes les opérations appartenant à $\mathcal{M}(D,I)$, et de la même manière D, F, H et I peuvent être ordonnancées après toutes les opérations appartenant à $\mathcal{M}(A,C)$. Donc de manière générale, du point de vue de l'ordre partiel toutes les opérations appartenant à $\mathcal{M}(A,C)$ peuvent être ordonnancées avant toutes les opérations appartenant à $\mathcal{M}(D,I)$, et réciproquement.

Dans le cas de la relation X, les précédences imposent sur les opérations appartenant à $\mathcal{M}(D,F)$ d'être ordonnancées avant les opérations appartenant à $\mathcal{M}(L,L)$ et aux opérations appartenant à $\mathcal{M}(H,I)$ d'être ordonnancées après les opérations appartenant à $\mathcal{M}(J,K)$. Cependant elles n'imposent aucun ordre entre les opérations appartenant à $\mathcal{M}(D,F)$ et les opérations appartenant à $\mathcal{M}(J,K)$. Par exemple D et F peuvent être ordonnancées avant ou après J et K, et H et I doivent être ordonnancées après J et K, et réciproquement.

En prenant en compte ce qui précède sur les relations entres les paires d'opérations, on étudie la condition d'ordonnançabilité pour les contraintes de latence définies sur des paires en relation Z et/ou en relation ||, et ensuite on étudie la condition d'ordonnançabilité pour les contraintes de latence définies sur des paires en relation X. Enfin, on donne une condition générale d'ordonnançabilité pour des systèmes avec des contraintes de latence définies sur des paires en relation ||, Z et X.

Ordonnançabilité pour les contraintes de latence définies sur des paires en relation Z et/ou en relation ||

On commence par repérer les paires sur lesquelles des contraintes de latence ont été définies et on s'assure qu'il y a seulement des paires en relation Z et/ou en relation ||. Par exemple, dans la figure 2.26 on a trois contraintes de latence $L(A,C,)$, $L(D,I)$ et $L(J,L)$, définies sur des paires en relation Z et/ou en relation ||.

En utilisant les relations entre les paires d'opérations vues précédemment seule la relation Z impose un ordre partiel sur les paires tandis que la relation || n'impose aucun ordre. Par

exemple, dans la figure 2.26 l'ordre partiel impose seulement à A et à B de s'ordonnancer avant H et I.

L'étude d'ordonnançabilité cherche parmi tous les ordonnancements possibles ceux conduisant aux ensembles minimaux correspondant aux latence. Le lemme 2 permet de déterminer ces ordonnancements et le lemme 3 prouve que pour les paires en relation Z et les paires en relation || ces ensembles minimaux sont les mêmes. En conséquence, même si du point de vu de l'ordre partiel, les deux cas ne sont pas équivalents, du point de vue de l'ordonnaçabilité ils sont équivalents. Ceci implique qu'on peut donner une condition commune d'ordonnançabilité pour les deux relations. Cette condition est donnée dans le théorème 16 qui finit ce sous-chapitre.

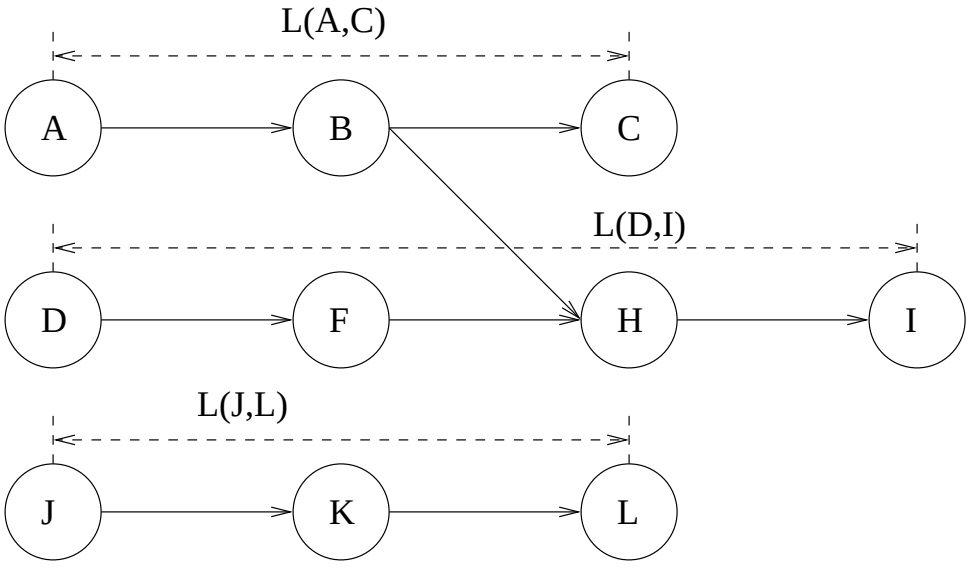

FIG. 2.26 – *Contraintes de latence*

Lemme 2 *Soient A,$B \in \mathcal{V}$ deux opérations appartenant à un système défini par le graphe $\mathcal{G} = (\mathcal{V},\mathcal{E})$ avec $P(A,B) \in \mathcal{P}$. On note par $S \in \mathcal{S}$ un ordonnancement tel que $\forall H \notin \mathcal{M}(A,B)$, soit on a $s_H + C_H < s_A + C_A$, soit on a $s_B + C_B < s_H + C_H$. Si $\mathcal{S}'$ est l'ensemble des ordonnancements S' tel que $\exists H \notin \mathcal{M}(A,B)$ avec $s'_A + C_A < s'_H + C_H < s'_B + C_B$, alors*

$$\delta_{AB}(S) \leq \delta_{AB}(S') \ , \forall S' \in \mathcal{S}' \tag{2.9}$$

Preuve Puisque S est un ordonnancement tel que $\forall H \notin \mathcal{M}(A,B)$, soit on a $s_H + C_H < s_A + C_A$, soit on a $s_B + C_B < s_H + C_H$, alors on ordonnance de s_A à s_B seulement les opérations appartenant à $\mathcal{M}(A,B)$ et on a

$$\delta_{AB}(S) = \sum_{I \in \mathcal{M}(A,B)} C_I \tag{2.10}$$

Pour un ordonnancement quelconque $S' \in \mathcal{S}'$, soit $N = \{H \in \mathcal{V}$ tel que $H \notin \mathcal{M}(A,B)$ et $s'_A + C_A < s'_H + C_H < s'_B + C_B\}$. On a

$$\delta_{AB}(S') = \sum_{I \in \mathcal{M}(A,B)} C_I + \sum_{H \in N} C_H, \forall S' \in \mathcal{S}' \tag{2.11}$$

En soustrayant les équations (2.10) et (2.11), on obtient $\delta_{AB}(S') - \delta_{AB}(S) = \sum_{H \in N} C_H \geq 0$, $\forall S' \in \mathcal{S}$. Finalement on a $\delta_{AB}(S) \leq \delta_{AB}(S')$, $\forall S' \in \mathcal{S}'$. Le lemme est prouvé $\boxdot$

Dans le lemme 2 l'inégalité 2.9 revient à dire que $\delta_{AB}(S) = min_{\forall S'} \delta_{AB}(S')$, c'est-à-dire que tout ordonnancement appartenant à $\mathcal{S}$ a la plus petite durée d'ordonnancement entre A et B. Cela implique que dans le cas où on impose une contrainte de latence sur la paire (A,B), tout ordonnancement appartenant à $\mathcal{S}$ conduit à l'ensemble minimal correspondant à la contrainte de latence.

Lemme 3 *Pour quatre opérations A,B,C et $D \in \mathcal{V}$, si (A,B) || (C,D) ou si (A,B) Z (C,D) alors on a $\mathcal{M}(A,B) \bigcap \mathcal{M}(C,D) = \emptyset$.*

Preuve On utilise un raisonnement par l'absurde pour prouver que si (A,B) et (C,D) sont en relation Z alors $\mathcal{M}(A,B) \bigcap \mathcal{M}(C,D) = \emptyset$.

On suppose que $\exists F \in \mathcal{M}(A,B) \bigcap \mathcal{M}(C,D)$. Puisque $F \in \mathcal{M}(A,B)$, respectivement $F \in \mathcal{M}(C,D)$, on a

$$P(A,F) \in \mathcal{P} \tag{2.12}$$

et

$$P(F,B) \in \mathcal{P} \tag{2.13}$$

respectivement

$$P(C,F) \in \mathcal{P} \tag{2.14}$$

et

$$P(F,D) \in \mathcal{P} \tag{2.15}$$

A partir des relations (2.12) et (2.15), respectivement, des relations (2.13) et (2.14), on obtient $P(A,D) \in \mathcal{P}$ et $P(C,B) \in \mathcal{P}$. Ceci est en contradiction avec la définition de la relation Z, qui dit qu'il y a au moins un chemin d'une paire à l'autre paire et qu'il n'y a pas de chemin à partir de la dernière paire à la première. La supposition faite qu'il y a $F \in \mathcal{M}(A,B) \bigcap \mathcal{M}(C,D)$ est fausse, donc on a $\mathcal{M}(A,B) \bigcap \mathcal{M}(C,D) = \emptyset$.

On utilise le même raisonement afin de prouver que si (A,B) || (C,D) alors $\mathcal{M}(A,B) \bigcap \mathcal{M}(C,D) = \emptyset$. Le lemme est prouvé $\boxdot$

Le lemme 3 prouve qu'une contrainte de latence $L(A,B)$ en relation || ou en relation Z avec une autre contrainte de latence $L(C,D)$ ne contient aucune opération appartenant à cette dernière ($\mathcal{M}(A,B) \bigcap \mathcal{M}(C,D) = \emptyset$). Cela implique que chaque ensemble minimal correspondant à la contrainte de latence contient seulement les opérations appartenant à cette contrainte, donc dans les deux cas l'ensemble minimal correspondant à chaque contrainte de latence est le même que ce soit pour le || ou le Z.

Théorème 16 *On a un système d'opérations défini par un graphe $\mathcal{G} = (\mathcal{V},\mathcal{E})$ tel que chaque couple de deux paires sur lesquelles on a défini des contraintes de latence sont soit en relation* || *, soit en relation* Z*. Le système d'opérations est ordonnançable si et seulement si*

$$\sum_{H \in \mathcal{M}(A,B)} C_H \leq L_{AB}$$

pour tous A et B ayant une contrainte de latence.

Preuve Le lemme 2 montre que, pour deux opérations A et B liées par au moins un chemin, en ordonnançant entre A et B seulement des opérations qui appartiennent à $\mathcal{M}(A,B)$, on va diminuer la valeur de $\delta_{AB}(.)$. Ceci signifie que si un ordonnancement, pour lequel seulement les opérations appartenant à $\mathcal{M}(A,B)$ sont ordonnancées entre A et B, ne satisfait pas la contrainte de latence L_{AB} pour A et B, alors aucun autre ordonnancement ne le fera. De plus, puisque $\mathcal{M}(A,B) \bigcap \mathcal{M}(C,D) = \emptyset$, pour toutes les paires qui sont en relation || ou en relation Z (le lemme 3), on va considérer à partir de maintenant seulement des ordonnancements pour lequel seulement les opérations appartenant à $\mathcal{M}(A,B)$ sont ordonnancées entre A et B .

Un ordonnancement S satisfait toutes les contraintes de latence si et seulement $\delta_{AB} \leq L_{AB}$, $\forall A$ et $\forall B$ ayant une contrainte de latence (définition 5). Puisque $\forall H \notin \mathcal{M}(A,B)$, soit on a $s_H + C_H < s_A + C_A$, soit on a $s_B + C_B < s_H + C_H$, alors un ordonnancement S satisfait toutes les contraintes de latence si et seulement si $\delta_{AB} = \sum_{H \in \mathcal{M}(A,B)} C_H \leq L_{AB}$, pour toutes les opérations A et B ayant une contrainte de latence. Le théorème est prouvé $\boxdot$

Remarque 11 *Dans le cas où les contraintes de latence sont en relation* || *ou en relation* Z *l'ensemble minimal correspondant à une contrainte de latence est égal à l'ensemble des opérations appartenant à cette contrainte. La première partie de l'exemple 10 illustre un tel ensemble.*

Exemple 11 *Cet exemple illustre la condition d'ordonnançabilité pour un système d'opérations avec toutes les contraintes de latence en relation* || *ou en relation* Z*. Soit le graphe donné dans la figure 2.27 avec des contraintes de latence $L(A,C)$, $L(D,F)$, $L(H,I)$ et, respectivement, $L(J,L)$ définies sur les paires (A,C), (D,F), (H,I) et, respectivement, (J,L). Toutes les contraintes de latence sont égales à 3 et $C_A = 1, \forall A \in \mathcal{V}$. On remarque que toutes les paires sont soit en relation* ||*, soit en relation* Z*. La condition d'ordonnançabilité implique qu'il suffit que les latence soient plus grandes que la somme des durées d'exécution des opérations. On suppose que cette condition est satisfaite alors il y a au moins un ordonnancement qui satisfait toutes les contraintes du système. Dans la figure 2.28 on donne un tel ordonnancement.*

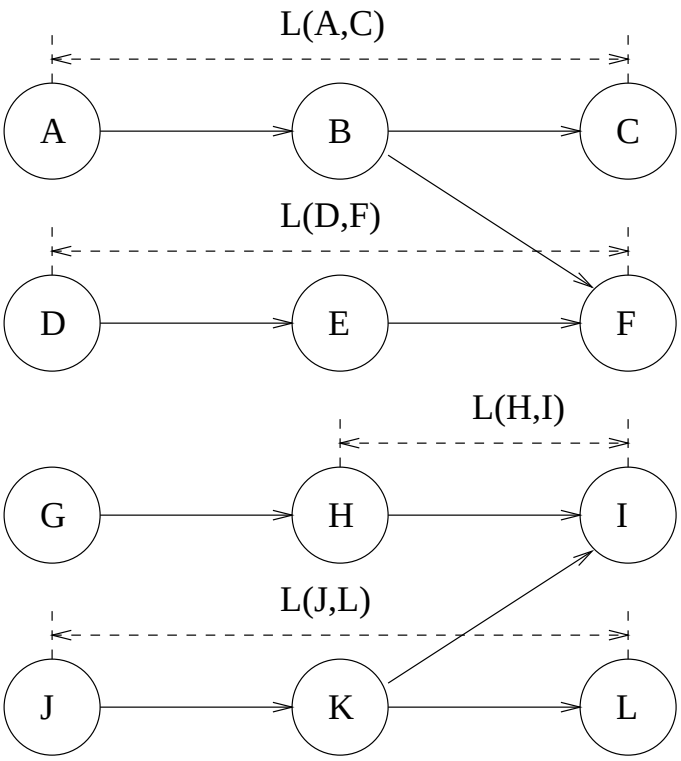

FIG. 2.27 – *Système d'opérations avec contraintes de latence en relation* || *ou en relation* Z

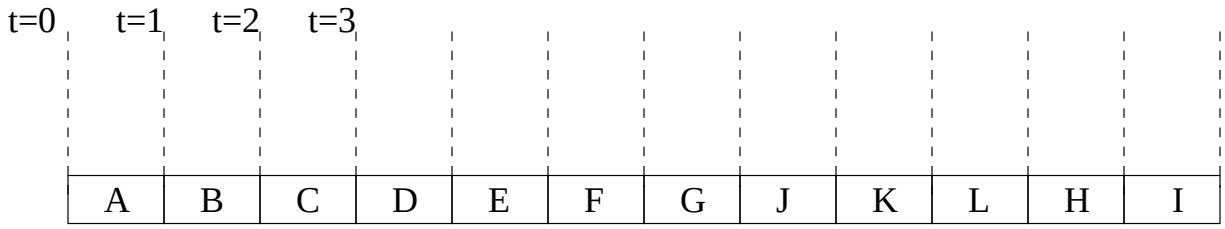

FIG. 2.28 – *Ordonnancement satisfaisant toutes les contraintes de latence*

Ordonnançabilité pour les contraintes de latence définies sur des paires en relation X

Pour présenter les résultats dans le cas des systèmes avec des contraintes de latence sur des paires en relation X on introduit deux nouvelles notations.

Soient $(A,B),(C,D) \in \mathcal{V} \times \mathcal{V}$ avec $P(A,B),P(C,D) \in \mathcal{P}$ deux paires quelconques en relation X. Chacune des deux paires a des opérations qui doivent s'ordonnancer afin que les opérations de l'autre paire puissent s'ordonnancer aussi. Comme dans le cas avec des paires en relation || et/ou en relation Z, l'étude d'ordonnançabilité pour des paires en relation X s'appuie sur les ensembles minimaux correspondant aux contraintes de latence.

Pour deux paires en relation X, ft désigne l'ensemble des opérations qui sont premières, sur tous les chemins reliant la première opération d'une paire et ces opérations, relativement à l'existence d'au moins un chemin partant de l'autre paire et aboutissant à ces opérations.

On note par $ft_{CD}(A,B)$ l'ensemble des opérations $E \in \mathcal{M}(A,B)$ pour lequel $\exists F \in \mathcal{M}(C,D)$ tel que $P(F,E) \in \mathcal{P}$ et $\nexists G \in \mathcal{M}(A,E)$ pour lequel $\exists F \in \mathcal{M}(C,D)$ tel que $P(F,G) \in \mathcal{P}$. Ceci signifie que $ft_{CD}(A,B)$ est l'ensemble des opérations E appartenant à un des chemins de A à B tel qu'il y a un chemin vers E à partir d'une opération appartenant à un des chemins de C à D et il n'y a pas d'opération G appartenant à un des chemins de A à E tel qu'il y a un chemin vers G à partir d'une opération appartenant à un des chemins de C à D. On note par $\Gamma^{-}_{CD}(A,B)$ l'ensemble des prédécesseurs de $ft_{CD}(A,B)$, c'est-à-dire l'ensemble des opérations $E \in \mathcal{V}$ pour lequel il y a une opération $F \in ft_{CD}(A,B)$ tel que $(E,F) \in \mathcal{E}$.

Pour deux paires en relation X, lt désigne l'ensemble des opérations qui sont dernières, sur tous les chemins reliant ces opérations et la dernière opération d'une paire, relativement à l'existence d'au moins un chemin partant de ces opérations et aboutissant à l'autre paire.

On note par $lt_{CD}(A,B)$ l'ensemble des opérations $E \in \mathcal{M}(A,B)$ pour lequel $\exists F \in \mathcal{M}(C,D)$ tel que $P(E,F) \in \mathcal{P}$ et $\nexists G \in \mathcal{M}(E,B)$ pour lequel $\exists F \in \mathcal{M}(C,D)$ tel que $P(G,F) \in \mathcal{P}$. Ceci signifie que $lt_{CD}(A,B)$ est l'ensemble des opérations E appartenant à un des chemins de A à B tel qu'il y a un chemin de E vers une des opérations appartenant à un des chemins de C à D et il n'y a pas d'opération G appartenant à un des chemins de E à B tel qu'il y a un chemin de G vers une des opérations appartenant à un des chemins de C à D. On note par $\Gamma^{+}_{CD}(A,B)$ l'ensemble des successeurs de $lt_{CD}(A,B)$, c'est-à-dire l'ensemble des opérations $E \in \mathcal{V}$ pour lequel il y a une opération $F \in lt_{CD}(A,B)$ tel que $(F,E) \in \mathcal{E}$.

Exemple 12 *Pour illustrer les deux notations, on donne l'exemple suivant. Dans la figure 2.29, pour* (A,B) X (C,D), *on a* $ft_{CD}(A,B) = \{F,G\}$, $lt_{CD}(A,B) = \{G\}$, $ft_{AB}(C,D) = \{G\}$ *et* $lt_{AB}(C,D) = \{G,I\}$.

L'étude d'ordonnançabilité s'appuie de nouveau sur l'ensemble minimal d'opérations appartenant à une contrainte de latence L_1 qui doivent s'ordonnancer pour que la dernière opération de l'autre contrainte de latence L_2 puisse le faire quand on a L_1 X L_2. Pour cela on étudie la position dans l'ordonnancement des opérations appartenant à L_1 qui ne font qu'allonger la durée d'ordonnancement entre la première opération de L_2 et la dernière opération de L_2 sans modifier la durée d'ordonnancement entre la première opération de L_1

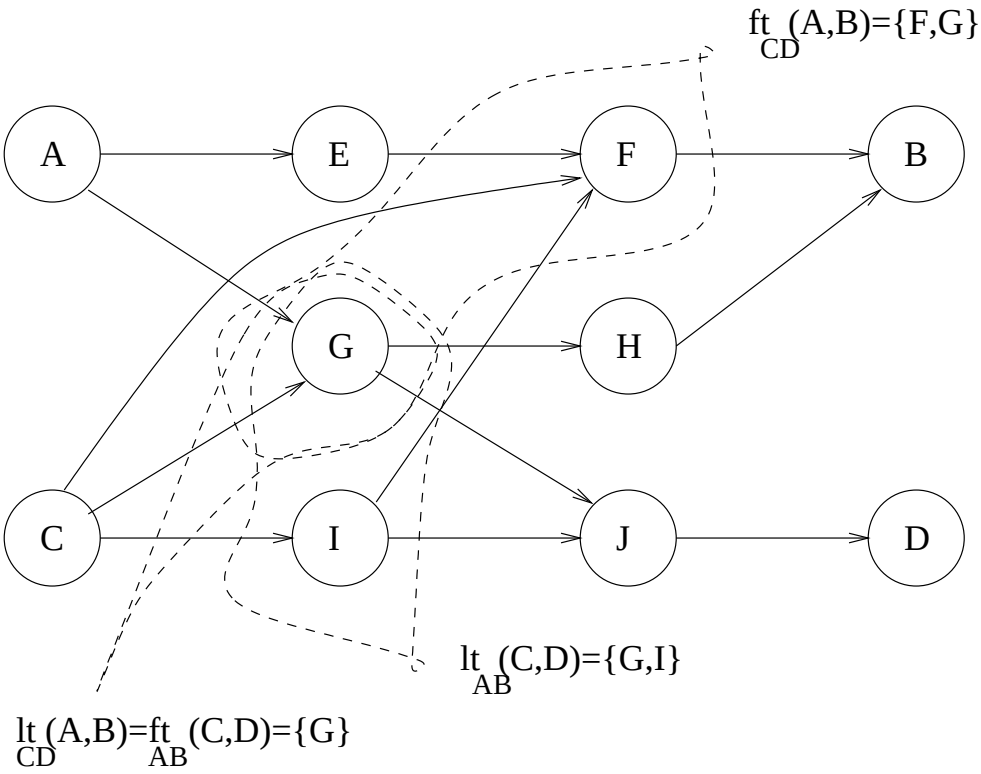

FIG. 2.29 – *Graphe illustrant les notions de ft et de lt*

et la dernière opération de L_1, une fois que les premières opérations ou les dernières opérations des deux contraintes ont été ordonnancées. Les opérations appartenant à L_1 qui ne font qu'allonger la durée d'ordonnancement entre la première et la dernière opération de L_2 une fois que la première opération de L_1 a été ordonnancée correspondent à l'ensemble $\Gamma^-_{L_2}(L_1)$. Les opérations appartenant à L_1 qui ne font qu'allonger la durée d'ordonnancement entre la première et la dernière opération de L_2 une fois que la dernière opération de L_2 a été ordonnancée correspondent à l'ensemble $\Gamma^+_{L_2}(L_1)$. Dans ce cas l'ensemble minimal de L_1 est égal à $\mathcal{M}(L_1) \bigcup \{\mathcal{M}(L_2) \backslash \Gamma^+_{L_1}(L_2)\}$. Du même, l'ensemble minimal de L_2 est égal à $\mathcal{M}(L_2) \bigcup \{\mathcal{M}(L_1) \backslash \Gamma^-_{L_2}(L_1)\}$.

Exemple 13 *Dans la figure 2.30, on considère que L_1 est la contrainte de latence sur la paire (A,B) et que L_2 est la contrainte de latence sur la paire (C,D). Si on suppose qu'on ordonnance A avant C, alors quelque soit la position dans l'ordonnancement de l'opération $E \in lt_{CD}(A,B)$ par rapport à C, le temps écoulé entre A et B reste le même. Toujours dans le cas où on suppose qu'on ordonnance A avant C, le changement de la position de E par rapport à C change le temps écoulé entre C et D. Si on suppose qu'on ordonnance E après C on ne fait qu'allonger la durée d'ordonnancement entre C et D.*

De la même manière, dans la figure 2.31, on considère que L_1 est la contrainte de latence sur la paire (A,B) et que L_2 est la contrainte de latence sur la paire (C,D). Si on suppose qu'on ordonnance B avant D, alors quelque soit la position dans l'ordonnancement de l'opération $F \in ft_{AB}(C,D)$, par rapport à B, le temps écoulé entre C et D reste le même. Toujours dans le cas où on suppose qu'on ordonnance B avant D, le changement de la position de F par rapport à B change le temps écoulé entre A et B. Si on ordonnance F avant B on ne fait qu'allonger la durée d'ordonnancement entre A et B.

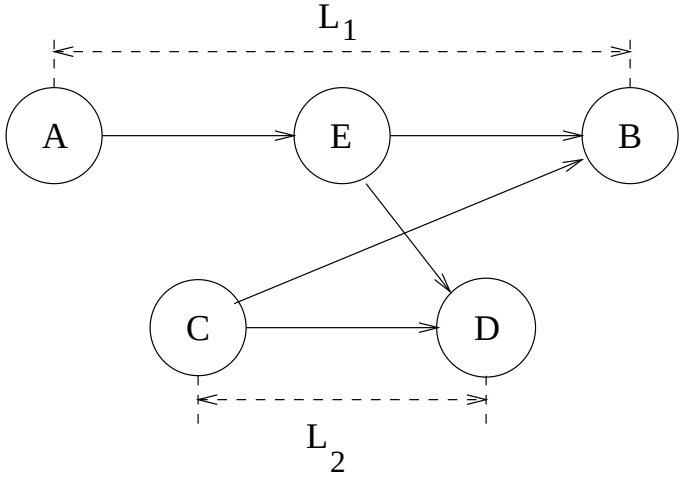

FIG. 2.30 – *Opérations se trouvant au début d'une contrainte de latence qui ne font qu'allonger la durée d'ordonnancement pour l'autre contrainte de latence*

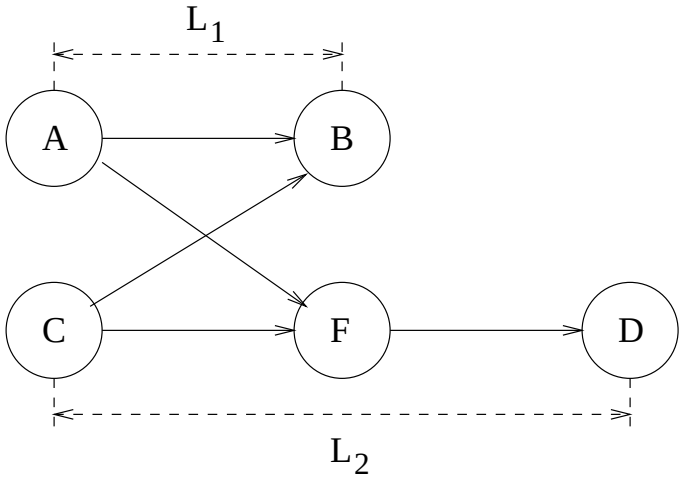

FIG. 2.31 – *Opérations se trouvant à la fin d'une contrainte de latence qui ne font qu'allonger la durée d'ordonnancement pour l'autre contrainte de latence*

Théorème 17 *On a un système défini par un graphe $\mathcal{G} = (\mathcal{V},\mathcal{E})$ tel qu'il y a deux contraintes de latence imposées sur les paires (A,B) et (C,D), où (A,B) X (C,D) et $L_{AB} = \sum_{H\in\mathcal{M}(\mathcal{A},\mathcal{B})} C_H$. Le système est ordonnançable si et seulement si $L_{CD} \geq \sum_{H\in\mathcal{M}(A,B)\bigcup\mathcal{M}(C,D)} C_H$.*

Preuve Puisque $L_{AB} = \sum_{H\in\mathcal{M}(\mathcal{A},\mathcal{B})} C_H$ alors entre l'opération A et l'opération B on peut ordonnancer seulement des opérations appartenant à $\mathcal{M}(A,B)$. Donc aucune opération appartenant à $\mathcal{M}(C,D)$ et n'appartenant pas à $\mathcal{M}(A,B)$ ne peut s'ordonnancer entre A et B.

En conséquence dans le cas où C et D n'appartiennent pas à $\mathcal{M}(A,B)$, alors ces opérations doivent être ordonnancées avant A et après B, ce qui implique que toutes les opérations appartenant à $\mathcal{M}(A,B)$ seront ordonnancées entre C et D. On a donc $L_{CD} \geq \sum_{H\in\mathcal{M}(A,B)\bigcup\mathcal{M}(C,D)} C_H$. Le théorème est prouvé ⊡

Remarque 12 *Dans ce théorème on ne prend pas en compte le cas trivial où C et D appartiennent à $\mathcal{M}(A,B)$. En effet si C et D appartenaient à $\mathcal{M}(A,B)$, alors toutes les opérations appartenant à $\mathcal{M}(C,D)$ appartiennent aussi à $\mathcal{M}(A,B)$, donc la contrainte de latence imposée sur la paire (A,B) doit être au moins plus grande ou égale à la somme des durées d'exécution des opérations appartenant à $\mathcal{M}(A,B)$.*

Remarque 13 *Pour le cas traité par le théorème 17 l'ensemble minimal de la contrainte de latence (A,B) est égale à l'ensemble des opérations appartenant à la contrainte de latence et l'ensemble minimal de la contrainte de latence (C,D) est égale à l'ensemble contenant toutes les opérations appartenant à ces deux contraintes de latence. Par exemple dans la figure 2.25 si on considère que la contrainte de latence $L(D,I)$ est égale à la somme des durées d'exécution des opérations appartenant à $\mathcal{M}(D,I)$ alors l'ensemble minimal de la contrainte (D,I) est égale à $\{D,F,H,I\}$ et l'ensemble minimal de la contrainte (J,L) est égale à $\{J,K,L,D,F,H,I\}$*

Les deux propriétés illustrées dans la exemple 13 concernant la position des premières et des dernières opérations des deux contraintes dans l'ordonnancement sont prouvées par les lemmes 4 et 5. Le théorème 19 qui les suit utilise ces deux résultats pour prouver la condition d'ordonnançabilité dans le cas où toutes les contraintes de latence sont en relation X.

Lemme 4 *Soit un système défini par un graphe $\mathcal{G} = (\mathcal{V},\mathcal{E})$ tel que les paires sur lesquelles il y a des contraintes de latence définies sont en relation* X. *Pour (A,B) X (C,D) et un ordonnancement quelconque $S \in \mathcal{S}$ avec $s_A \leq s_C$, si toutes les opérations $E \in \bigcup_{H \in \Gamma^-_{CD}(A,B)} \mathcal{M}(A,H)$ sont ordonnancées entre s_A et s_C, alors $\delta_{AB}(S)$ n'est pas modifié par rapport au cas où au moins une opération $E \in \bigcup_{H \in \Gamma^-_{CD}(A,B)} \mathcal{M}(A,H)$ est ordonnancée après s_C. Si au moins une opération $E \in \bigcup_{H \in \Gamma^-_{CD}(A,B)} \mathcal{M}(A,H)$ est ordonnancée après s_C, alors $\delta_{CD}(S)$ augmente par rapport au cas où toutes les opérations $E \in \bigcup_{H \in \Gamma^-_{CD}(A,B)} \mathcal{M}(A,H)$ sont ordonnancées entre s_A et s_C.*

Preuve Puisque $E \in \bigcup_{H \in \Gamma^-_{CD}(A,B)} \mathcal{M}(A,H) \subset \mathcal{M}(A,B)$, E est toujours ordonnancée entre s_A et sa position dans l'ordonnancement ne modifie pas $\delta_{AB}(S)$.

Puisque $E \in \bigcup_{H \in \Gamma^-_{CD}(A,B)} \mathcal{M}(A,H)$, il n'y a pas d'opération $F \in \mathcal{M}(C,D)$ qui doit être ordonnancée avant E. Donc, E peut être ordonnancée après C. Si on ordonnance une opération quelconque $E \in \bigcup_{H \in \Gamma^-_{CD}(A,B)} \mathcal{M}(A,H)$ après C, il est évident que pour un ordonnancement quelconque S, $\delta_{CD}(S)$ est augmenté avec la somme des durées d'exécution de toutes ces opérations E. Le lemme est prouvé $\boxdot$

Lemme 5 *On a un système défini par un graphe $\mathcal{G} = (\mathcal{V},\mathcal{E})$ tel que toutes les paires sur lesquelles il y a des contraintes de latence définies sont en relation* X. *Pour (A,B) X (C,D) et un ordonnancement quelconque $S \in \mathcal{S}$ avec $s_D \leq s_B$, si toutes les opérations $E \in \bigcup_{H \in \Gamma^+_{CD}(A,B)} \mathcal{M}(H,B)$ sont ordonnancées entre s_D et s_B, alors $\delta_{AB}(S)$ n'est pas modifiée par rapport au cas où au moins une opération $E \in \bigcup_{H \in \Gamma^+_{CD}(A,B)} \mathcal{M}(H,B)$ est ordonnancée avant s_D. Si au moins une opération $E \in \bigcup_{H \in \Gamma^+_{CD}(A,B)} \mathcal{M}(H,B)$ est ordonnancée avant s_D, alors $\delta_{CD}(S)$ augmente par rapport au cas où toutes les opérations $E \in \bigcup_{H \in \Gamma^+_{CD}(A,B)} \mathcal{M}(H,B)$ sont ordonnancées entre s_D et s_B.*

Preuve Puisque $E \in \bigcup_{H \in \Gamma^+_{CD}(A,B)} \mathcal{M}(H,B) \subset \mathcal{M}(A,B)$, E est toujours ordonnancée entre s_A et s_B, donc $\delta_{AB}(S)$ va toujours contenir la durée d'exécution de E et la position de E dans l'ordonnancement ne modifie pas $\delta_{AB}(S)$.

Puisque $E \in \bigcup_{H \in \Gamma^+_{CD}(A,B)} \mathcal{M}(H,B)$, il n'y a pas d'opération $F \in \mathcal{M}(C,D)$ qui doit être ordonnancée après E. Donc D peut être ordonnancée avant toutes les opérations $E \in \bigcup_{H \in \Gamma^+_{CD}(A,B)} \mathcal{M}(H,B)$. Si on ordonnance une opération quelconque $E \in \bigcup_{H \in \Gamma^+_{CD}(A,B)} \mathcal{M}(H,B)$ avant D, il est évident que pour un ordonnancement quelconque S, $\delta_{CD}(S)$ est augmentée avec la somme des durées d'exécution des ces opérations E. Le lemme est prouvé ⊡

Théorème 18 *On a un système défini par un graphe $\mathcal{G} = (\mathcal{V},\mathcal{E})$ tel qu'il y a deux contraintes de latence imposées sur les paires (A,B) et (C,D), où (A,B) X (C,D). Les deux contraintes de latence ne sont pas égales à la somme des durées d'exécution des opérations appartenant à la contrainte. Le système est ordonnançable si et seulement si*

$$\begin{cases} L_{AB} \geq \sum_{H \in \mathcal{M}(A,B)} C_H + \sum_{H \in \bigcup_{E \in lt_{AB}(C,D)} \mathcal{M}(C,E)} C_H \\ L_{CD} \geq \sum_{H \in \mathcal{M}(C,D)} C_H + \sum_{H \in \bigcup_{E \in ft_{CD}(A,B)} \mathcal{M}(E,B)} C_H \end{cases} \tag{2.16}$$

ou vice versa.

Preuve La preuve est faite par double implication.

On prouve d'abord que si le système est ordonnançable alors le système 2.16 ou vice-versa est satisfait. Puisque le système est ordonnançable alors il y a au moins un ordonnancement S qui satisfait toutes les contraintes de précédences et de latences. Dans cet ordonnancement A est ordonnancée avant ou après C.

Dans le cas où A est ordonnancée avant C puisqu'on est dans le cas des contraintes de latence qui ne sont pas égales à la somme des durées d'exécution des opérations appartenant à la contrainte alors B est ordonnancée avant D (voir figure 2.32).

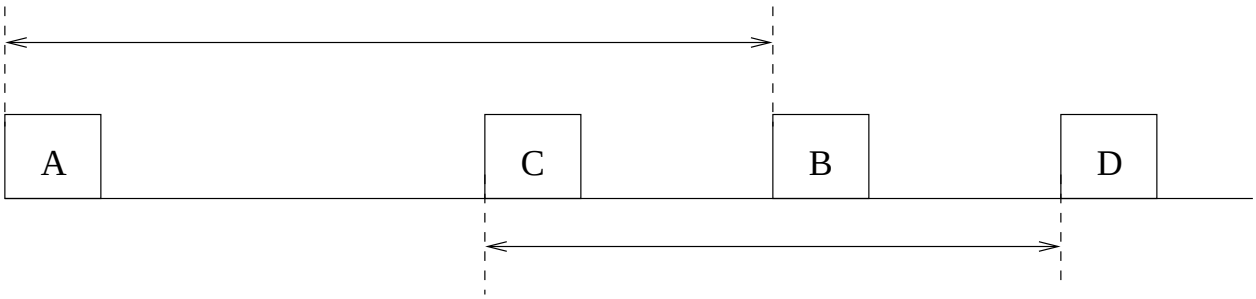

FIG. 2.32 – *Ordonnancement illustrant le théorème 18*

Dans le cas où A est ordonnancée après C puisqu'on est dans le cas des contraintes de latence qui ne sont pas égales à la somme des durées d'exécution des opérations appartenant

à la contrainte alors B est ordonnancée après D (voir figure 2.33). Le théorème est prouvé ⊡

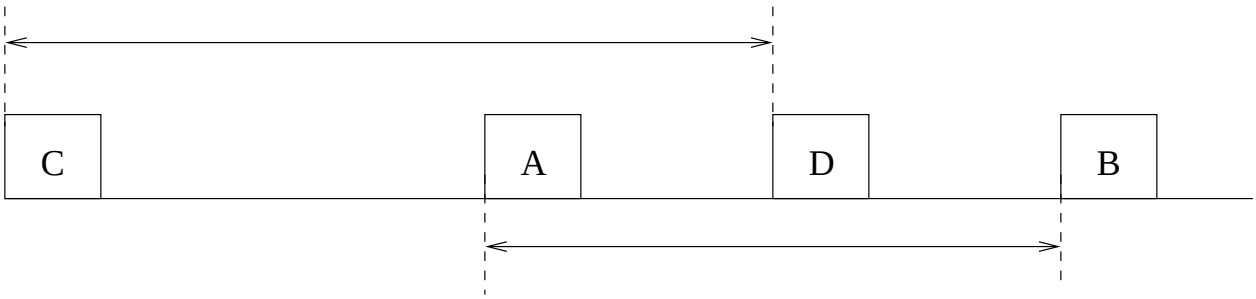

FIG. 2.33 – *Ordonnancement illustrant le théorème 18*

Remarque 14 *Pour le cas traité dans le théorème 18 les ensembles minimaux des deux contraintes de latence contiennent pas seulement des opérations appartenant à la contrainte de latence, mais aussi des opérations appartenant à l'autre contrainte de latence. L'exemple 10 illustre des tels ensembles.*

Exemple 14 *Cet exemple illustre le théorème 18. On considère le système d'opérations donné dans la figure 2.34 avec les contraintes de latence $L(A,C)$ et $L(D,I)$.*

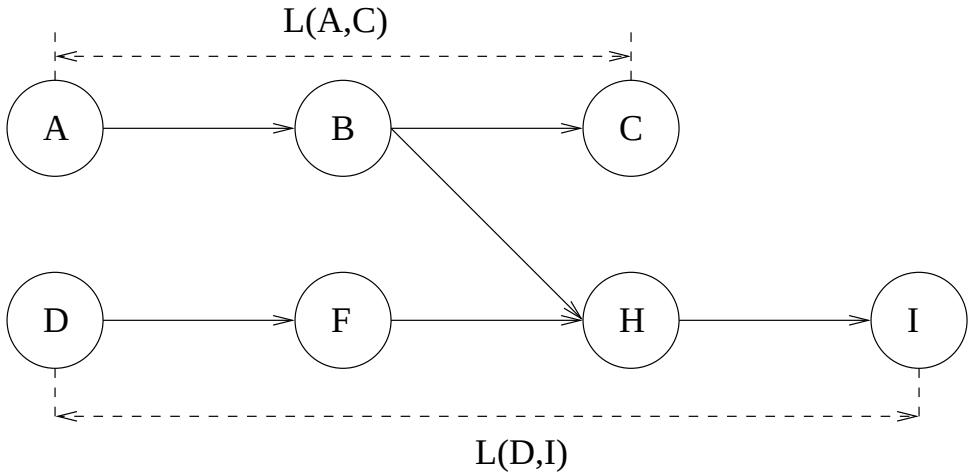

FIG. 2.34 – *Graphe illustrant le théorème 18*

Théorème 19 *On a un système défini par un graphe $\mathcal{G} = (\mathcal{V},\mathcal{E})$ tel que toutes les paires sur lesquelles il y a des contraintes de latence définies sont en relation* X. *Le système est ordonnançable si et seulement si pour chaque contrainte de latence définie sur une paire*

(A,B) en relation X *avec n paires $(C_i,D_i),\forall i\{1,2,\ldots,n\}$, sur lesquelles il y a des contraintes de latence définies, on a*

$$L_{AB} \geq \sum_{H \in \mathcal{W}} C_H \tag{2.17}$$

où :

$$\mathcal{W} = \mathcal{M}(A,B) \bigcup \bigcup_{i \in \{1,\ldots,j\}; E \in lt_{AB}(C_i,D_i)} \mathcal{M}(C_i,E) \bigcup \bigcup_{i \in \{j+1,\ldots,k\}; E \in ft_{AB}(C_i,D_i)} \mathcal{M}(E,D_i) \bigcup \bigcup_{i \in \{k+1,\ldots,m\}} \mathcal{M}(C_i,D_i),$$

où :

$j < k < m < n$ sont des nombres entiers positifs qui satisfont les relations suivantes :

$$L_{C_iD_i} \geq \sum_{H \in \{\mathcal{M}(C_i,D_i) \bigcup \bigcup_{E \in ft_{C_iD_i}(AB)} \mathcal{M}(E,B)\}} C_H,\ \forall i \in \{1,\ldots,j\}$$

$$L_{C_iD_i} \geq \sum_{H \in \{\mathcal{M}(C_i,D_i) \bigcup \bigcup_{E \in lt_{C_iD_i}(A,B)} \mathcal{M}(A,E)\}} C_H,\ \forall i \in \{j+1,\ldots,k\}$$

$$L_{C_iD_i} \geq \sum_{H \in \mathcal{M}(C_i,D_i)} C_H,\ \forall i \in \{k+1,\ldots,m\}\ et$$

$$L_{C_iD_i} \geq \sum_{H \in \{\mathcal{M}(C_i,D_i) \bigcup \mathcal{M}(A,B)\}} C_H,\ \forall i \in \{m+1,\ldots,n\}.$$

Preuve La preuve est faite par double implication. D'abord, on part de l'hypothèse que l'on a un système ordonnançable, ce qui signifie qu'il y a un ordonnancement $S \in \mathcal{S}$ qui satisfait toutes les contraintes de latence : $s_B - s_A + C_B \leq L_{AB}$, $\forall L(A,B) \in \mathcal{S}$. Mais $s_B - s_A + C_B$ est la somme des durées d'exécution des opérations qui sont ordonnancées de s_A à s_B. Du fait de l'ordre partiel, l'ensemble des opérations ordonnancées de s_A à s_B contient au moins des opérations appartenant à $\mathcal{M}(C_i,D_i),\forall i \in \{1,2,\ldots,n\}$. Les opérations de $\mathcal{M}(C_i,D_i)$ qui sont ordonnancées de s_A à s_B résultent de la façon dont la paire (C_i,D_i),est ordonnancée par rapport à la paire (A,B), $\forall i \in \{1,2,\ldots,n\}$ dans l'ordonnancement S. En conséquence on doit étudier tous les ordres possibles pour ces deux paires d'opérations, c'est-à-dire six ordres tel que A soit toujours ordonnancée avant B et C avant D. Puisque (A,B) est en relation X avec (C_i,D_i), parmi ces ordres, les ordres $s_A + C_A \leq s_B + C_B \leq s_{C_i} + C_{C_i} \leq s_{D_i} + C_{D_i}$ et $s_{C_i} + C_{C_i} \leq s_{D_i} + C_{D_i} \leq s_A + C_A \leq s_B + C_B$ sont interdits. Il reste quatre cas possibles pour ordonnancer les paires (A,B) et $(C_i,D_i),\forall i \in \{1,2,\ldots,n\}$:

1. $s_A + C_A \leq s_{C_i} + C_{C_i} \leq s_B + C_B \leq s_{D_i} + C_{D_i},\forall i \in \{1,2,\ldots,j\}$;
2. $s_{C_i} + C_{C_i} \leq s_A + C_A \leq s_{D_i} + C_{D_i} \leq s_B + C_B,\forall i \in \{j,\ldots,k\}$;
3. $s_{C_i} + C_{C_i} \leq s_A + C_A \leq s_B + C_B \leq s_{D_i} + C_{D_i},\forall i \in \{k+1,\ldots,m\}$;
4. $s_A + C_A \leq s_{C_i} + C_{C_i} \leq s_{D_i} + C_{D_i} \leq s_B + C_B,\forall i \in \{m+1,\ldots,n\}$;

En utilisant le lemme 4, le cas 1 implique que toutes les opérations appartenant à $\mathcal{M}(A,E), E \in \Gamma^{-}_{C_i,D_i}(AB)$, $\forall i \in \{1,2,\ldots,j\}$ sont ordonnancées de A à B. En conséquence, puisque C_i est ordonnancée avant B le reste des opérations appartenant à $\mathcal{M}(A,B)$, c'est-à-dire les opérations appartenant à $\mathcal{M}(E,B), E \in ft_{C_iD_i}(A,B)$ sont ordonnancées entre C_i et $D_i, \forall i \in \{1,2,\ldots,j\}$.

En utilisant le lemme 5, le cas 1 implique que toutes les opérations appartenant à $\mathcal{M}(E,D_i), E \in \Gamma^{+}_{AB}(C_i,D_i)$, $\forall i \in \{1,2,\ldots,j\}$ sont ordonnancées de B à D_i. En conséquence, puisque C_i est ordonnancée avant B le reste des opérations appartenant à $\mathcal{M}(A,B)$, c'est-à-dire les opérations appartenant à $\mathcal{M}(C_i,E), E \in lt_{AB}(C_i,D_i)$ sont ordonnancées entre A et B.

En utilisant le lemme 4, le cas 2 implique que toutes les opérations appartenant à $\mathcal{M}(C_i,E), E \in \Gamma^{-}_{AB}(C_i,D_i)$, $\forall i \in \{j+1,\ldots,k\}$ sont ordonnancées de C_i à A. En conséquence, puisque A est ordonnancée avant D_i le reste des opérations appartenant à $\mathcal{M}(C_i,D_i)$, c'est-à-dire les opérations appartenant à $\mathcal{M}(E,D_i), E \in ft_{AB}(C_i,D_i)$ sont ordonnancées entre A et B.

En utilisant le lemme 5, le cas 2 implique que toutes les opérations appartenant à $\mathcal{M}(E,B), E \in \Gamma^{+}_{C_iD_i}(A,B)$, $\forall i \in \{j+1,\ldots,k\}$ sont ordonnancées de C_i à D_i. En conséquence, puisque A est ordonnancée avant D_i le reste des opérations appartenant à $\mathcal{M}(A,B)$, c'est-à-dire les opérations appartenant à $\mathcal{M}(A,E), E \in lt_{C_i,D_i}(A,B)$ sont ordonnancées entre C et D.

Le cas 3 implique que toutes les opérations appartenant à $\mathcal{M}(C_i,D_i), \forall i \in \{k+1,\ldots,m\}$ sont ordonnancées de A à B. Le cas 3 ne permet pas de dire si les opérations appartenant à $\mathcal{M}(A,B)$ sont ordonnancées de C_i à D_i.

Le cas 4 implique que toutes les opérations appartenant à $\mathcal{M}(A,B)$ sont ordonnancées de C_i à $D_i, \forall i \in \{m+1,\ldots,n\}$. Le cas 4 ne permet pas de dire si les opérations appartenant à $\mathcal{M}(C_i,D_i)$ sont ordonnancées de A à B.

En conclusion, l'ensemble des opérations qui sont ordonnancées de A à B contient au moins les opérations appartenant à

$$\mathcal{W} = \mathcal{M}(A,B) \bigcup \bigcup_{i\in\{1,\ldots,j\};E\in lt_{AB}(C_i,D_i)} \mathcal{M}(C_i,E) \bigcup \bigcup_{i\in\{j+1,\ldots,k\};E\in ft_{AB}(C_i,D_i)} \mathcal{M}(E,D_i) \bigcup \bigcup_{i\in\{k+1,\ldots,\}} \mathcal{M}(C_i,D_i), \text{ où}$$

$j,k,m \in \mathbb{N}$ sont des nombres entiers positifs qui satisfont les relations suivantes :

$$s_{D_i} - s_{C_i} + C_{D_i} \geq \sum_{H\in\{\mathcal{M}(C_i,D_i) \bigcup \bigcup_{E\in ft_{C_iD_i}(A,B)} \mathcal{M}(E,B)\}} C_H, \ \forall i \in \{1,\ldots,j\},$$

$$s_{D_i} - s_{C_i} + C_{D_i} \geq \sum_{H\in\{\mathcal{M}(C_i,D_i) \bigcup \bigcup_{E\in lt_{C_iD_i}(A,B)\}} \mathcal{M}(A,E)\}} C_H, \ \forall i \in \{j+1,\ldots,k\},$$

$$s_{D_i} - s_{C_i} + C_{D_i} \leq \sum_{H\in\mathcal{M}(C_i,D_i)} C_H, \ \forall i \in \{k+1,\ldots,m\} \text{ et}$$

$$s_{D_i} - s_{C_i} + C_{D_i} \geq \sum_{H \in \{\mathcal{M}(C_i,D_i) \bigcup \bigcup_{E \in \mathcal{M}(A,B)}} C_H \quad , \forall i \in \{m+1,\ldots,n\}.$$

En utilisant $L(A,B) \geq s_{D_i} - s_{C_i} + C_{D_i}$, on obtient l'inégalité (2.17). L'implication directe est prouvée.

On considère l'inégalité (2.17) satisfaite pour chaque paire (A,B) avec une contrainte de latence définie sur elle. Ceci signifie qu'on peut ordonnancer de A à B les opérations appartenant à $\mathcal{W} = \mathcal{M}(A,B) \bigcup \bigcup_{i \in \{1,\ldots,j\}; E \in lt_{AB}(C_i,D_i)} \mathcal{M}(C_i,E) \bigcup \bigcup_{i \in \{j+1,\ldots,k\}; E \in ft_{AB}(C_i,D_i)} \mathcal{M}(E,D_i) \bigcup \bigcup_{i \in \{k+1,\ldots,\}} \mathcal{M}(C_i,D_i)$. L'ensemble $\mathcal{W}$ contient toutes les opérations appartenant à $\mathcal{M}(C_i,D_i)$, $\forall i \in \{1,\ldots,n\}$ qui doivent être ordonnancées de A à B pour satisfaire l'ordre partiel. En utilisant l'inégalité (2.17) pour chaque paire (A,B) on obtient un ordonnancement qui satisfait l'ordre partiel. Puisque les contraintes de latence sont déjà satisfaites par l'inégalité (2.17), le système est ordonnançable. L'implication inverse est prouvée et le théorème est prouvé ⊡

Remarque 15 *Le théorème 19 donne des conditions d'ordonnançabilité qui permettent d'obtenir les ensembles minimaux correspondant aux contraintes de latence. Puisque construire ces ensembles est probablement un problème NP-difficile et pour rendre le problème simple, on considère à partir de maintenant seulement les cas où chaque contrainte de latence est en relation* X *avec au plus une autre contrainte de latence. Ces cas correspondent à des cas réalistes dans l'industrie et on donne pour ces cas un algorithme d'ordonnancement polynomial dans la section suivante.*

On donne un exemple d'un système avec toutes les contraintes de latence en relation X.

Exemple 15 *Soit le graphe $\mathcal{G} = (\mathcal{V},\mathcal{E})$ donné dans la figure 2.35 avec les durées d'exécution $C_A = 1, \forall A \in \mathcal{V}$ et avec les contraintes de latence $L_{AC} = 3$, $L_{DF} = 9$ et $L_{HJ} = 5$. On remarque que (A,C)* X *(D,F) et (D,F)* X *(H,J). Les conditions d'ordonnançabilité sont satisfaites et la figure 2.36 donne un ordonnancement qui satisfait les contraintes du système.*

Ordonnançabilité pour les contraintes de latence définies sur des paires en relation ||, Z et X

Donner une condition générale d'ordonnançabilité revient à prouver qu'il y a un ordonnancement qui satisfait toutes les contraintes de latence si et seulement si chaque contrainte de latence en relation ||, Z ou X avec une autre contrainte de latence est satisfaite. Jusqu'à maintenant, on a obtenu séparément une condition d'ordonnançabilité pour des systèmes avec toutes les paires en relation || et Z et une condition d'ordonnançabilité pour des systèmes avec toutes les paires en relation X. A partir de maintenant, on va étudier l'ordonnançabilité des systèmes avec des contraintes de latence $L_{X,Y}$ plus grandes que la somme des durées

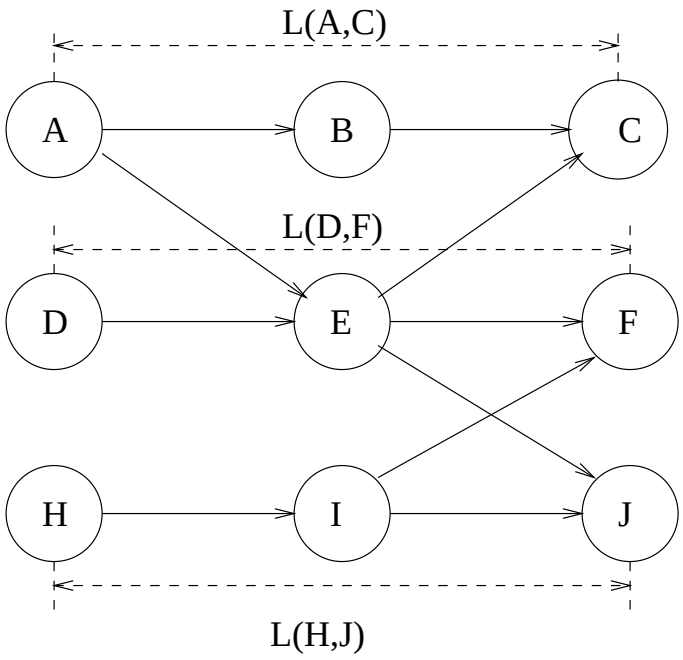

FIG. 2.35 – *Système d'opérations avec les contraintes de latence en relation* X

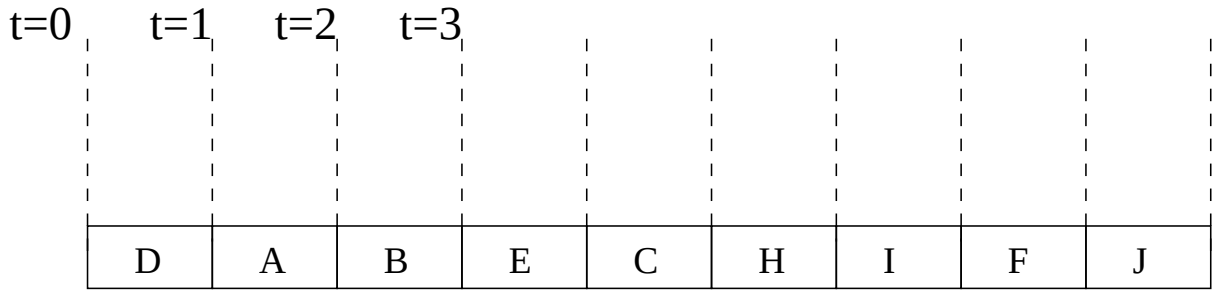

FIG. 2.36 – *Ordonnancement satisfaisant toutes les contraintes*

d'exécution des opérations appartenant à $\mathcal{M}(X,Y)$ puisque les systèmes qui ne satisfont pas cette dernière condition ne sont pas ordonnançables, le théorème 15 le prouvant.

Le théorème suivant donne une condition générale d'ordonnançabilité pour les systèmes avec des contraintes de latence quelconques.

Théorème 20 *Soit $\mathcal{G} = (\mathcal{V},\mathcal{E})$ un graphe et $\mathcal{L}$ un ensemble des contraintes de latence définies sur des paires d'opérations appartenant à $\mathcal{V}$. Il y a un ordonnancement qui satisfait toutes les contraintes de latence appartenant à $\mathcal{L}$ d'un système des opérations défini par le graphe $\mathcal{G}$ si et seulement si l'inégalité (2.17) est satisfaite pour chaque paire (A,B) en relation* X *avec n paires $(C_i,D_i),\forall i \in \{1,2,\dots n\}$.*

Preuve La preuve est faite par double implication et elle suit les mêmes principes que la preuve du théorème 19. Donc l'implication directe suppose que le système est ordonnançable et prouve que l'inégalité (2.17) est satisfaite. L'implication inverse construit un ordonnancement qui satisfait toutes les contraintes de latence, en utilisant l'inégalité (2.17) supposée satisfaite ⊡

Remarque 16 *Pour un système avec n contraintes de latence, le nombre d'inégalités de type (2.17) qui doivent être vérifiées est au plus $n(n-1)$. En effet, si on supposait que toutes les n contraintes de latence sont définies sur des paires en relation* X *alors chaque contrainte de latence doit être comparée avec $n-1$ contraintes de latence.*

Exemple 16 *Pour le graphe dans la figure 2.37 on a toutes les durées d'exécution égales à 1, $C_A = 1,\forall A \in \mathcal{V}$. On définit les contraintes de latence suivantes : $L_{AB} = 7$, $L_{CD} = 12$, $L_{MF} = 3$ et $L_{OR} = 4$. On remarque que (A,B)*X*(C,D), (A,B)*Z*(M,F), (C,D)*X*(M,F) et (O,R) est en relation || avec toutes les autres contraintes de latence. D'abord, on vérifie que toutes les contraintes de latence $L_{X,Y}$ sont plus grandes que la somme des durées d'exécution des opérations appartenant à $\mathcal{M}(X,Y)$. En utilisant le théorème 20 le système est ordonnançable si et seulement si les relations suivantes sont satisfaites :*

$$L_{AB} \geq \sum_{X \in \{\mathcal{M}(A,B) \bigcup \bigcup_{Y \in lt_{AB}(C,D)} \mathcal{M}(C,Y)\}} C_X$$

$$L_{CD} \geq \sum_{X \in \{\mathcal{M}(C,D) \bigcup \bigcup_{Y \in ft_{CD}(A,B)} \mathcal{M}(Y,B) \bigcup \mathcal{M}(M,F)\}} C_X$$

$$L_{MF} \geq \sum_{X \in \mathcal{M}(M,F)} C_X$$

On obtient:

$$\begin{cases} L_{AB} \geq C_A + C_G + C_H + C_I + C_B + C_C + C_K \\ L_{CD} \geq C_C + C_I + C_J + C_K + C_L + C_D + C_M + C_N + C_F + C_H + C_B \\ L_{MF} \geq C_M + C_N + C_F \end{cases}$$

Puisque les trois équations sont vérifiées le système est ordonnançable. En effet il y a un ordonnancement S avec $s_O \leq s_P \leq s_R \leq s_E \leq s_A \leq s_G \leq s_C \leq s_K \leq s_I \leq s_H \leq s_B \leq s_M \leq s_N \leq s_F \leq s_J \leq s_L \leq s_D$ qui satisfait toutes les contraintes de latence. Dans le sous-chapitre suivant, on donne un algorithme qui permet d'une manière générale de trouver un ordonnancement qui satisfait les contraintes de latence.

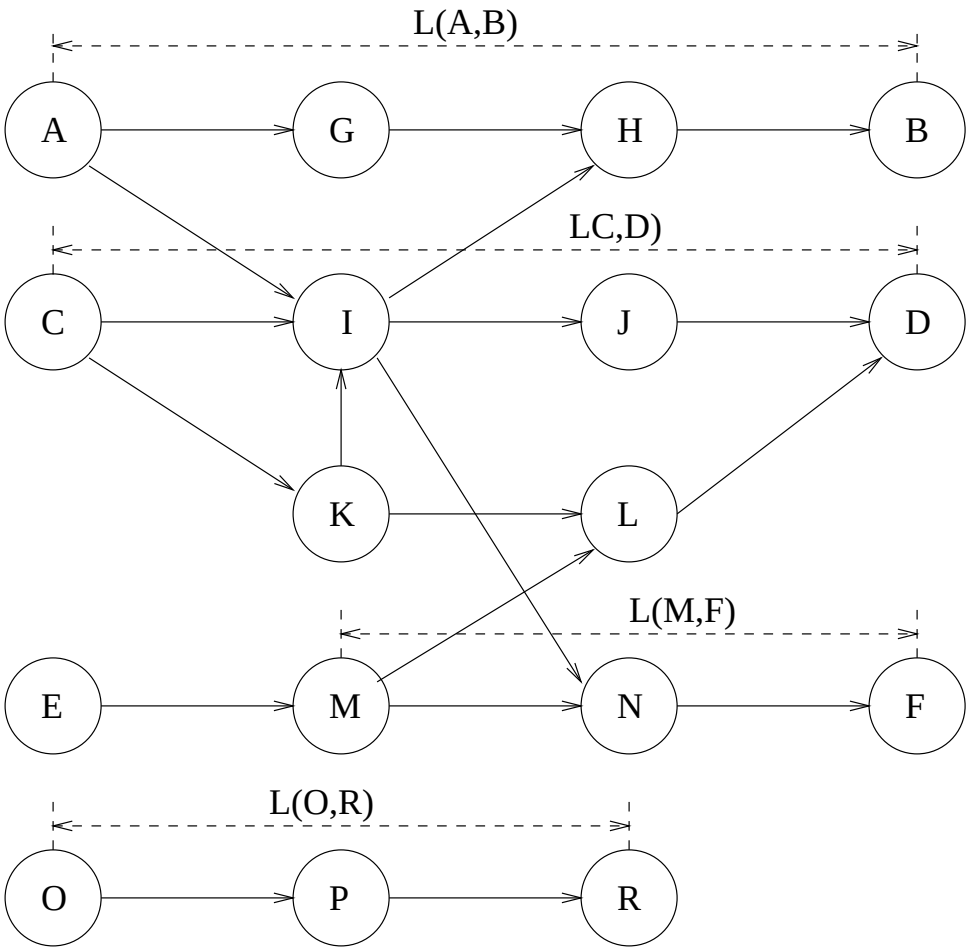

FIG. 2.37 – *Graphe illustrant la condition générale d'ordonnançabilité*

2.2.3 Algorithme optimal d'ordonnancement

Un algorithme d'ordonnancement transforme l'ordre partiel défini par le graphe en un ordre total (un des ordres possible) qui satisfait les contraintes de latence des opérations. Dans notre cas, l'algorithme d'ordonnancement est appliqué seulement sur le motif et il utilise un marquage des opérations du graphe. Ce marquage des sommets est obtenu à l'aide de l'algorithme 3, lui aussi appliqué seulement sur le motif. Il "prévoit" quelles opérations sont importantes pour les contraintes de latence.

On note par $\mathcal{G} = (\mathcal{V},\mathcal{E})$ le graphe-motif orienté acyclique des opérations, où $\mathcal{V}$ est l'ensemble des opérations et $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ est l'ensemble des arcs et par $\mathcal{D}(A) = \{B \in \mathcal{V}$ tel que $\exists$ au moins un chemin de A à $B\}$ l'ensemble des opérations liées par au moins un chemin de A. On note par $\mathcal{W}$ l'ensemble de travail utilisé pendant le déroulement de l'algorithme.

L'algorithme de marquage associe à chaque opération A une marque qui indique si l'opération A doit être ordonnancée avant la première opération d'une contrainte de latence quand il y a au moins un chemin de A vers la dernière opération de cette contrainte de latence. S'il y a plusieurs opérations satisfaisant cette propriété alors la marque sera égale à la plus petite contrainte de latence. On note cette marque par $marque(A) \in \mathbb{N}^* \bigcup \{\infty\}$, où ∞ est un nombre entier positif plus grand que tous les autres nombres appartenant à $\mathbb{N}^*$. Les marques peuvent changer de valeurs pendant l'algorithme de marquage.

Lemme 6 *Si une paire d'opérations (A,B) a une contrainte de latence $L(A,B)$ et si il y a une opération C avec $A \in \mathcal{D}(C)$, alors $s_B + C_B - s_A \leq L_{AB}$ est vérifiée quelle que soit s_C.*

Preuve Si (A,B) a une contrainte de latence alors $B \in \mathcal{D}(A)$ et $s_A \leq s_B$. Puisque $A \in \mathcal{D}(C)$ on a $s_C \leq s_A$. Donc on obtient $s_C \leq s_A \leq s_B$ et puisque C est déjà ordonnancée alors quand A devient ordonnançable la date de début de B n'est pas modifiée par la date de début de C. Le lemme est prouvé $\boxdot$

Algorithme 3

Initialisation : si (A,B) a une contrainte de latence $L(A,B)$ alors $marque(B) = L(A,B)$ et $marque(A) = \infty$, sinon $marque(A) = marque(B) = \infty$. De plus, si B appartient à plusieurs paires avec des contraintes de latence alors $marque(B) = min_{(C,B)\in\mathcal{L}}\{L(C,B)\}$ et $marque(A) = \infty$. Soit $\mathcal{W} = \mathcal{L}$.

Pas 1 : pour $(A,B) \in \mathcal{W}$ et pour chaque opération $C \in \mathcal{V}\backslash\{A,B\}$, il y a trois possibilités :
(a) si $A \in \mathcal{D}(C)$ $marque(C) = marque(C)$;
(b) si $A \notin \mathcal{D}(C)$ et $B \in \mathcal{D}(C)$, alors $marque(C) = min(marque(C),marque(B))$;
(c) si $A \notin \mathcal{D}(C)$ et $B \notin \mathcal{D}(C)$ alors $marque(C) = marque(C)$.
Soit $\mathcal{W} = \mathcal{W}\backslash\{(A,B)\}$.

Pas 2 : si $\mathcal{W} \neq \emptyset$ alors on va au Pas 1, sinon l'algorithme s'arrête.

On note par $\mathcal{L}_X$ l'ensemble des latences qui sont en relation X avec d'autres contraintes de latence et qui n'ont pas la différence entre la latence et la somme des durées d'exécution des opérations appartenant à la paire égale à 0. On considère cet ensemble ordonné, si $L(A,B)$ est avant $L(C,D)$ dans l'ensemble $\mathcal{L}_X$ alors $L_{AB} < L_{CD}$.

L'algorithme d'ordonnancement utilise les marques obtenues après avoir appliqué l'algorithme de marquage. On note par $\mathcal{W}$ l'ensemble de travail, par s_T la date de début de la dernière opération ordonnancée, par C_T sa durée d'exécution et par $Prec(A)$ l'ensemble des prédécesseurs de l'opération A. Pendant le déroulement de l'algorithme l'ensemble $\mathcal{W}$

contient les opérations prêtes à être ordonnancées, c'est-à-dire les opérations dont les prédécesseurs sont déjà ordonnancés. On remarque qu'entre deux opérations qui sont disponibles au même instant, il ne peut y avoir un chemin. Pendant le déroulement de l'algorithme, chaque fois qu'une opération est ordonnancée, s_T (respectivement C_T) est remplacé par la date de début (la durée d'exécution) de cette opération.

Algorithme 4

Initialisation : $\mathcal{W} = \bigcup_{A \in \mathcal{V} \ et \ Prec(A)=\emptyset} \{A\}$ et $s_T = 0, C_T = 0$.

Pas 1 *(opération avec contrainte de latence)*:
si $\exists A \in \mathcal{W}$ tel que $marque(A) \neq \infty$ alors on choisit A tel que $marque(A) = min_{B \in \mathcal{W}}\{marque(B)\}$. Si $\exists L(C_1,D_1) \in \mathcal{L}_X$ et $\exists L(C_i,D_i) \in \mathcal{L}_X, \forall i = 2,3,\ldots n$ tel que $A \in \bigcup_{E \in \Gamma^+_{C_i D_i}(C_1,D_1)} \mathcal{M}(E,D_1), \forall i = 2,3,\ldots n$ alors soit $B \in \mathcal{W}$ l'opération avec la plus petite marque tel que $\exists i \in \{2,3,\ldots,n\}$ avec $B \in \bigcup_{E \in \Gamma^+_{C_1 D_1}(C_i,D_i)} \mathcal{M}(E,D_i)$. Si l'opération B existe alors $s_B = s_T + C_T$, on enlève B de $\mathcal{W}$, toutes les opérations qui deviennent ordonnançables sont ajoutées à $\mathcal{W}$ et on va au Pas 4. Si l'opération B n'existe pas alors $s_A = s_T + C_T$, on enlève A de $\mathcal{W}$, toutes les opérations qui deviennent ordonnançables sont ajoutées à $\mathcal{W}$ et on va au Pas 4.

Pas 2*(opération qui n'est pas première d'une contrainte de latence)*:
si $\exists A \in \mathcal{W}$ tel qu'il n'y a pas d'opération $B \in \mathcal{V}$ avec $(A,B) \in \mathcal{L}$ alors $s_A = s_T + C_T$, on l'enlève de $\mathcal{W}$, toutes les opérations qui deviennent ordonnançables sont ajoutées à $\mathcal{W}$ et on va au Pas 4.

Pas 3 *(opération qui est première d'une contrainte de latence)*:
si $\exists A \in \mathcal{W}$ tel qu'il y a $B \in \mathcal{V}$ avec $(A,B) \in \mathcal{L}$ alors $s_A = s_T + C_T$ avec $L(A,B) = max_{(C,D) \in \mathcal{L} \ et \ C \in \mathcal{W}}\{L(C,D)\}$, on l'enlève de $\mathcal{W}$, toutes les opérations qui deviennent ordonnançables sont ajoutées à $\mathcal{W}$ et on va au Pas 4.

Pas 4: si $\mathcal{W} = \emptyset$ alors l'algorithme s'arrête, sinon on va au Pas 1.

L'algorithme d'ordonnancement 4 est prouvé optimal dans le cas où chaque contrainte de latence est en relation X avec au plus une autre contrainte de latence. Avant de prouver l'optimalité de l'algorithme d'ordonnancement 4 on donne un exemple d'ordonnancement obtenu à l'aide de l'algorithme.

Exemple 17 *Soit $\mathcal{G} = (\mathcal{V},\mathcal{E})$ le graphe donné dans la figure 2.38 avec $C_A = 2$, $C_B = 1$ et $C_C = 2$ et les contraintes de latence $L(A_2,C_2) = 10$ et $L(B,C_1) = 9$. Après avoir appliqué l'algorithme de marquage on obtient le tableau 2.3 où on utilise $m(A)$ pour désigner $marque(A)$.*

Après avoir appliqué l'algorithme d'ordonnancement 4, on obtient le tableau 2.4.

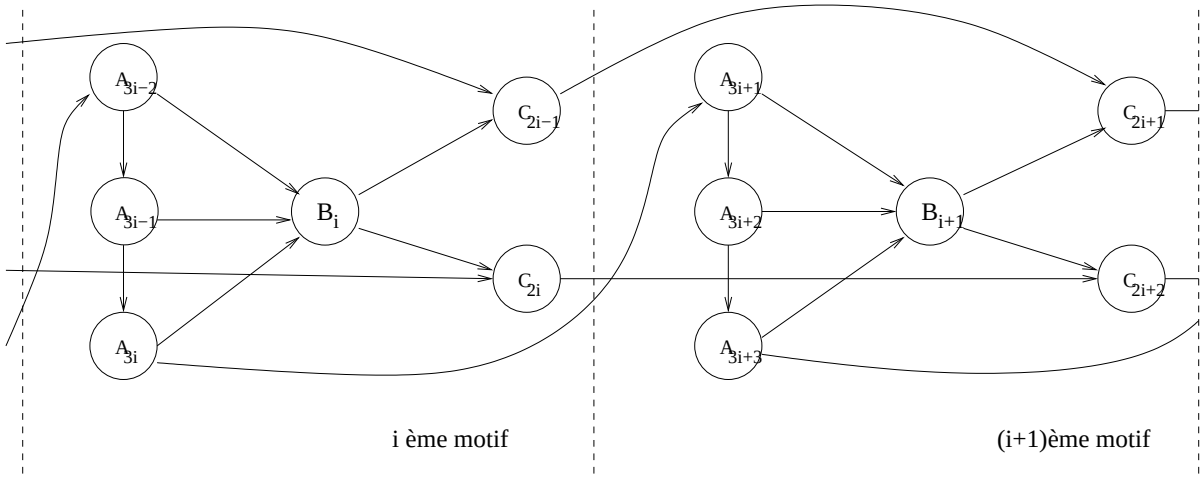

FIG. 2.38 – *Graphe illustrant l'algorithme d'ordonnancement 4*

	$m(A_1)$	$m(A_2)$	$m(A_3)$	$m(B)$	$m(C_1)$	$m(C_2)$
Initialisation	∞	∞	∞	∞	9	10
$L(B,C_1)$	∞	∞	∞	∞	9	10
$L(A_2,C_2)$	∞	∞	10	10	9	10

TAB. 2.3 – *Marques obtenues après avoir utilisé l'algorithme de marquage*

Théorème 21 *L'algorithme d'ordonnancement 4 est optimal (s'il y a un ordonnancement, l'algorithme le trouvera).*

Preuve On prouve que l'algorithme d'ordonnancement 4 construit des ordonnancements qui satisfont les conditions d'ordonnançabilité et si ces ordonnnancements ne satisfont pas les contraintes de latence alors aucun autre ordonnancement ne le fera d'où l'optimalité de l'algorithme. La preuve a trois parties : une partie concernant les contraintes de latence définies sur des paires en relation ||, une deuxième partie concernant les contraintes de latence définies sur des paires en relation Z et une dernière partie concernant les contraintes de latence définies sur des paires en relation X.

Sans perdre de généralité, on considère que toutes les contraintes de latence ont des valeurs différentes.

On considère deux contraintes de latence quelconques $L(A,B)$ et $L(C,D)$ définies sur des paires qui sont en relation ||. Puisqu'il n'y aucun chemin d'une opération appartenant à la paire (A,B) à une opération appartenant à la paire (C,D) alors la marque des opérations appartenant à la paire (A,B) n'est pas modifée par la contrainte de latence $L(C,D)$. De même puisqu'il n'y aucun chemin d'une opération appartenant à la paire (C,D) à une opération appartenant à la paire (A,B) alors la marque des opérations appartenant à la paire (C,D) n'est pas modifée par la contrainte de latence $L(A,B)$. Donc les opérations appartenant aux deux paires ont des marques différentes et elles vont s'ordonnancer soit dans l'ordre $s_A+C_A <$

$\mathcal{W}$	s_T	C_T	Pas utilisé	Décision
$\{A_1\}$	0	0	Pas 2	$s_{A_1} = 0$
$\{A_2\}$	0	2	Pas 3	$sA_2 = 2$
$\{A_3\}$	2	2	Pas 1	$s_{A_3} = 4$
$\{B\}$	4	2	Pas 1	$s_B = 6$
$\{C_1,C_2\}$	6	1	Pas 1	$s_{C_2} = 7$
$\{C_1\}$	7	2	Pas 1	$s_{C_1} = 9$

TAB. 2.4 – *Ordonnancement obtenu en utilisant l'algorithme 4*

$s_B + C_B < s_C + C_C < s_D + C_D$, soit dans l'ordre $s_C + C_C < s_D + C_D < s_A + C_A < s_B + C_B$. Les deux ordres correspondent bien à la condition d'ordonnançabilité pour les contraintes de latence définies sur des paires en relation ||.

On considère deux contraintes de latence quelconques $L(A,B)$ et $L(C,D)$ définies sur des paires qui sont en relation Z. Puisqu'il n'y aucun chemin d'une opération appartenant à la paire (C,D) à une opération appartenant à la paire (A,B) alors la marque des opérations appartenant à la paire (C,D) n'est pas modifée par la contrainte de latence $L(A,B)$. Puisqu'il y a un chemin entre des opérations appartenant à la paire (A,B) à au moins une opération appartenant à la paire (C,D), alors la marque de A peut être modifiée par la contrainte de latence $L(C,D)$ seulement si $L_{AB} > L_{CD}$. Dans ce cas les opérations appartenant aux deux paires vont s'ordonnancer dans l'ordre $s_A + C_A < s_B + C_B < s_C + C_C < s_D + C_D$. Dans le cas contraire, la marque de A n'est pas modifiée par la contrainte de latence $L(C,D)$ et les opérations appartenant aux deux paires vont s'ordonnancer soit dans l'ordre $s_A + C_A < s_B + C_B < s_C + C_C < s_D + C_D$, soit dans l'ordre $s_C + C_C < s_D + C_D < s_A + C_A < s_B + C_B$. Les deux ordres correspondent bien à la condition d'ordonnançabilité pour les contraintes de latence définies sur des paires en relation Z.

On considère deux contraintes de latence quelconques $L(A,B)$ et $L(C,D)$ définies sur des paires qui sont en relation X. Puisqu'il y a un chemin entre des opérations appartenant à la paire (A,B) à au moins une opération appartenant à la paire (C,D), alors la marque de l'opération A peut être modifiée par la contrainte de latence $L(C,D)$ seulement si $L_{AB} > L_{CD}$. Dans ce cas l'existence d'un chemin entre des opérations appartenant à la paire (C,D) à au moins une opération appartenant à la paire (A,B) ne modifie pas la marque. De la même manière on raisonne pour le cas contraire où $L_{AB} < L_{CD}$.

Pour les trois cas, les conditions d'ordonnançabilité sont satisfaites par les ordonnancements obtenus en utilisant l'algorithme 4. Le théorème est prouvé ⊡

On passe au cas des systèmes avec contraintes de précédences, de périodicités et de latences. On s'attend à ce que les résultats concernant les systèmes avec contraintes de périodicité soient généralisés pour ce cas. En effet l'existence des contraintes de latence n'empêche pas l'héritage des périodes entre les opérations. Cela signifie qu'on prouve aussi l'existence d'un motif dans l'ordonnancement pour le cas des systèmes avec contraintes de précédences, périodicités et latences. Au contraire, les contraintes de latence sont affectées par les contraintes de périodicité puisque les opérations qui ont des périodes plus petites que

les opérations appartenant à une paire avec une contrainte de latence vont forcement être ordonnancées plusieurs fois entre la première et la dernière opération de la latence. Donc la durée de l'ordonnancement entre les deux opérations va être plus grande que dans le cas des systèmes avec contraintes de précédences et de latences.

2.3 Contraintes de précédences, de périodicités et de latences

Ce chapitre commence par rappeler le modèle utilisé dans le chapitre 2.1. On continue par l'étude d'ordonnançabilité qui commence par présenter les résultats-conséquences de la présence de latences et de périodicités dans le même système et puis on généralise les résultats obtenus dans les deux chapitres précédents. On finit ce chapitre par donner un algorithme d'ordonnancement prouvé optimal. La présence de contraintes de latence parmi les contraintes impose l'utilisation de l'algorithme de marquage puisque l'algorithme d'ordonnancement doit être capable de prévoir quelles sont les opérations importantes des contraintes de latence.

2.3.1 Modèle

On commence par rappeler la définition de la périodicité :

Définition 6 *une opération A a une contrainte de périodicité T_A si $s_{A_{i+1}} - s_{A_i} = T_A, \forall i \geq 1$, où A_i,A_{i+1} sont les répétitions consécutives i et $i+1$ de l'opération A. On note par A_1 la première répétition de A.*

Le théorème 22 prouve que la contrainte de périodicité est un cas particulier de latence. Cela signifie qu'on peut définir une contrainte de latence dans un graphe d'opérations entre deux opérations avec contraintes de périodicité. D'abord on rappelle la définition de la latence telle qu'elle a été donnée dans le chapitre 2.2 :

Définition 7 *deux opérations différentes A et B appartenant au même motif telles que $\exists P(A,B) \in \mathcal{P}$, ont une contrainte de latence $L_{AB} \in \mathbb{N}$ si elles sont ordonnancées telles que $s_B + C_B - s_A \leq L_{AB}$. Soit $\mathcal{L}$ l'ensemble de toutes les contraintes de latence définies pour un système.*

Remarque 17 *En utilisant la définition 2, un système d'opérations avec contraintes de précédences, de latences et de périodicités est ordonnançable s'il y a au moins un ordonnancement qui satisfait toutes ses contraintes de précédences, de latences et de périodicités. Trouver un tel ordonnancement est le problème que nous cherchons à résoudre dans ce chapitre.*

Théorème 22 *Une contrainte de périodicité est une contrainte de latence.*

Preuve Soit A une opération périodique quelconque avec la période T_A. D'après la définition de la périodicité, on a

$$s_{A_{i+1}} - s_{A_i} = T_A, \forall i \geq 1 \tag{2.18}$$

où A_i,A_{i+1} sont les répétitions consécutives i et $i+1$ de l'opération A. Puisque les répétitions de l'opération A sont consécutives, alors il y a au moins un chemin de A_i à A_{i+1}. Si les deux répétitions n'appartiennent pas au même motif, on redéfinit le motif du graphe comme étant deux motifs consécutifs. Donc on peut définir une contrainte de latence entre les deux opérations puisqu'elles sont liées par au moins un chemin et qu'elles appartiennent au même motif. On considère $L_{A_iA_{i+1}} = T_A$ et l'égalité 2.18 devient : $s_{A_{i+1}} - s_{A_i} = L_{A_iA_{i+1}}$ qui est la définition de la latence (obtenue en changeant l'inégalité par l'égalité dans la définition de la latence). Le théorème est prouvé ⊡

En conséquence, on utilise le modèle proposé dans les deux sous-chapitres précédents. Pour rappeler ce modèle, on donne l'exemple suivant. Dans l'exemple on définit aussi des contraintes de latence.

Exemple 18 *Soit le graphe $\mathcal{G}$ donné dans la figure 2.39. En plus de la contrainte de périodicité de l'opération A avec la période T_A, on définit la contrainte de latence $L(A_2,C_1)$. On observe qu'à l'intérieur de la contrainte de latence il y a plusieurs répétitions de l'opération A. Puisqu'il y a des chemins entre A_2 et C_1 la présence de plusieurs répétitions d'une opération ne change en rien la définition de la contrainte de latence. Donc utiliser des répétitions finies à l'intérieur d'une contrainte de latence est possible grâce au fait que la contrainte de latence est définie pour des opérations liées au moins par un chemin.*

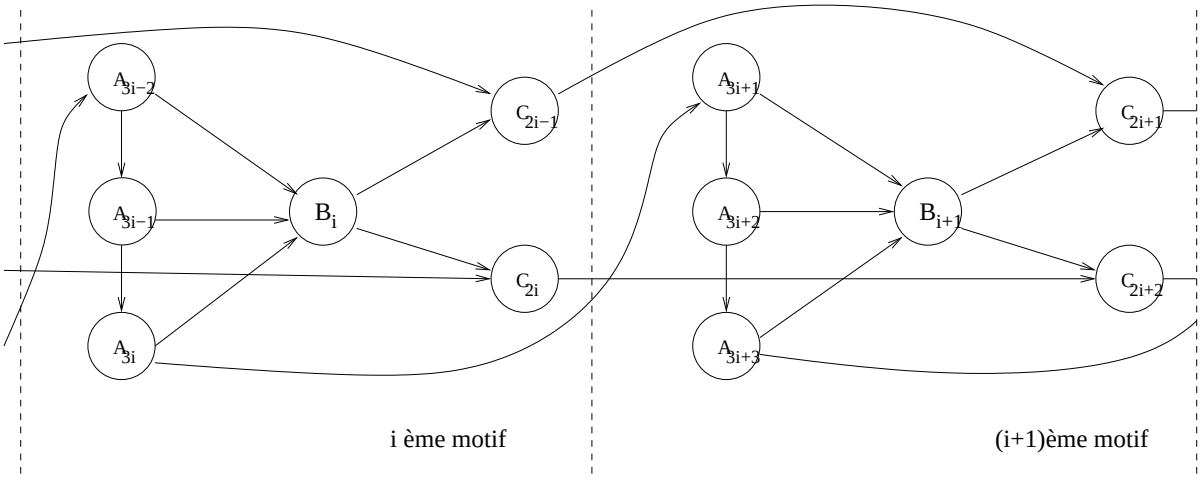

FIG. 2.39 – *Graphe non factorisé*

2.3.2 Condition d'ordonnançabilité

Le premier résultat généralise l'existence d'un motif dans l'ordonnancement pour les systèmes avec contraintes de précédences, de latences et de périodicités.

Théorème 23 *pour un système avec des contraintes de précédences, de périodicités et de latences on a*

$$S(s_{max}+iT,s_{max}+iT+t)=S(s_{max}+(i+1)T,s_{max}+(i+1)T+t),\forall 0<t\leq T \text{ et } i\geq 0$$

où s_{max} est la date de début de la dernière opération ordonnancée parmi les opérations du premier motif.

Preuve Evidemment le théorème est une conséquence du théorème 14 qui prouve l'existence d'un motif dans l'ordonnancement pour le cas des systèmes avec contraintes de précédences et de périodicités. La preuve du théorème 14 s'appuie sur l'héritage de périodes des opérations. Puisque les opérations doivent hériter des périodes afin que les contraintes de périodicité imposées au système soient satisfaites, l'héritage reste valable pour le cas des systèmes avec contraintes de périodicités et de latences. En plus, les latences modifient seulement l'ordre des opérations appartenant au premier motif du graphe. L'ordre des opérations des motifs suivants est imposé par les périodes des opérations. Donc le résultat du théorème 14 reste valable pour le cas des systèmes avec des contraintes de précédences, de périodicités et de latences. Le théorème est prouvé ⊡

Le théorème suivant prouve que les opérations périodiques appartenant à une contrainte de latence doivent avoir la même période afin que le système soit ordonnançable.

Théorème 24 *Soit un système d'opérations avec contraintes de précédences, de périodicités et de latences, où les précédences sont définies par un graphe $\mathcal{G}=(\mathcal{V},\mathcal{E})$. Pour une contrainte de latence (A,B), si les opérations appartenant à $\mathcal{M}(A,B)$ n'ont pas la même contrainte de périodicité alors la contrainte de latence n'est satisfaite pour aucune répétition du motif.*

Preuve On utilise un raisonnement par l'absurde. On suppose qu'il y au moins deux opérations C et $D\in\mathcal{M}(A,B)$ avec $T_C\neq T_D$.

Sans perdre de généralité on suppose que $T_C<T_D$. Puisque C et $D\in\mathcal{M}(A,B)$ en utilisant l'inégalité (2.1), on a $T_A<T_C$ et $T_D<T_B$ qui implique que $T_A<T_B$ est satisfaite.

On note par s_A^i respectivement s_B^i la date de début de l'opération A respectivement la date de début de l'opération B appartenant au i^{me} motif. On a $s_B^i-s_A^i+C_B=s_B^1+iT_B+s_A^1+iT_A+C_B$. Cela signifie que

$$lim_{i\rightarrow+\infty}s_B^i-s_A^i+C_B=+\infty$$

Donc, il n'y a pas de nombre $L_{AB}\in\mathbb{N}^*$ tel que $s_B^i-s_A^i+C_B,\forall i\geq 1$. La contrainte de latence n'est satisfaite pour aucune répétition du motif. Le théorème est prouvé ⊡

Si pour les contraintes de périodicité les résultats restent valables pour les systèmes avec contraintes de précédences, de latences et de périodicités, ce n'est plus le cas pour les résultats obtenus pour les contraintes de latence. Pour mieux expliquer on donne l'exemple suivant :

Exemple 19 *On donne ce contre-exemple pour prouver que la condition d'ordonnançabilité pour des systèmes d'opérations avec contraintes de précédences et latences n'implique plus que les contraintes de latence sont satisfaites pour des systèmes d'opérations avec contraintes de précédences, latences et périodicités. On considère le graphe donné dans la figure 2.40 avec les durées d'exécution* $C_A = 1, \forall A \in \mathcal{V}$ *et avec les contraintes de latence* $L_{AB} = 2$ *et* $L_{CD} = 2$. *On remarque que* (A,B) *et* (C,D) *sont en relation* Z.

D'abord, on considère le système sans contrainte de périodicité, qui est ordonnançable puisque $L_{AB} \geq C_A + C_B$ *et* $L_{CD} \geq C_C + C_D$. *En effet l'ordonnancement donné dans la figure 2.41 satisfait les contraintes de latence.*

On considère le système avec contraintes de périodicité $T_C = 3$, $T_B = 3$, $T_C = 6$ *et* $T_D = 6$. *Les inégalités* $L_{AB} \geq C_A + C_B$ *et* $L_{CD} \geq C_C + C_D$ *sont satisfaites mais il n'y a pas d'ordonnancement qui satisfait les contraintes de latences et de périodicités du système (voir figure 2.42).*

FIG. 2.40 – *Graphe illustrant le contre-exemple*

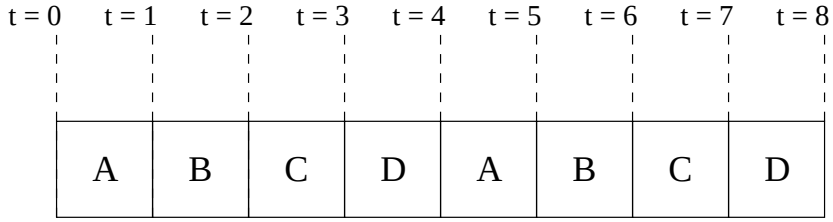

FIG. 2.41 – *Ordonnancement du graphe de la figure 2.40 sans contrainte de périodicité*

On rappelle que dans l'étude d'ordonnançabilité la relation X est la seule relation entre les latences qui intervient dans la condition d'ordonnançabilité. Le théorème suivant prouve que la relation X entre les latences implique que les périodes des opérations appartenant aux latences doivent être égales.

Théorème 25 *Soit un système d'opérations avec contraintes de précédences, de latences et de périodicités, où les précédences sont définies par un graphe* $\mathcal{G} = (\mathcal{V},\mathcal{E})$. *Soient* $L(A,B)$ *et*

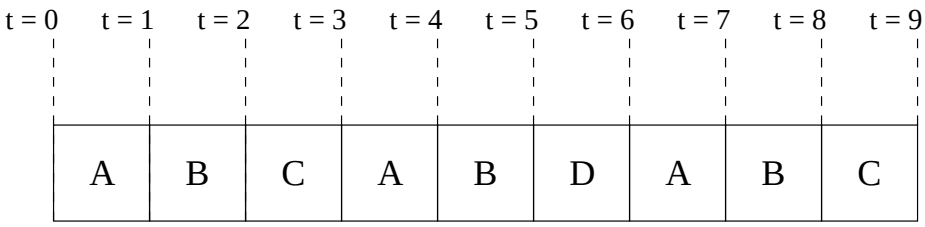

FIG. 2.42 – *Ordonnancement du graphe de la figure 2.40 avec contraintes de périodicité*

$L(C,D)$ deux contraintes de latence en relation X. *Si les opérations appartenant à $\mathcal{M}(A,B) \bigcup \mathcal{M}(C,D)$ n'ont pas la même contrainte de périodicité alors les contraintes de latence ne sont satisfaites pour aucune répétition du motif.*

Preuve En appliquant le théorème 24 puisque (A,B) et (C,D) ont des contraintes de latence, on obtient $T_A = T_B$ et $T_C = T_D$. De plus les deux latences sont en relation X donc il y a au moins un chemin de A à D et de C à B qui implique (inégalité (2.1)) $T_A \leq T_D$ et $T_C \leq T_B$. On obtient donc : $T_A = T_B = T_C = T_D$. En conséquence toutes les opérations ont la même période. Le théorème est prouvé ⊡

Le théorème central de ce sous-chapitre donne une condition d'ordonnançabilité pour qu'un système avec contraintes de précédences, de latences et de périodicités soit ordonnançable. Pour prouver ce résultat on se sert de trois lemmes. Le lemme 7 prouve que la contrainte de latence doit avoir une valeur comparable à celle de la période des opérations appartenant à la contrainte de latence. Le lemme 8 prouve que le temps écoulé entre la première opération et la dernière opération dans la contrainte de latence reste toujours le même pour toutes les répétitions des opérations. Le lemme 9 donne le nombre des répétitions d'une opération entre les dates de début de la première opération et de la dernière opération d'une contrainte de latence dans le cas où l'opération n'appartient pas à la contrainte de latence.

Lemme 7 *Soit un système d'opérations avec des contraintes de précédences, latences et périodicités, où les précédences sont définies par un graphe $\mathcal{G} = (\mathcal{V},\mathcal{E})$. Soit $L(A,B)$ une contrainte de latence. Si $\sum_{C \in \mathcal{M}(A,B)} > T_A$ alors le système n'est pas ordonnançable.*

Preuve On utilise un raisonnement par l'absurde. On suppose que $\exists S$ un ordonnancement qui satisfait toutes les contraintes du système. On note par $\mathcal{FR}$ l'ensemble des opérations appartenant à $\mathcal{M}(A,B)$ qui sont ordonnancées de s_A à $s_A + T_A$. A l'instant $s_A + T_A$, A doit être ordonnancée et après elle toutes les opérations appartenant à $\mathcal{FR}$. Cela implique que les opérations appartenant à $\mathcal{M}(A,B) \backslash \mathcal{FR}$ ne sont jamais ordonnancées et $s_B - s_A \rightarrow \infty$. Donc la contrainte de latence (A,B) n'est pas satisfaite. Cela est en contradiction avec le fait qu'il y a un ordonnancement S qui satisfait toutes les contraintes du système. Le lemme est prouvé ⊡

Lemme 8 *Soit un système d'opérations avec des contraintes de précédences, latences et périodicités, où les précédences sont définies par un graphe $\mathcal{G} = (\mathcal{V},\mathcal{E})$ et (A,B) une contrainte de latence. Pour un ordonnancement S on a $\delta_{A_i B_i}(S) = \delta_{A_{i+1} B_{i+1}}(S), \forall i \geq 1$.*

Preuve En utilisant le lemme 7, on a $\delta_{AB}(S) < T_A$ qui implique :

$$\begin{cases} \delta_{A_iB_i}(S) = T_A - s_{A_i+1} + s_{B_i} \\ \delta_{A_{i+1}B_{i+1}}(S) = T_B - s_{A_i+1} + s_{B_i} \end{cases}$$

Puisque A et B ont une contrainte de latence on a $T_A = T_B$. Cela implique que $\delta_{A_iB_i}(S) = \delta_{A_{i+1}B_{i+1}}(S), \forall i \geq 1$. Le lemme est prouvé ⊡

Lemme 9 *Soit un système d'opérations avec des contraintes de précédences, latences et périodicités, où les précédences sont définies par un graphe $\mathcal{G} = (\mathcal{V},\mathcal{E})$ et (A,B) une contrainte de latence. Les opérations avec une période égale à celle de A qui appartient à la contrainte de latence sont ordonnancées de s_A à s_B. Les opérations C avec une période plus petite que celle de A sont ordonnancées $\frac{L_{AB}}{T_C}$ fois de s_A à s_B.*

Preuve Pour un ordonnancement quelconque S la longueur de l'ordonnancement $\delta_{AB}(S)$ ne peut pas être plus grande que L_{AB}, à cause de la contrainte de latence $L(A,B)$ donc l'opération C se répète $\frac{L_{AB}}{T_C}$ fois de s_A à s_B. Le lemme est prouvé ⊡

Théorème 26 *Soit un système d'opérations avec des contraintes de précédences, latences et périodicités, où les précédences sont définies par un graphe $\mathcal{G} = (\mathcal{V},\mathcal{E})$. On note par $\mathcal{L}$ l'ensemble des contraintes de latence et par $\mathcal{P}$ l'ensemble des opérations périodiques. S'il y a un ordonnancement satisfaisant toutes les contraintes alors les relations suivantes sont satisfaites :*

- $\Sigma_{A\in\mathcal{V}}\frac{C_A}{T_A} \leq 1$
- $\forall A$ *tel que* $\exists B, T_B < T_A$, $\sum_{H\in\mathcal{V},T_H<T_A}\frac{L_{AB}}{T_H}C_H + C_A \leq T_A$
- $L_{AB} \geq \sum_{H\in\mathcal{W}_1} C_H + \sum_{H\in\mathcal{V},T_H<T_A} C_H\frac{L_{AB}}{T_H}, \forall L_{AB} \in \mathcal{L}$ *où :*
  $\mathcal{W}_1 = \mathcal{M}(A,B) \bigcup \bigcup_{i\in\{1,\ldots,j\};E\in lt_{AB}(C_i,D_i)} \mathcal{M}(C_i,E) \bigcup \bigcup_{i\in\{j+1,\ldots,k\};E\in ft_{AB}(C_i,D_i)} \mathcal{M}(E,D_i) \bigcup \bigcup_{i\in\{k+1,\ldots,m\}} \mathcal{M}(C_i,D_i),$
  où $j < k < m < n$ sont des nombres entiers positifs qui satisfont les relations suivantes :
  $L_{C_iD_i} \geq \sum_{H\in\{\mathcal{M}(C_i,D_i)\bigcup \bigcup_{E\in ft_{C_iD_i}(AB)} \mathcal{M}(E,B)\}} C_H, \forall i \in \{1,\ldots,j\}$
  $L_{C_iD_i} \geq \sum_{H\in\{\mathcal{M}(C_i,D_i)\bigcup \bigcup_{E\in lt_{C_iD_i}(A,B)} \mathcal{M}(A,E)\}} C_H, \forall i \in \{j+1,\ldots,k\}$
  $L_{C_iD_i} \geq \sum_{H\in\mathcal{M}(C_i,D_i)} C_H, \forall i \in \{k+1,\ldots,m\}$ *et* $L_{C_iD_i} \geq \sum_{H\in\{\mathcal{M}(C_i,D_i)\bigcup\mathcal{M}(A,B)\}} C_H, \forall i \in \{m+1,\ldots,n\}$

Preuve La première inégalité est impliquée par l'existence du motif dans l'ordonnancement. Les lemmes indiquent quelles sont les opérations qui sont répétées plusieurs fois entre la première et la dernière opération d'une contrainte de latence. Donc dans les inégalités obtenues lors de l'étude d'ordonnançabilité pour les systèmes avec contraintes de précédences et de latences, on ajoute ces opérations. On obtient de cette manière les autres inégalités. Le théorème est prouvé ⊡

2.3.3 Algorithme optimal d'ordonnancement

On note par $\mathcal{G} = (\mathcal{V},\mathcal{E})$ le graphe orienté acyclique des opérations où $\mathcal{V}$ est l'ensemble des opérations et $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ est l'ensemble des arcs et par $\mathcal{D}(A) = \{B \in \mathcal{V}$ tel que $\exists$ au moins un chemin orienté de A à $B\}$ l'ensemble des opérations liées par au moins un chemin de A.

L'algorithme d'ordonnancement utilise les marques obtenues après avoir appliqué l'algorithme de marquage 3. On note par $\mathcal{W}$ l'ensemble de travail, par s_T la date de début de la dernière opération ordonnancée, par C_T sa durée d'exécution, par $\mathcal{P}er$ l'ensemble des opérations qui ont une contrainte de périodicité et par $Prec(A)$ l'ensemble des prédécesseurs de l'opération A. Pendant le déroulement de l'algorithme, l'ensemble $\mathcal{W}$ contient les opérations dsponibles, c'est-à-dire les opérations dont les prédécesseurs sont déjà ordonnancés. On remarque qu'entre deux opérations prêtes à être ordonnancées dans le même temps, il ne peut pas y avoir un chemin. Aussi, pendant le déroulement de l'algorithme, chaque fois qu'une opération est ordonnancée, s_T (respectivement C_T) est remplacée par la date de début (la durée d'exécution) de cette opération.

On note par $\mathcal{L}_X$ l'ensemble des latences qui sont en relation X avec d'autres contraintes de latence dont la différence entre la valeur de la latence et la somme des durées d'exécution des opérations appartenant à la paire n'est pas égale à 0. On considère que cet ensemble est ordonné c'est-à-dire $L(A,B)$ précède dans l'ensemble toute contrainte de latence $L(C,D)$ dont la différence entre la valeur de la latence et la somme des durées d'exécution des opérations appartenant à la paire est plus grande que celle de la contrainte $L(A,B)$.

L'algorithme comporte deux parties. Une première partie (Pas 1 au Pas 5) ordonnance les opérations avant que la première répétition d'une opération périodique soit ordonnancée. Cette partie correspond à un régime transitoire pendant lequel aucune opération périodique n'est ordonnancée et on ordonnance d'abord les opérations sans aucune contrainte, puis les opérations importantes pour une contrainte de latence et finalement une opération périodique. Une fois qu'une opération périodique est ordonnancée, on passe au régime permanent. Cette deuxième partie (Pas 6 au Pas 9) ordonnance les opérations périodiques d'abord, ensuite les opérations importantes pour une contrainte de latence et finalement les opérations sans aucune contrainte.

Algorithme 5

Initialisation : $\mathcal{W} = \bigcup_{A\in\mathcal{V} \ et \ Prec(A)=\emptyset}\{A\}$ et $s_T = 0, C_T = 0$.

Pas 1 *(opération avec contrainte de latence)*:
si $\exists A \in \mathcal{W}$ tel que $marque(A) \neq \infty$ alors on choisit A tel que $marque(A) =$

$min_{B\in\mathcal{W}}\{marque(B)\}$. Si $\exists L(C_1,D_1) \in \mathcal{L}_X$ et $\exists L(C_i,D_i) \in \mathcal{L}_X, \forall i = 2,3,\ldots n$ tel que $A \in \bigcup_{E\in\Gamma^+_{C_iD_i}(C_1,D_1)} \mathcal{M}(E,D_1), \forall i = 2,3,\ldots n$ alors soit $B \in \mathcal{W}$ l'opération avec la plus petite marque tel que $\exists i \in \{2,3,\ldots,n\}$ avec $B \in \bigcup_{E\in\Gamma^+_{C_1D_1}(C_i,D_i)} \mathcal{M}(E,D_i)$. Si l'opération B existe alors $s_B = s_T + C_T$, on enlève B de $\mathcal{W}$, toutes les opérations qui deviennent ordonnançables sont ajoutées à $\mathcal{W}$ et on va au Pas 4. Si l'opération B n'existe pas alors $s_A = s_T + C_T$, on enlève A de $\mathcal{W}$, toutes les opérations qui deviennent ordonnançables sont ajoutées à $\mathcal{W}$ et on va au Pas 5.

Pas 2 *(opération sans contrainte de périodicité qui n'est pas première d'une contrainte de latence)*:
si $\exists A \in \mathcal{W}$ tel qu'il n'y pas d'opération $B \in \mathcal{V}$ avec $(A,B) \in \mathcal{L}$ et $A \notin \mathcal{P}er$ alors $s_A = s_T + C_T$, on l'enlève de $\mathcal{W}$, toutes les opérations qui deviennent ordonnançables sont ajoutées à $\mathcal{W}$ et on va au Pas 1.

Pas 3 *(opération qui n'est pas première d'une contrainte de latence)*:
si $\exists A \in \mathcal{W}$ tel qu'il y a $B \in \mathcal{V}$ avec $(A,B) \in \mathcal{L}$ alors $s_A = s_T + C_T$ avec $L(A,B) = max_{(C,D)\in\mathcal{L}\ avec\ C\in\mathcal{W}}\{L(C,D)\}$, on l'enlève de $\mathcal{W}$, toutes les opérations qui deviennent ordonnançables sont ajoutées à $\mathcal{W}$ et on va au Pas 5.

Pas 4 *(opération avec contrainte de périodicité pour laquelle la première répétition n'est pas ordonnancée)*:
on a $s_A = s_T + C_T$ tel que $T_A = min_{C\in\mathcal{W}\bigcap\mathcal{P}er\ \ avec\ C_0\ \ non\ ordonnancée}\{T_C\}$, on enlève A de $\mathcal{W}$, toutes les opérations qui deviennent ordonnançables sont ajoutées à $\mathcal{W}$ et on va au Pas 6.

Pas 5: si $\nexists A \in \mathcal{W}\bigcap\mathcal{P}er$ avec A_1 déjà ordonnancée alors on va au Pas 1.

Pas 6: on cherche une opération $A \in \mathcal{W}\bigcap\mathcal{P}er$ pour laquelle la première répétition A_1 est ordonnancée et on a

$$s_{A_i} + T_A - s_T - C_T = min_{B_i\in\mathcal{W}\bigcap\mathcal{P}er\ \ et\ B_1\ \ déjà\ ordonnancée}\{s_{B_i} + T_B - s_T - C_T\}$$

où A_i est la dernière répétition ordonnancée de A. Si on trouve plusieurs opérations qui satisfont les conditions satisfaites par A alors le système n'est pas ordonnançable et l'algorithme s'arrête.

Pas 7 *(opération avec contrainte de latence)*:
si $\exists C \in \mathcal{W}\backslash\{\{A\}\bigcap P\}$ tel que $s_T + C_T + C_C \leq s_{A_i} + T_A$ avec A l'opération trouvée au Pas 6 et $marque(C) \neq \infty$, alors on a $s_C = s_T + C_T$ avec : $marque(C) =$
$min_{B\in\mathcal{W}}\{marque(B)\}$. Si $\exists L(C_1,D_1) \in \mathcal{L}_X$ et $\exists L(C_i,D_i) \in \mathcal{L}_X, \forall i = 2,3,\ldots n$ tel que $A \in \bigcup_{E\in\Gamma^+_{C_iD_i}(C_1,D_1)} \mathcal{M}(E,D_1), \forall i = 2,3,\ldots n$ alors soit $B \in \mathcal{W}$ l'opération avec la plus petite marque telle que $\exists i \in \{2,3,\ldots,n\}$ avec $B \in \bigcup_{E\in\Gamma^+_{C_1D_1}(C_i,D_i)} \mathcal{M}(E,D_i)$. Si l'opération B existe alors $s_B = s_T + C_T$, on enlève B de $\mathcal{W}$, toutes les opérations qui deviennent ordonnançables

sont ajoutées à $\mathcal{W}$ et on va au Pas 6. Si l'opération B n'existe pas alors $s_C = s_T + C_T$, on enlève C de $\mathcal{W}$, toutes les opérations qui deviennent ordonnançables sont ajoutées à $\mathcal{W}$ et on va au Pas 6.

Pas 8 *(opération avec contrainte de périodicité pour laquelle la première répétition n'est pas ordonnancée)*:
si $\exists C \in \mathcal{W} \bigcap \mathcal{P}er$ avec C_1 ordonnancée et $s_T + C_T + C_C \leq s_{A_i} + T_A$ avec A l'opération trouvée au Pas 6, alors on a $s_C = s_T + C_T$ avec $T_C = min_{D \in \mathcal{W} \bigcap \mathcal{P}er \ et \ C_D \leq T_A}\{T_D\}$, et avec C ayant la plus grande durée d'exécution, on enlève l'opération de $\mathcal{W}$, toutes les opérations qui deviennent ordonnançables sont ajoutées à $\mathcal{W}$ et on va au Pas 6.

Pas 9 *(opération avec contrainte de périodicité)*:
on a $s_A = s_T + C_T$ tel que $s_{A_{i+1}} = s_{A_i} + T_A$, on enlève l'opération de $\mathcal{W}$, toutes les opérations qui deviennent ordonnançables sont ajoutées à $\mathcal{W}$ et on va au Pas 6.

Avant de prouver l'optimalité de l'algorithme d'ordonnancement 5 on donne un exemple d'ordonnancement obtenu à l'aide de l'algorithme.

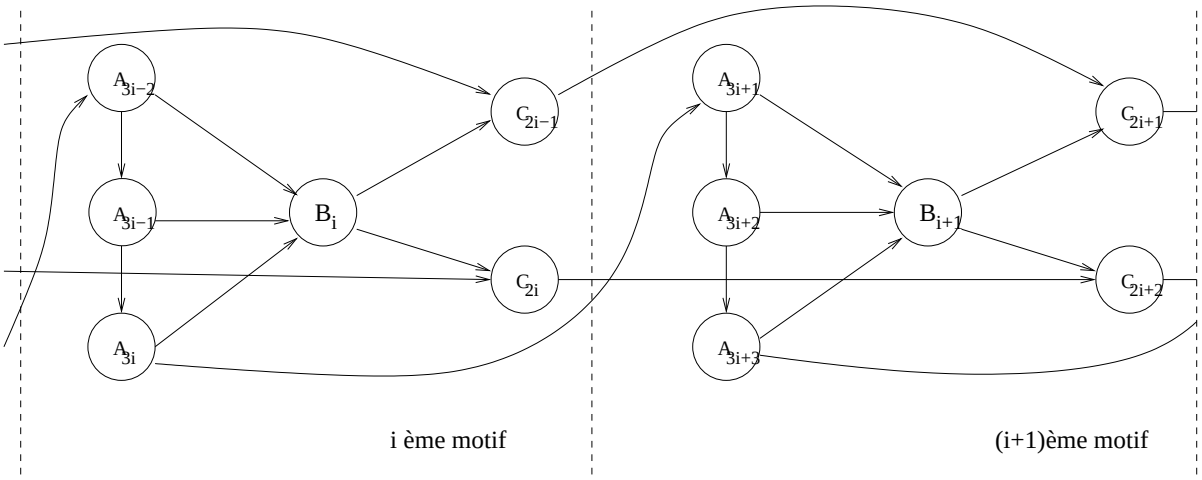

FIG. 2.43 – *Graphe illustrant l'utilisation de l'algorithme d'ordonnancement 5*

Exemple 20 *Soit $\mathcal{G} = (\mathcal{V},\mathcal{E})$ le graphe donné dans la figure 2.43 avec $C_A = 2$, $C_B = 1$ et $C_C = 2$, les périodes $T_A = 5$ et $T_B = 15$ et les contraintes de latence $L(A_{3i-1},C_{2i}) = 10$ et $L(B_i,C_{2i-1}) = 9$. Après avoir appliqué l'algorithme de marquage 3 on obtient le tableau 2.5, où on utilise $m(A)$ pour désigner $marque(A)$.*

Après avoir appliqué l'algorithme d'ordonnancement 5, on obtient le tableau 2.6.

Remarque 18 *Pour la preuve du théorème 27, on note par k_1 les opérations A avec $marque(A) \neq \infty$, par k_2 les opérations A avec $marque(A) = \infty$ et il n'y pas d'opération $B \in \mathcal{V}$ avec*

	$m(A_{3i-2})$	$m(A_{3i-1})$	$m(A_{3i})$	$m(B_i)$	$m(C_{2i-1})$	$m(C_{2i})$
Initialisation	∞	∞	∞	∞	9	10
$L(B_i,C_{2i-1})$	∞	∞	∞	∞	9	10
$L(A_{3i-1},C_{2i})$	∞	∞	10	10	9	10

TAB. 2.5 – *Marques obtenues après avoir utilisé l'algorithme de marquage 3*

$(A,B) \in \mathcal{L}$ *et* $A \notin \mathcal{P}er$, *par* k_3 *les opérations* A *avec* $marque(A) = \infty$ *et* $\exists B \in \mathcal{V}$ *avec* $(A,B) \in \mathcal{L}$ *et par* k_4 *les opérations* A *avec* $marque(A) = \infty$ *et* $A \in \mathcal{P}er$, *des opérations à ordonnancer pendant l'algorithme.*

Théorème 27 *L'algorithme d'ordonnancement 5 est optimal (s'il y a un ordonnancement, l'algorithme le trouvera).*

Preuve Au début de l'ordonnancement on a deux possibilités :

1. il y a des opérations du type k_1 parmi les opérations disponibles. Une opération avec *marque* $\neq \infty$ peut avoir ou non des contraintes de périodicité. La contrainte de périodicité doit être satisfaite seulement après avoir ordonnancée la première répétition d'une opération avec contrainte de périodicité. Une fois que le Pas 1 ordonnance la première répétition d'une opération avec contrainte de périodicité, les opérations disponibles vont être ordonnancées en passant par les Pas 7, 8 et 9. En conséquence, avant que la répétition d'une opération avec contrainte de périodicité soit ordonnancée, on passe seulement par le Pas 1 pour ordonnancer les opérations. On a deux possibilités :

   (a) aucune première répétition d'une opération avec contrainte de périodicité n'a été encore ordonnancée. Cela signifie qu'il y a seulement des contraintes de latence à satisfaire et on passe seulement par le Pas 1, dont le choix est optimal pour satisfaire les contraintes de latence (Théorème 21).

   (b) la répétition d'une opération avec contrainte de périodicité a été déjà ordonnancée, donc il y a une contrainte de périodicité à satisfaire. Après avoir calculé la date de début de l'opération A telle que

$$s_{A_i} + T_A - s_T - C_T = min_{B_i \in \mathcal{W} \bigcap \mathcal{P}er \ \ et \ B_1 \ \ déjà \ ordonnancée}\{s_{B_i} + T_B$$

   $-s_T - C_T\}$ (Pas 6), les opérations qui restent sont ordonnancées en passant par les Pas 7 et 8. Les Pas 7 et 8 vont choisir les opérations qui peuvent être ordonnancées avant la date de début de l'opération A choisie. Puisqu'on a déjà choisi la contrainte de périodicité devant être satisfaite, on a seulement des opérations avec des contraintes de latence à satisfaire, dont des opérations périodiques avec la première répétition non-ordonnancées. Ainsi on ordonnance seulement des opérations en prenant en compte les trois relations entre les paires d'opérations sur lesquelles des contraintes de latence ont été définies. En conséquence toutes les

$\mathcal{W}$	s_T	C_T	Pas utilisé	Décision
$\{A_1\}$	0	0	Pas 4	$s_{A_1} = 0$
$\{A_2\}$	0	2	Pas 6	A choisi
$\{A_2\}$	0	2	Pas 9	$sA_2 = 5$
$\{A_3\}$	5	2	Pas 6	A choisi
$\{A_3\}$	5	2	Pas 9	$s_{A_3} = 10$
$\{A_1,B\}$	10	2	Pas 6	A choisi
$\{A_1,B\}$	10	2	Pas 8	$s_B = 12$
$\{A_1,C_1,C_2\}$	12	1	Pas 6	A choisi
$\{A_1,C_1,C_2\}$	12	1	Pas 7	$s_{C_2} = 13$
$\{A_1,C_2\}$	13	2	Pas 6	A choisi
$\{A_1,C_2\}$	13	2	Pas 9	$s_{A_1} = 15$
$\{A_2,C_2\}$	15	2	Pas 6	A choisi
$\{A_2,C_2\}$	15	2	Pas 7	$s_{C_1} = 17$
$\{A_2\}$	17	2	Pas 6	A choisi
$\{A_2\}$	17	2	Pas 9	$s_{A_2} = 20$
$\{A_3\}$	20	2	Pas 6	A choisi
$\{A_3\}$	20	2	Pas 9	$s_{A_3} = 25$
$\{A_1,B\}$	25	2	Pas 6	B choisi
$\{A_1,B\}$	25	2	Pas 9	$s_B = 27$
$\{A_1,C_1,C_2\}$	27	1	Pas 6	A choisi
$\cdots$				

TAB. 2.6 – *Ordonnancement obtenu en utilisant l'algorithme 5*

répétitions périodiques futures de ces opérations vont satisfaire les contraintes de latence, tout en satisfaisant les contraintes de périodicité.
S'il y a aucune opération avec *marque* $\neq \infty$ qui peut être ordonnancée avant la date de début déjà calculée alors les opérations restantes peuvent être ordonnancées dans n'importe quel ordre. Finalement, au Pas 9 on ordonnance les opérations A avec la date de début déjà calculée.

2. il n'y aucune opération du type k_1 parmi les opérations disponibles. Cela implique que l'ordonnancement est obtenu en passant par les Pas 2, 3 et 4 avant qu'une opération avec une contrainte de périodicité et/ou du type k_1 devienne disponible. Le Pas 3 est utilisé seulement s'il n'y a pas d'opération du type k_2. Une fois que le Pas 3 ordonnance une opération, les opérations du type k_1 deviennent disponibles et on ne va jamais passer par le Pas 4. Si le Pas 4 ordonnance une opération alors seulement les Pas 7, 8 et 9 vont ordonnancer les opérations restantes. Donc les Pas 2, 3 et 4 ordonnancent soit des opérations du type k_2 avant qu'ils ordonnancent des opérations du type k_3, soit des opérations du type k_2 avant qu'ils ordonnancent des opérations du type k_4. Dans 1.(b) on prouve que l'ordre dans lequel on ordonnance les opérations du type k_2 et k_3

n'est pas important. Une opération du type k_2 n'a aucune contrainte à satisfaire et une opération du type k_4 doit satisfaire sa contrainte de périodicité une fois sa première répétition ordonnancée. En conséquence, pour les opérations du type k_2 et k_4 on réduit le nombre des opérations qui doivent être ordonnancées pour satisfaire des contraintes de périodicité. Une fois que les Pas 3 et 4 ont été utilisés on se trouve dans les cas 1.(a) et 1.(b). Le théorème est prouvé ⊡

Après avoir présenté les résultats concernant notre problème d'ordonnancement, on utilise le modèle temps réel classique et on étudie les conséquences des résultats obtenus dans ce chapitre sur le modèle classique.

2.4 Cas particulier de système temps réel avec dates de réveil et échéances

Dans ce chapitre on particularise les résultats obtenus en utilisant notre modèle pour le problème d'ordonnancement sans préemption des systèmes temps réel utilisant le modèle proposé par Liu et Layland [4]. Cela nous permet de retrouver des résultats déjà connus qui prouvent la justesse de notre approche, mais aussi la généralité de notre modèle.

Le chapitre commence par rappeler les deux modèles, en soulignant leurs différences. On donne un théorème qui prouve un lien entre la contrainte de latence et la contrainte d'échéance. Le deuxième sous-chapitre formule le problème avec échéances absolues comme un cas particulier de notre modèle. Cette particularité permet de donner un algorithme d'ordonnancement prouvé optimal et des conditions d'ordonnançabilité pour ce problème avec dates de réveil et échéances. De la même manière on prouve des résultats pour le problème avec dates de réveil et échéances.

2.4.1 Différences entre le modèle classique et notre modèle

Pour un ordonnancement S qui satisfait toutes les contraintes du système, on rappelle que s_A est la date de début de l'opération A dans l'ordonnancement S.

Dans le modèle introduit par Liu et Layland [4] une opération A possède une date de réveil r_A à laquelle l'opération devient disponible, une échéance D_A définie par rapport à la date de réveil, une période T_A et une durée d'exécution C_A. Si le premier réveil de l'opération se fait à r_A, le $i^{\text{ème}}$ réveil se fait à $r_A + (i-1)T_A$, où T_A est la période de l'opération. On considère l'ordonnancement sans préemption.

Si les opérations ne possèdent pas de dates de réveil, alors les échéances sont définies par rapport au début de l'ordonnancement. Ce type d'échéance s'appelle échéance absolue.

Dans notre modèle, une opération possède une période T_A et une durée d'exécution C_A. Les opérations ne possèdent pas de dates de réveil et la période impose le temps écoulé entre deux ordonnancements successifs d'une opération. Si la première date de début d'une opération est s_A, alors le $i^{\text{ème}}$ ordonnancement se fait à $s_A + (i-1)T_A$, où T_A est la période de l'opération. On considère l'ordonnancement sans préemption.

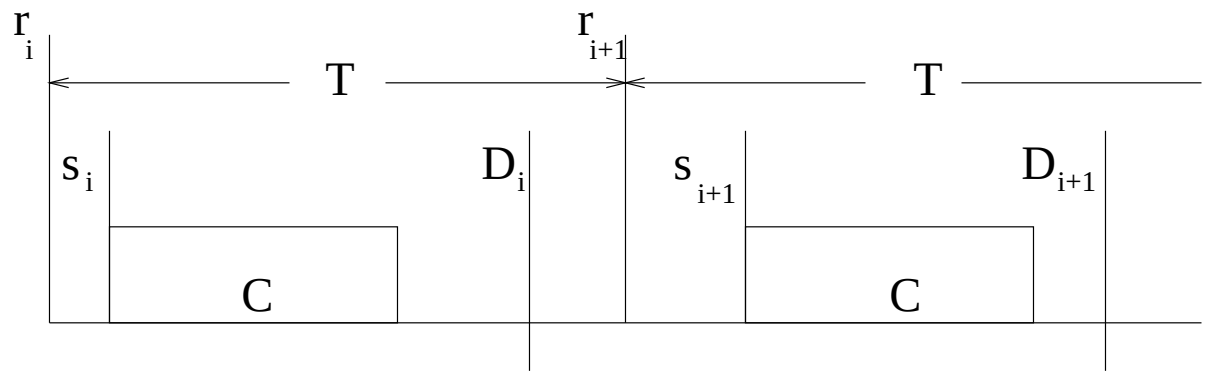

FIG. 2.44 – *Modèle proposé par Liu et Layland*

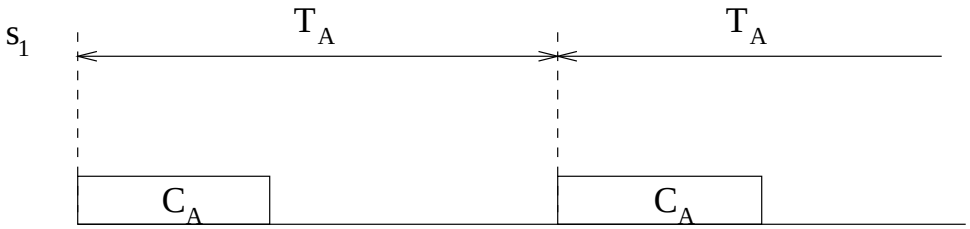

FIG. 2.45 – *Notre modèle*

2.4.2 Problème avec échéances absolues

Théorème 28 *La contrainte de latence $L(A,B)$ est équivalente à une échéance absolue de l'opération B une fois que A est ordonnancée.*

Preuve On prouve par double implication. Tout d'abord on prouve l'implication directe. On prouve que la contrainte de latence $L(A,B)$ peut être exprimée comme une échéance absolue de l'opération B une fois que A est ordonnancée.

Puisque (A,B) a une contrainte de latence $L(A,B)$ alors on a $s_B \leq L(A,B) + s_A - C_B$, pour un ordonnancement quelconque qui satisfait les contraintes du système. On note par $L(A,B) + s_A - C_B$ par D_B et on obtient :

$$s_B \leq D_B \tag{2.19}$$

Une fois que A est ordonnancée s_A est connue et D_B devient connu, aussi. Puisque s_A est calculée par rapport au début de l'ordonnancement alors l'inégalité (2.19) définit une échéance absolue pour B.

On passe à l'implication inverse. On prouve que l'échéance absolue D_B d'une opération B peut être exprimée comme une contrainte de latence $L(\cdot,B)$. Puisque B a une échéance absolue D_B alors $s_B \leq D_B$ pour un ordonnancement quelconque S qui satisfait les contraintes du système. On note par A la première opération de l'ordonnancement S. Puisque l'échéance de B est définie par rapport au début de l'ordonnancement on a $s_B - s_A \leq D_B$. En plus,

puisque toutes les opérations C sans prédécesseur peuvent être ordonnancées en premier dans l'ordonnancement, on a $s_B - s_C \leq D_B, \forall C \in \mathcal{V}$ avec $B \in \mathcal{D}(C)$. En conséquence afin d'exprimer l'échéance de B on définit plusieurs contraintes de latence $L(C,B) = D_B$ pour chaque C telle que $B \in \mathcal{D}(C)$. Le théorème est prouvé $\boxdot$

Afin d'ordonnancer ce type de système d'opérations il suffit de modifier l'algorithme de marquage qui doit prendre en compte les échéances absolues et non pas les latences. On obtient donc l'algorithme suivant :

Algorithme 6

Initialisation: $\mathcal{W} = \bigcup_{A \in \mathcal{V} \ et \ Suc(A)=\emptyset} Prec(A)$ est l'ensemble de travail. Si A a une échéance D_A, alors $marque(A) = D_A$, sinon $marque(A) = \infty$.

Pas 1: Pour $A \in \mathcal{W}$,$marque(A) = min(marque(A), min_{B \in Suc(A)}\{marque(B)\})$ et $\mathcal{W} = \mathcal{W} \backslash \{A\}$. On ajoute à $\mathcal{W}$ les opérations dont les prédécesseurs ont été tous enlevés de $\mathcal{W}$.

Pas 2: On répète Pas 1 jusqu'à $\mathcal{W} = \emptyset$.

Remarque 19 *D'une manière récursive les opérations héritent des échéances de leurs successeurs. A la fin de l'algorithme 6, la marque d'une opération est égale soit à l'échéance propre de l'opération, soit à l'échéance héritée de l'opération. En conséquence l'algorithme ne modifie pas les marques des opérations sans successeur. Evidemment comme dans le cas de l'algorithme de marquage pour les latences, l'algorithme 6 est appliqué sur le motif.*

L'exemple suivant donne une application de l'algorithme 6 pour un système d'opérations avec contraintes de précédences et des échéances absolues.

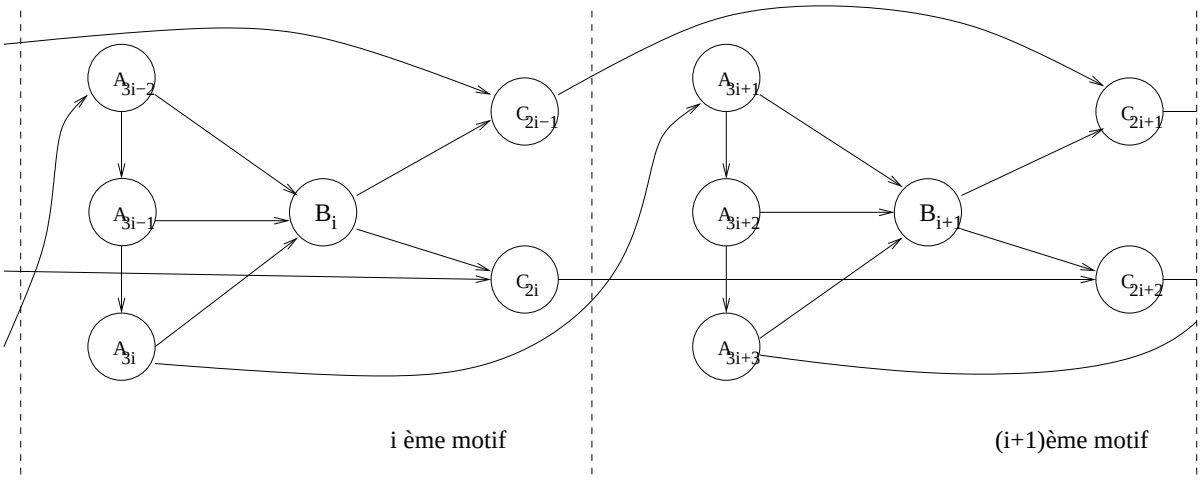

FIG. 2.46 – *Graphe illustrant l'utilisation de l'algorithme 6*

Exemple 21 *Soit un graphe $\mathcal{G} = (\mathcal{V},\mathcal{E})$ donné dans la figure 2.46 avec l'échéance absolue D_{C_1}. Toutes les opérations vont avoir la marque D_{C_1} sauf C_2. Le déroulement de l'algorithme 6 est donné dans le tableau 2.7, où on utilise $m(A)$ pour désigner marque(A).*

	$m(A_1)$	$m(A_2)$	$m(A_3)$	$m(B)$	$m(C_1)$	$m(C_2)$
Initialisation	∞	∞	∞	∞	D_{C_1}	∞
D_{C_1}	D_{C_1}	D_{C_1}	D_{C_1}	D_{C_1}	D_{C_1}	D_{C_1}

TAB. 2.7 – *Marques obtenues après avoir utilisé l'algorithme de marquage*

2.4.3 Problème avec dates de réveil et échéances

Pour un problème quelconque avec dates de réveil et échéances, on formule un problème d'ordonnancement en utilisant notre modèle et on prouve que ce problème est équivalent au premier problème avec dates de réveil et échéances.On considère un système avec n opérations et pour chaque opération A on a la durée d'exécution C_A, la date de réveil r_A et l'échéance par rapport à la date de réveil D_A. L'opération A est périodique avec la période T_A. On construit un nouveau problème d'ordonnancement en utilisant l'algorithme suivant :

Algorithme 7

Pas 1 : pour chaque opération A on ajoute une opération fictive A' de durée nulle.

Pas 2 : on ajoute une contrainte de précédence entre A' et A. On obtient ainsi un graphe de précédences qui a $2n$ opérations et n arcs.

Pas 3 : on impose des contraintes de latence pour chaque paire (A',A) avec $L_{A'A} = D_A$.

Pas 4 : on impose des contraintes de périodicité pour chaque opération A' avec T'_A égale à la période de A dans le problème d'origine.

On donne un exemple de construction du problème en utilisant l'algorithme 7.

Exemple 22 *Pour un système avec trois opérations A, B et C, on obtient le graphe de précédences donné dans la figure 2.47.*

Théorème 29 *Le problème de décision associé à ce problème d'ordonnancement d'origine a la réponse oui si et seulement si le problème de décision associé au problème d'ordonnancement obtenu en utilisant l'algorithme 7 a la réponse oui.*

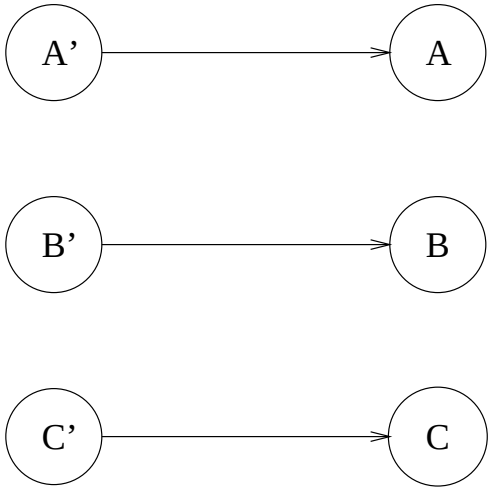

FIG. 2.47 – *Graphe obtenu après avoir utilisé l'algorithme 7*

Preuve On prouve le théorème par double implication.

On note par PRE le problème de décision associé au problème d'ordonnancement d'origine : chaque opération A possède une date de réveil r_A auquelle l'opération devient disponible, une échéance D_A définie par rapport à la date de réveil, une période T_A et une durée d'exécution C_A. On note par $PPLc$ le problème de décision associé au problème d'ordonnancement construit à partir de PRE en utilisant l'algorithme 7.

On suppose que le problème PRE a la réponse oui et on prouve que le problème $PPLc$ a aussi la réponse oui. On note par S un des ordonnancement qui satisfait toutes les contraintes d'une instance du problème PRE. Cela implique que pour chaque opération A on a la date de début $s_A \in S$ telle que $r_A \leq s_A \leq D_A$.

A partir de l'ordonnancement S, on construit un ordonnancement S' pour le problème $PPLc$. Chaque opération A' associée à une opération A de l'instance du problème PRE a la date de début de la première répétition $s'_{A'_1}$ égale à r_A. Chaque opération A a la date de début égale à la date de début de l'opération A dans l'ordonnancement S. La figure 2.48 illustre cette construction. L'opération A' ayant la durée nulle est marquée en pointillé sur la figure. En conséquence comme toutes les contraintes d'échéances sont satisfaites dans l'ordonnancement S alors toutes les contraintes de latence sont satisfaites dans l'ordonnancement S'. Etant donné que les opérations A' ont une durée nulle et leur date de début est périodique due à la périodicité de la date de réveil dans l'ordonnancement S, alors toutes les contraintes de périodicité sont satisfaites dans l'ordonnancement A'.

L'implication inverse construit de la même manière à partir d'un ordonnancement pour le problème $PPLc$ un ordonnancement pour le problème PRE. En conséquence, on a au moins un ordonnancement pour le problème d'origine si et seulement si le problème construit à l'aide de l'algorithme 7 a aussi un ordonnancement. Le théorème est prouvé ⊡

On a exprimé le problème avec dates de réveil et échéances à l'aide de notre modèle. Les contraintes de latence sont toutes en relation ||, donc leurs existence ne rend pas le

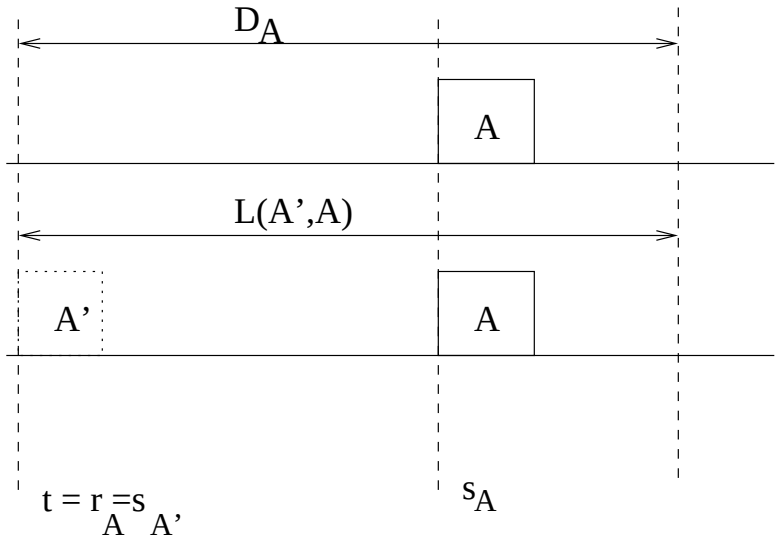

FIG. 2.48 – *Construction d'un ordonnancement pour PPLc à partir d'un ordonnancement PRE*

problème d'ordonnancement plus difficile. Puisque les opérations périodiques ont des durées nulles, la contrainte d'avoir des séquences divisibles pour les périodes n'est plus nécessaire. En conséquence on obtient un problème d'ordonnancement de systèmes avec contraintes de précédences, de périodicités et de latences pour lequel on peut utiliser tous les résultats obtenus dans ce chapitre.

Chapitre 3

Ordonnancement et distribution pour systèmes avec contraintes de précédences, de périodicités et de latences dans le cas multiprocesseur

Dans le cas multiprocesseur on n'étudie pas seulement un problème d'ordonnancement mais aussi un problème de distribution car il s'agit de choisir sur quelles ressources sont distribuées (allouées) les opérations et comment les opérations distribuées sur chaque ressource sont ordonnancées. Cette distribution et cet ordonnancement peuvent être réalisés en même temps, ou bien ils peuvent être réalisés l'un après l'autre. Dans la suite nous réalisons la distribution et l'ordonnancent en même temps. Cependant il faut noter qu'il n'existe aucun résultat prouvant que l'autre possibilité est meilleure. Comme la distribution et l'ordonnancement correspondent en réalité à ce qui est habituellement appelé "implantation", pour simplifier le discours on parle par la suite de "problème d'implantation" pour désigner le problème de l'ordonnancement et de la distribution. De même on utilisera le terme d'opérateur plutôt que celui de processeur afin de rester cohérent avec la notion d'opération. Ainsi on parlera d'implanter une opération sur un opérateur.

Ce chapitre traite le problème d'implantation de systèmes avec contraintes de précédences, de périodicités et de latences dans le cas multiprocesseur. Dans ce cas on prouve que notre problème d'implantation est NP-difficile et en conséquence les implantations proposées ne sont plus cherchées à l'aide d'algorithmes, mais à l'aide d'heuristiques. On rappelle qu'un algorithme donne une solution pour chaque instance du problème à résoudre et une heuristique donne une solution seulement pour certaines instances du problème à résoudre.

Dans le cas multiprocesseur on propose des heuristiques pour les trois problèmes d'implantation suivants :

- problème d'implantation avec contraintes de périodicités et précédences ;
- problème d'implantation avec contraintes de latences et précédences ;
- problème d'implantation avec contraintes de périodicités, de latences et précédences.

On rappelle que dans le cas monoprocesseur traité au chapitre 2, les deux premiers algorithmes proposés étaient des cas particuliers d'un troisième algorithme résolvant le cas général d'ordonnancement de système avec contraintes de précédences, de périodicités et de latences. Dans le cas multiprocesseur, les deux premières heuristiques ne sont pas des cas particuliers de l'heuristique dans le cas général. Chaque heuristique prend en compte les particularités des contraintes imposées aux systèmes et elle les utilise pour augmenter les chances de trouver une implantation qui satisfait les contraintes.

Les performances de chaque heuristique sont comparées à celles d'un algorithme exact. Bien que les heuristiques prennent en compte les temps de communication et qu'elles soient données pour des architectures hétérogènes, afin de simplifier le problème et sans que cela change les conclusions, les études de performances sont faites sur des cas avec des temps de communication nuls et des architectures homogènes.

Les trois problèmes d'implantation prennent en compte les données et ils utilisent des extensions des modèles présentés dans le chapitre 2. Ces extensions s'appliquent au modèle proposé dans les travaux sur la méthodologie AAA/SynDEx [87], dont cette thèse-même est la continuation. Les heuristiques présentées ici s'inspirent de l'heuristique proposée dans AAA/SynDEx par l'équipe OSTRE de l'INRIA Rocquencourt. Pour cette raison, on débute ce chapitre en rappelant le modèle et l'heuristique AAA/SynDEx qui traite le problème d'ordonnancement de systèmes avec contraintes de précédence et une contrainte unique de périodicité égale à la latence.

3.1 Contraintes de précédence et contrainte unique de périodicité égale à la latence

3.1.1 Modèle

Les systèmes temps réel modélisés dans la méthodologie AAA/SynDEx sont des systèmes qui n'ont pas de contrainte à satisfaire, en revanche le problème d'ordonnancement et de distribution possède un critère d'optimalité : minimiser la *cadence* du système égale au temps écoulé entre la première opération ordonnancée et la dernière opération ordonnancée du motif. Ce dernier temps correspond à une latence que l'on cherche ici à minimiser comme la cadence. En conséquence une opération est répétée d'une manière périodique avec la période appelée cadence égale à la latence, mais cette cadence n'est pas une donnée du problème d'implantation, elle est déterminée par une heuristique qui la minimise. Après minimisation on peut vérifier si la contrainte unique de cadence égale à la latence est vérifiée.On rappelle que dans cette méthodologie il y a aussi des contraintes de précédence.

Afin de mieux expliquer la cadence et la latence utilisées dans le modèle AAA/SynDEx on présente dans l'exemple donné figure 3.1 un système de quatre opérations de durée d'exécution égale à 1 à implanter sur un seul opérateur. Evidemment dans ce cas la latence est égale à la somme des durées d'exécution de toutes les opérations.

Un des deux ordonnancements possibles est donné dans la figure 3.2. Toutes les opérations sont répétées à une cadence égale à la latence et ont donc la même période.

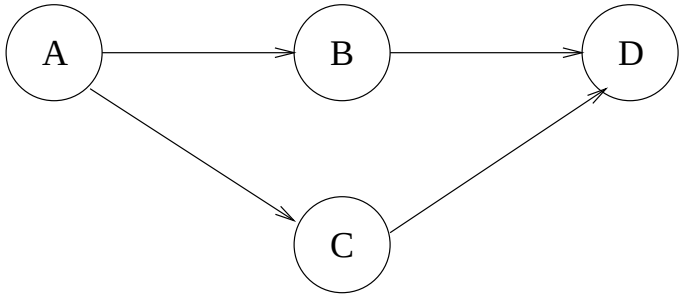

FIG. 3.1 – *Graphe d'algorithme*

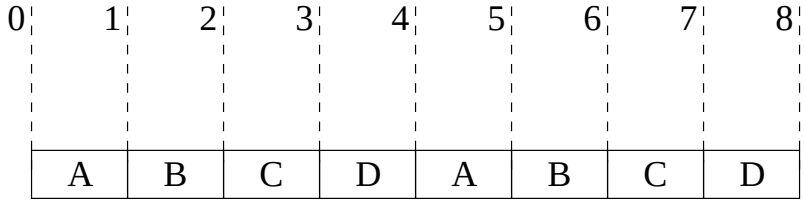

FIG. 3.2 – *Ordonnancement pour des opérations répétées à une cadence égale à la latence*

Les précédences entre les opérations sont définies à l'aide d'un graphe orienté appelé *graphe d'algorithme* $\mathcal{G}_{al} = (\mathcal{V}_{al},\mathcal{E}_{al})$ où $\mathcal{V}_{al}$ représente l'ensemble des opérations et $\mathcal{E}_{al}$ représente les contraintes de précédence. Puisqu'on traite des systèmes réactifs, le graphe d'algorithme est le résultat d'un motif répété infiniment. Dans le cas multiprocessor, les opérations peuvent avoir des durées d'exécution différentes pour chaque opérateur afin de prendre en compte des architectures hétérogènes. Une opération a besoin d'un seul opérateur pour s'exécuter et il y a au moins un opérateur sur lequel elle peut s'exécuter. Le mot "opérateur" peut désigner un processeur qui séquentialisera un ensemble d'opérations ou un circuit intégré spécifique qui lui n'exécutera qu'une seule opération spécifique. L'architecture est définie par un graphe non-orienté $\mathcal{G}_{ar} = (\mathcal{V}_{ar},\mathcal{E}_{ar})$ appelé *graphe d'architecture* où $\mathcal{V}_{ar}$ représente l'ensemble des opérateurs et $\mathcal{E}_{ar}$ représente les média permettant aux opérateurs de communiquer. Ce modèle simplifié peut en réalité être beaucoup plus détaillé et a été formalisé dans [**?**], mais comme il est hiérarchique on peut voir l'architecture de manière simplifiée (figure 3.1), c'est-à-dire sans faire apparaître les détails internes d'un opérateur, tels que les bus et leurs arbitres, les mémoires finies et leurs multiplexeurs et démultiplexeurs d'accès, et les opérateurs particuliers séquentialisant les communications sur les média.

Deux opérations qui échangent des données et qui sont distribuées sur des opérateurs différents donnent lieu à des communications entre les opérateurs. Ces communications sont modélisées par des opérations ajoutées au graphe d'algorithme entre les opérations concernées. Ensuite elles sont distribuées et ordonnancées sur les média.

Si une communication est nécessaire entre deux opérateurs qui ne sont pas directement connectés par un médium alors les communications se font à travers des opérateurs et média

intermédiaires établissant une route qui relie ces deux opérateurs. Par exemple dans la figure 3.3 on a une architecture composée de trois opérateurs et pour l'algorithme donné dans la figure 3.1 en distribuant les opérations A et C sur les opérateurs $P1$ et $P3$ on engendre une communication qui est routée par P_2 via les média $m1$ et $m2$.

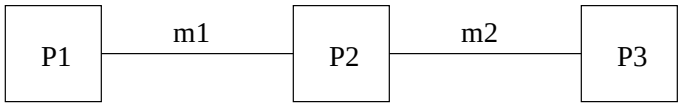

FIG. 3.3 – *Graphe d'architecture*

3.1.2 Heuristique

L'heuristique proposée [88] résout le problème d'ordonnancement et de distribution en même temps. Donc à chaque pas on choisit l'opérateur sur lequel les opérations disponibles sont distribuées et l'opération à ordonnancer. Le choix est fait en utilisant la théorie des graphes et plus particulièrement leurs chemins critiques avec les notions de « date au plus tôt » et « date au plus tard » utilisées dans l'ordonnancement classique.

Au début de l'heuristique on considère que les opérations ont une durée d'exécution égales à la moyenne de toutes leurs durées d'exécution possibles sur les différents opérateurs et la longueur du chemin critique R est calculée par rapport à ces valeurs. On note par $C_p(A)$ la durée d'exécution sur l'opérateur p et par $C(A)$ la durée moyenne d'exécution et on a : $C(A) = \frac{\sum_{p \in V_{ar}} C_p(A)}{p}$. Une fois qu'une opération est implantée (distribuée et ordonnancée) sur un opérateur p on connaît sa durée exacte d'exécution et la longueur du chemin critique est recalculée. Sa nouvelle valeur est notée par R'.

On note par $\Gamma(A)$ l'ensemble des successeurs de l'opération A et par $\Gamma^-(A)$ l'ensemble des prédécesseurs de l'opération A.

On présente les notations nécessaires à la définition de la fonction de coût qui minimise la latence égale à la cadence. Pour une implantation on définit les dates de début et de fin d'exécution de chaque opération. Ces dates sont de deux types :

- dates définies depuis le début de l'ordonnancement du motif de graphe qui font intervenir les dates de ses prédécesseurs,
- dates définies depuis la fin de l'ordonnancement du motif de graphe qui font intervenir les dates de ses successeurs.

Les dates définies depuis le début sont appelées des dates au plus tôt. On note par $S(A)$ la date de début au plus tôt de l'opération A et par $E(A)$ la date de fin au plus tôt de l'opération A. Les dates définies depuis la fin de l'ordonnancement sont appelées des dates au plus tard. On note par $S^-(A)$ la date de début au plus tard de l'opération A et par $E^-(A)$ la date de fin au plus tard de l'opération A. Les relations suivantes définissent ces dates et donnent aussi la formule de calcul de la longueur du chemin critique R :

- $S(A) = \max_{B \in \Gamma^-(A)} E(B)$ ou 0 si $\Gamma^-(A) = \emptyset$,

- $E(A) = S(A) + C(A)$,
- $E^-(A) = \max_{B \in \Gamma(A)} S^-(B)$ ou 0 si $\Gamma(A) = \emptyset$,
- $S^-(A) = E^-(A) + C(A)$,
- $R = \max_{B \in V_{al} E(B) = \max_{B \in V_{al}} S^-(B)}$.

La fonction de coût appelée « pression d'ordonnancement » est notée par $\sigma(A)$ et est définie par la relation suivante :

$$\sigma(A) = S'(A) + C_p(A) + E^-(A) - R$$

où $S'(A)$ est la valeur modifiée de $S(A)$ après avoir ordonnancée A et donc $C(A)$ devient $C_p(A)$ puiqu'on connaît l'opérateur sur lequel elle est distribuée. Elle permet de minimiser la latence ainsi que l'allongement du chemin critique dû aux communications inter-opérateurs.

L'heuristique utilise un ensemble de travail $\mathcal{W}$ qui contient à chaque pas les opérations disponibles.

Heuristique 1

Pas 1 : pour chaque opération $A \in \mathcal{W}$ on choisit le meilleur opérateur pour lequel σ_A est la plus petite pression d'ordonnancement,
Pas 2 : on choisit l'opération A_0 qui est l'opération avec la plus grande pression d'ordonnancement parmi les pressions choisies au Pas 1 pour le meilleur opérateur,
Pas 3 : l'opération A_0 est ordonnancée sur l'opérateur correspondant et on enlève l'opération de $\mathcal{W}$, toutes les opérations qui deviennent ordonnançables sont rajoutées à $\mathcal{W}$
Pas 4 : si $\mathcal{W} = \emptyset$ alors on s'arrête, sinon on va au Pas 1.

Pour illustrer le déroulement de l'heuristique on considère l'exemple suivant :

Exemple 23 *On considère le graphe d'algorithme donné dans la figure 3.4 et le graphe d'architecture donné dans la figure 3.5. On calcule d'abord les dates au plus tôt et les dates au plus tard pour chaque opération et le tableau 3.1 donne leurs valeurs numériques. Dans le même tableau, la deuxième colonne contient les durées d'exécution des opérations qu'on considère les mêmes sur les deux opérateurs. On considère la durée de communication sur le médium m égale à 6. La figure 3.6 donne l'implantation obtenue en utilisant l'heuristique 1.*

	C	$S(.)$	$E(.)$	$S^-(.)$	$E^-(.)$
A	5	0	5	25	
B	15	5	20	5	20
C	5	5	10	5	10
D	5	20	25		5

TAB. 3.1 – *Dates au plus tôt et dates au plus tard des opérations*

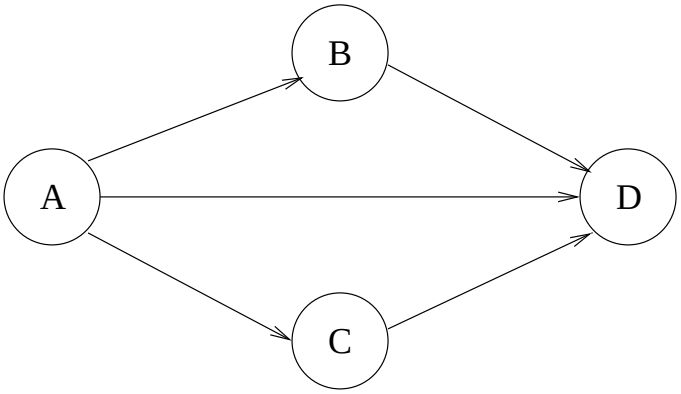

FIG. 3.4 – *Graphe d'algorithme*

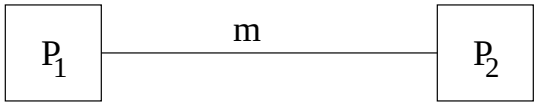

FIG. 3.5 – *Graphe d'architecture*

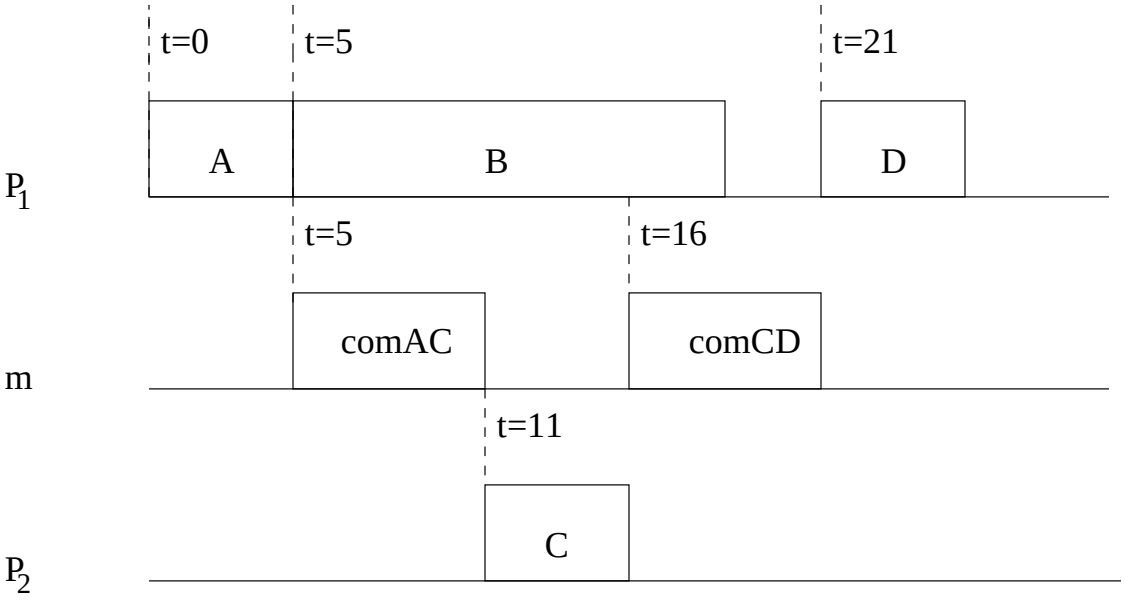

FIG. 3.6 – *Implantation obtenue en utilisant l'heuristique 1*

3.2 Contraintes de précédences et de périodicités

On commence ce chapitre par présenter le modèle utilisé afin de spécifier l'algorithme et l'architecture dans le cas des systèmes temps réel avec contraintes de précédences et de périodicités. On passe ensuite à la complexité du problème d'implantation et on finit par proposer une héuristique comparée à un algorithme exact.

3.2.1 Modèle

Le modèle d'architecture est le modèle d'architecture hétérogène AAA/SynDEx. On définit l'architecture à l'aide d'un graphe non-orienté appelé graphe d'architecture $\mathcal{G}_{al} = (\mathcal{V}_{al},\mathcal{E}_{al})$, les opérateurs communiquent à travers les média et un opérateur peut désigner un processeur ou un circuit intégré spécifique.

Le graphe d'algorithme définit les précédences entre les opérations. Ces précédences peuvent être de deux types :

- des précédences qui indiquent un échange de donnée entre les opérations. On note ce type de précédence entre deux opérations A et B par $(A,B)_d$ et dans le graphe on le marque par une flèche avec un d sur la flèche (voir figure 3.7) ;
- des précédences qui imposent seulement un ordre. On note de précédence entre deux opérations A et B par (A,B) et dans le graphe on le marque par une flèche simple. Evidemment si deux opérations A et B possèdent une simple précédence (A,B) alors si elles sont ordonnancées sur des opérateurs différents aucune communication n'est engendrée.

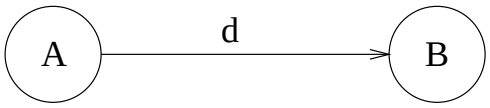

FIG. 3.7 – *Précédence avec échange de données*

A part les précédences, les opérations doivent satisfaire des contraintes de périodicité. Une fois qu'une répétition d'une opération a été ordonnancée sur un opérateur alors on choisit d'ordonnancer toutes ses répétitions sur le même opérateur. Évidemment pour que le problème ait une solution, on suppose qu'il y a au moins un opérateur sur lequel une opération quelconque peut être ordonnancée.

Pour un graphe d'algorithme et un graphe d'architecture donnés une solution pour notre problème de distribution et ordonnancement est décrite par les dates de début des opérations et l'opérateur sur lequel chaque opération a été ordonnancée. L'exemple suivant illustre une solution possible pour notre problème.

Exemple 24 *La figure 3.10 donne une des solutions possibles pour le graphe d'algorithme donné dans la figure 3.8 et pour le graphe d'architecture donné dans la figure 3.9, où toutes*

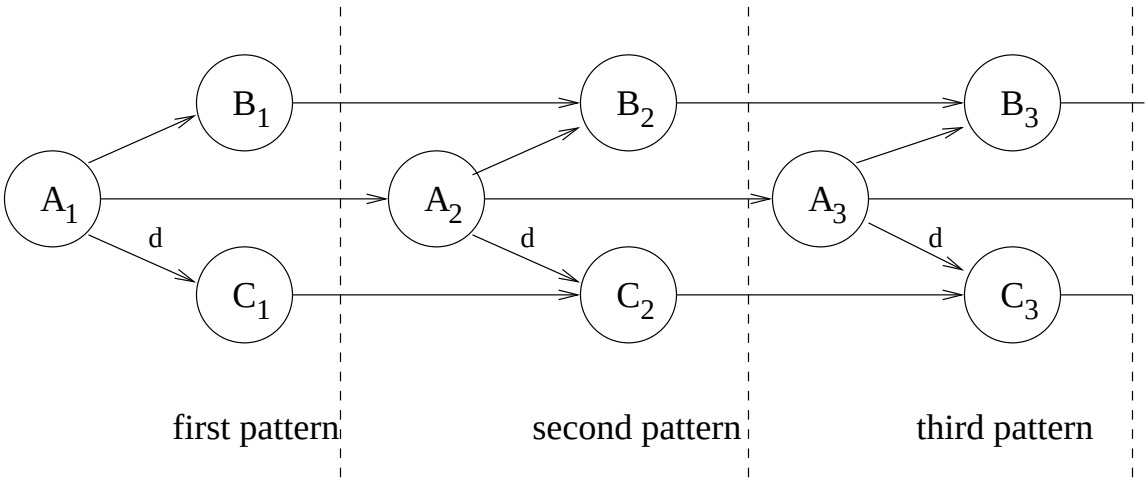

FIG. 3.8 – *Graphe d'algorithme illustrant l'exemple 24*

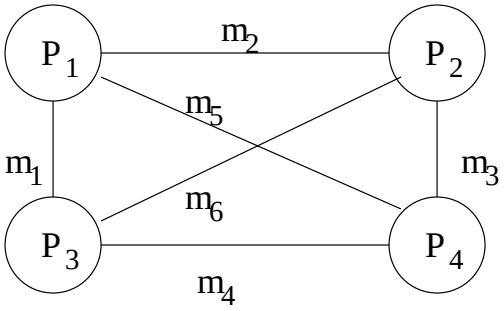

FIG. 3.9 – *Graphe d'architecture illustrant l'exemple 24*

	A	B	C
p_1	2	2	9
p_2	2	4	2
p_3	1	2	1
p_4	2	4	5

TAB. 3.2 – *Durées d'exécution pour le graphe d'algorithme donné dans la figure 3.8*

les durées d'exécution des opérations sont données dans le tableau 3.2 et les communications sont $m_1 = 2$, $m_2 = 1$, $m_3 = 1$, $m_4 = 4m_5 = 2$ *et* $m_6 = 1$. *Les périodes sont les suivantes :* $T_A = 4$, $T_B = 8$ *et* $T_C = 4$. *La solution proposée utilise seulement les opérateurs* p_1 *et* p_2.

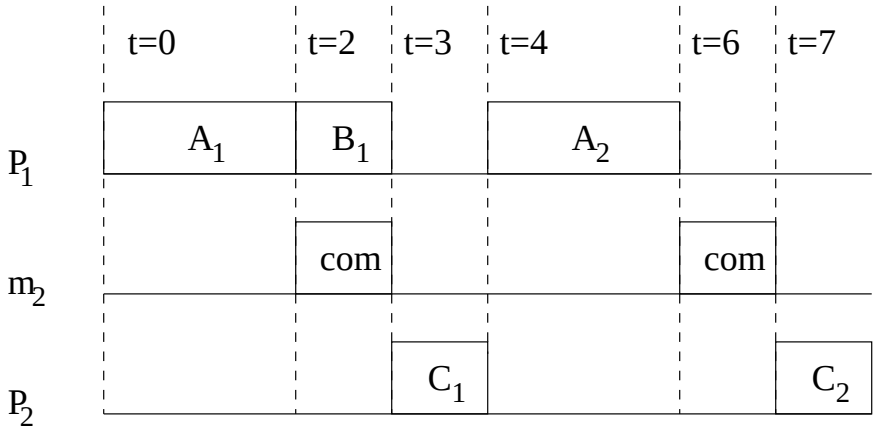

FIG. 3.10 – *Solution illustrant l'exemple 24*

Un premier résultat pour ce modèle est donné par le théorème suivant :

Théorème 30 *Pour deux opérations* A *et* B *périodiques avec au moins une précédence* $(A_i,B_j)_d$ *entre une répétition* i *de* A *et une répétition* j *de* B, *alors* $nT_A = mT_B$, *où* n *est le nombre de répétitions de* A *et* m *est le nombre de répétitions de* B.

Preuve

Puisqu'il y a au moins une précédence entre une répétition i de A et une répétition j de B, l'inégalité (3.1) est satisfaite c'est-à-dire $nT_A \leq mT_B$. On utilise un raisonnement par l'absurde pour prouver le théorème, en supposant que $nT_A < mT_B$ et qu'il y a un ordonnancement.

On note par s_A^k respectivement s_B^k la date de début de l'opération A respectivement la date de début de l'opération B appartenant au k^{me} motif. On a $s_B^k - s_A^k + C_B = s_B^1 + iT_B + s_A^1 + kT_A + C_B$. Cela signifie que

$$lim_{k \to +\infty} s_B^i - s_A^i + C_B = +\infty$$

et on a besoin d'une mémoire de taille infinie pour sauvegarder les données produites par A pour B. On ne peut pas mémoriser ces données et B ne peut pas être ordonnancée et donc la supposition faite est fausse et on obtient que $nT_A = mT_B$. Le théorème est prouvé ⊡

On passe à la preuve de complexité de notre problème. La preuve de complexité indique qu'il ne peut y avoir un algorithme polynômial qui propose une implantation satisfaisant toutes les contraintes.

3.2.2 Complexité du problème d'implantation

Pour prouver la complexité de notre problème d'implantation avec contraintes de précédences et de périodicités noté par PPP, on se rapporte de nouveau au problème étudié par Korst dans [23]. On rappelle que c'est le problème d'implantation avec valeurs quelconques de périodes et de durées d'exécution pour des systèmes d'opérations avec contraintes de périodicité. Le problème de décision associé au problème d'ordonnancement est noté par PSP et il formulé comme étant le suivant : soit $\mathcal{V}$ un ensemble de n opérations périodiques $A_1, \ldots, A_n$ avec T_i la période et $C_i \leq T_i$ la durée d'exécution de l'opération A_i pour tout $1 \leq i \leq n$, un ensemble d'opérateurs identiques et un entier k. Y a-t-il un ordonnancement des opérations sur $m \leq k$ opérateurs qui satisfait les contraintes de périodicité des opérations appartenant à $\mathcal{V}$?

On rappelle le théorème prouvant le fait que le problème de décision associé au problème d'implantation est NP-complet au sens fort.

Théorème 31 *[23] PSP est NP-complet au sens fort pour $k = n$.*

Théorème 32 *PPP est NP-difficile au sens fort.*

Preuve Le problème d'implantation de Korst ne prend pas en compte l'échange de données entre les opérations et donc la preuve d'équivalence du théorème 9 avec le problème de Korst prouve que le sous-cas sans donnée de notre problème est NP-difficile au sens fort. En conséquence notre problème d'implantation qui prend en compte l'échange de données est aussi NP-difficile au sens fort. Le théorème est prouvé ⊡

Ce résultat implique qu'on ne peut pas toujours trouver une implantation optimale et dans ce cas on propose l'heuristique suivante.

3.2.3 Heuristique et algorithme exact

L'heuristique proposée s'appuie sur les résultats d'ordonnançabilité monoprocesseur en ce qui concerne l'ordonnancement et sur l'heuristique AAA/SynDEx en ce qui concerne la distribution. Ces résultats sont les suivants :

- résultats monoprocesseur utilisées pendant l'heuristique et relations à satisfaire. Pour un système d'opérations avec précédences et périodicités les relations suivantes doivent

être satisfaites :

- s'il y a une contrainte de précédence entre au moins une répétition de i de A et une répétition de j de B alors on a :

$$nT_A \leq mT_B \tag{3.1}$$

  où n est le nombre de répétitions de A et m est le nombre de répétitions de B ;
- s'il y en plus un échange de données l'inégalité (3.1) devient égalité donc on a $nT_A = nT_B$;

– résultats multiprocesseur AAA/SynDEx

L'heuristique est basée sur le fait que deux opérations périodiques avec le périodes T_i et T_j qui ne forment pas une séquence telle que $T_i/T_j, \forall i < j$ ne peuvent pas être ordonnancées sur le même opérateur. et l'exemple suivant illustre cette situation.

FIG. 3.11 – *Graphe d'algorithme*

Exemple 25 *On considère le graphe d'algorithme donné dans la figure 3.11 avec $T_A = 2$ et $T_B = 3$ et toutes les durées d'exécution égales à 1. Le système n'est pas ordonnaçable dans le cas d'un seul opérateur. La figure 3.12 montre que la troisième répétition de l'opération A et la dexième répétition de l'opération B doivent avoir la même date de début égale à 4 et cela n'est pas possible.*

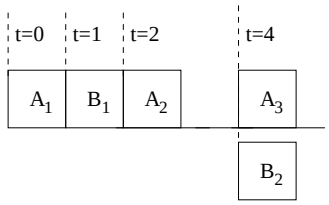

FIG. 3.12 – *Ordonnancement monoprocesseur*

Afin d'ordonnancer des opérations avec périodes qui ne forment pas des séquences des multiples sur des opérateurs différents, on définit des classes d'opérations où chaque classe $\mathcal{C}_T$ contient seulement des opérations avec les périodes multiples entre elles avec T la plus petite période des opérations appartenant à la classe. Ces classes ne s'excluent pas mutuellement. Par exemple, une opération avec une période $T = 10$ peut appartenir à la classe $\mathcal{C}_2$ et à la classe $\mathcal{C}_5$.

Une fois une opération ordonnacée sur un opérateur, seulement sa classe peut utiliser cet opérateur. En conséquence, le principe de l'heuristique proposée est le suivant :

- tant que possible choisir parmi les opérateurs p, sur lesquels ses prédécesseurs (qui ont la même période ou la même multiplicité de la période mise dans une classe) ont été ordonnancés, le meilleur opérateur p_m pour lequel l'opération A a la plus petite date de fin :
  $F(A,p_m) = max_{B \in Prec(A)}\{F(B,p)\} + C_{Ap_m}$, où $F(B,p)$ est la fin de l'exécution de l'opération B sur le opérateur p et C_{Ap_m} la durée d'exécution de l'opération A sur l'opérateur p_m ;
- si ce n'est plus possible choisir un opérateur quelconque p_q parmi les opérateurs p avec des opérations ordonnancées qui la même période ou la même multiplicité de la période et pour lequel on obtient la plus petite fin de l'exécution :
  $F'(A,p_q) = max_{B \in Prec(A)}\{F(B,p) + com(p,p_q)\} + C_{Ap_q}$, où $com(p,p_q)$ est la communication entre p_q et p.

L'heuristique utilise l'ensemble de travail $\mathcal{W}$ qui contient les opérations disponibles c'est-à-dire les opérations qui ont tous les prédécesseurs déjà ordonnancée. On note par s_{lp} et C_{lp} la date de début et la durée d'exécution de la dernière opération ordonnancée sur l'opérateur p. On initialise au début de l'heuristique toutes ces valeurs à 0. Afin de mettre à jour l'ensemble $\mathcal{W}$ on enlève l'opération ordonnancée de $\mathcal{W}$, toutes les opérations qui deviennent ordonnançables sont rajoutées à $\mathcal{W}$. On note $\mathcal{C}_T$ la classe des opérations qui ont la même période ou la même multiplicité de la période où T est la plus petite période de la classe. On note $\mathcal{P}er$ l'ensemble des opérations périodiques.

Heuristique 2

Pas 1 : on choisit parmi les opérations pour chaque classe avec la plus petite période de $\mathcal{W}$ celle qui a la plus petite durée d'exécution (en choisissant parmi toutes les durées sur tous les opérateurs). On ordonnance ces opérations. On met à jour $\mathcal{W}$, s_{lp} et C_{lp} ;

Pas 2 : pour chaque opérateur p, on cherche une opération $A \in \mathcal{W} \bigcap \mathcal{P}er$ avec A_1 déjà ordonnancée telle que :

$$s_{A_i} + T_A - s_{lp} - C_{lp} = min_{B_i \in \mathcal{W} \bigcap \mathcal{P}er \ \ et \ B_1 \ \ déjà \ ordonnancée} \{s_{B_i} + T_B - s_{lp} - C_{lp}\}$$

où A_i est la dernière répétition ordonnancée de A. Si on trouve plusieurs opérations A pour le même opérateur alors le système n'est pas ordonnançable avec l'heuristique. On choisit l'opérateur avec la date de fin $s_{A_i} + T_A - s_{lp} - C_{lp}$ la plus petite et on va au Pas 3 ;

Pas 3 : si $\exists C \in \mathcal{W} \bigcap C_{T_A}$ avec C_1 pas ordonnancée, C et A appartenant à la même classe et $s_{lp} + C_{lp} + C_C \leq s_{A_i} + T_A$, pour l'opération A et l'opérateur p trouvés précédemment, alors on a $s_C = s_{lp} + C_{lp}$ et on va au Pas 5. Sinon, on va au Pas 4

Pas 4 : on a $s_A = s_{lp} + C_{lp}$ t.q. $s_{A_{i+1}} = s_{A_i} + T_A$ et on va au Pas 5 ;

Pas 5 : s'il y a des opérations dans $\mathcal{W}$ appartenant à une classe dont aucune opération n'a encore été ordonnancée alors on choisit l'opération avec la plus petite période appartenant à la classe de celle qui a la plus petite durée d'exécution (en choisissant parmi toutes les durées sur tous les opérateurs) et on va au Pas 6 ;

Pas 6 : s'il y a des opérations appartenant à une classe dont les opérateurs utilisés ne permettent plus l'ordonnancement des opérations sans la première répétition ordonnancée, alors on choisit un nouvel opérateur pour cette classe et les opérateurs déjà utilisés par cette classe ne sont plus utilisables. L'opération de la classe qui a la plus petite période et la plus petite durée d'exécution est ordonnancée avec la date de début égale à la plus grande date de fin de ses prédécesseurs. On va au Pas 2.

Remarque 20 *L'heuristique ordonnance le plus possible d'opérations, éventuellement reliées par des échanges de données, qui ont la même période sur le même opérateur. Cela implique une baisse des communications puiqu'elles sont engendrées par les échanges de données.*

Voici un exemple d'utilisation de l'heuristique pour un graphe d'algorithme avec 9 opérations donné dans la figure 3.13 et un graphe d'architecture avec 3 opérateurs donné dans la figure 3.14. Dans cet exemple l'heuristique produit un ordonnancement et une distribution qui respectent les contraintes à satisfaire.

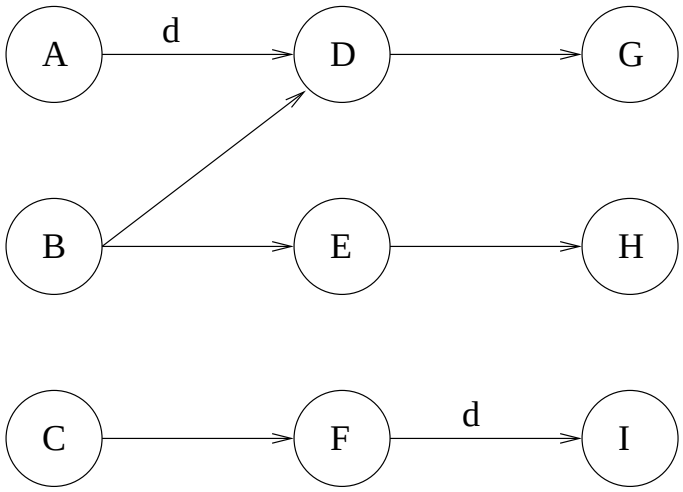

FIG. 3.13 – *Graphe d'algorithme illustrant l'heuristique 2*

Exemple 26 *On considère le graphe d'algorithme donné dans la figure 3.13 et le graphe d'architecture donné dans la figure 3.14. On définit les périodes* $T_A = T_B = T_D = T_E = 4$, $T_G = T_H = 8$ *et* $T_C = T_F = T_I = 3$. *On a ici les deux classes d'opérations* $\mathcal{C}_4$ *et* $\mathcal{C}_3$. *Les durées*

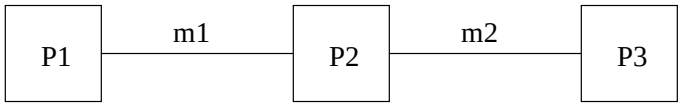

FIG. 3.14 – *Graphe d'architecture illustrant l'heuristique 2*

d'exécution sont toutes égales à 1 *et les durées de communication sur les média sont égales à* 1. *La figure 3.15 donne l'implantation obtenue en utilisant l'heuristique 2. Etant donné qu'aucune communication est faite via les média, on ne les réprésente plus sur la figure.*

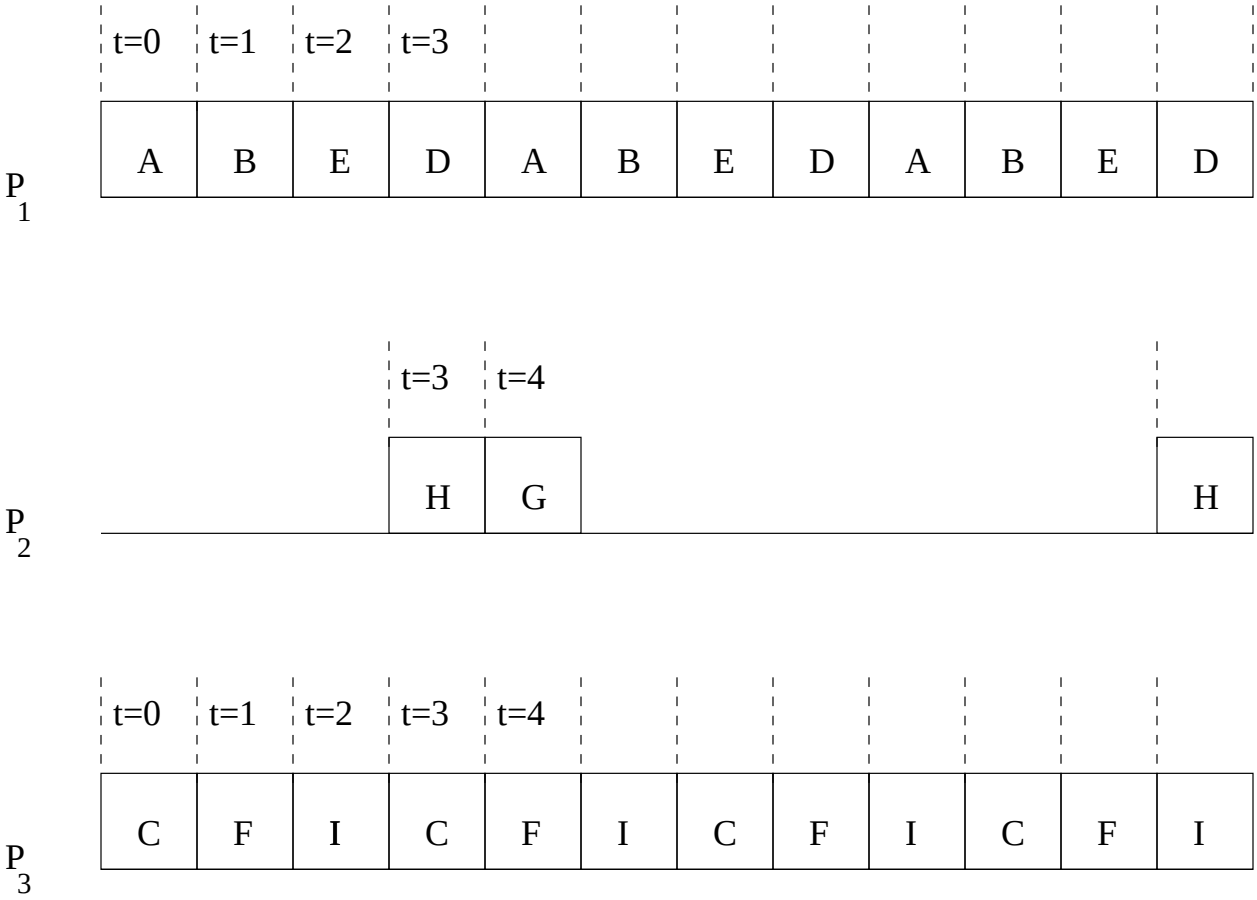

FIG. 3.15 – *Implantation obtenue en utilisant l'heuristique 2*

On présente maitenant l'algorithme exact de type branch & bound [89]. On suppose que l'on a m opérateurs mis dans une liste dans un ordre quelconque. Sans perdre de généralité, on suppose que l'on a n opérations mises dans une liste selon un ordre total tel que pour une opération quelconque i tous ses prédécesseurs ont un indice plus petit qu'elle et toutes

les opérations, qui deviennent disponibles en même temps qu'elle et qui ont une période plus grande qu'elle, ont un indice plus grand. L'exemple suivant suit ce type d'ordre.

Exemple 27 *On considère le graphe d'algorithme donné dans la figure 3.13 avec les périodes* $T_A = T_B = T_D = T_E = 4$, $T_G = T_H = 8$ *et* $T_C = T_F = T_I = 3$. *On obtient alors la liste ordonnée* $\{C,F,I,A,B,D,E,G,H\}$.

Algorithme 8

Initialisation : on initialise l'indice des opérations $i = 1$ et l'indice des opérateurs $j = 1$, la date de début de la dernière opération ordonnancée sur chaque opérateur $s(j) = 0$ et la durée de la dernière opération ordonnancée sur chaque opérateur $C(j) = 0$;

Pas 2 : Si $i \leq n$ alors on ordonnance toutes les répétitions de l'opération i sur l'opérateur j, $s(j) = s(j) + C(j)$ et $C(j) = C_i$, sinon on s'arrête. S'il y a au moins une contrainte qui n'est pas satisfaite alors i prend la valeur $i + 1$ et on va au Pas 2. Sinon on va au Pas 3 ;

Pas 3 : Si $j > n$ alors i prend $i - 1$ et j prend la valeur $j_i - 1$ où j_i est l'indice de l'opérateur sur lequel l'opération i a été ordonnancée et on va au Pas 2. Sinon on va au Pas 4 ;

Pas 4 : $j = j + 1$ et on va au Pas 2.

L'algorithme ne fait que parcourir toutes les solutions possibles en utilisant une technique branch & bound et il s'arrête dès qu'une solution est trouvée. Dans le cas où il n'y aucune solution possible satisfaisant les contraintes, l'algorithme s'arrête après avoir parcouru toutes ces solutions. En sachant que le problème à résoudre est un « problème de nombre » [34] dont la taille d'une instance est importante, l'heuristique est plus intéressante que l'algorithme 8 pour des graphes contenant beaucoup d'opérations.

Exemple 28 *On considère le graphe d'algorithme donné dans la figure 3.13 et le graphe d'architecture donné dans la figure 3.14. On définit les périodes* $T_A = T_B = T_D = T_E = 4$, $T_G = T_H = 8$ *et* $T_C = T_F = T_I = 3$. *En utilisant l'algorithme 8, on obtient l'implantation donnée dans la figure 3.16.*

Théorème 33 *L'algorithme 8 a la complexité* $O(n\,m)$ *où* n *est le nombre d'opérations et* m *est le nombre d'opérateurs.*

Preuve L'algorithme parcourt n fois l'ensemble d'opérations et pour chaque opération on teste au maximum m opérateurs. On obtient une complexité $\mathcal{O}(n\,m)$. Le théorème est prouvé ⊡

Remarque 21 *Un algorithme exact est, en général, un algorithme qui parcourt l'arbre de recherche des solutions possibles. La complexité* $\mathcal{O}(n\,m)$ *de notre algorithme exact vient du fait qu'on parcourt seulement une partie de cet arbre de recherche et cela est dû à l'ordre quelconque de la liste des opérateurs. Ce type d'algorithme est aussi connu sur le nom de « méthode arborescente » (voir [34]).*

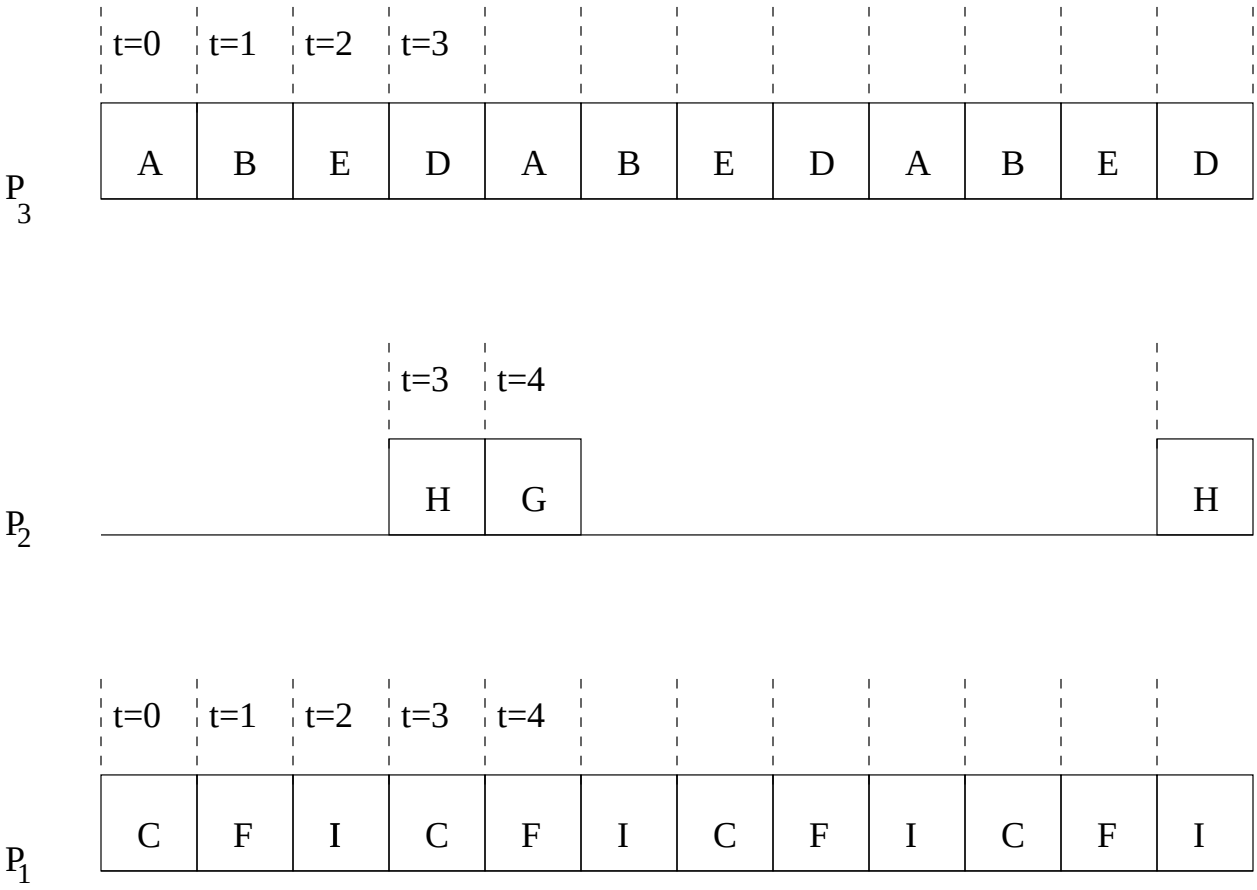

FIG. 3.16 – *Implantation obtenue en utilisant l'algorithme 8*

3.2.4 Etude de performances

On compare l'heuristique 2 à l'algorithme exact 8 afin d'étudier ses performances.

On suppose avoir appliqué les propriétés d'héritage des périodes (voir Corollaire 3) et donc n'avoir que des opérations périodiques. De plus, l'algorithme exact et l'heuristique partent d'une liste ordonnée d'opérations. Cette liste est obtenue en utilisant une fonction de rang [90] qui donne une liste ordonnée où toutes les opérations ont leurs prédécesseurs ordonnés avant eux dans la liste. Ainsi l'algorithme exact et l'heuristique ont à vérifier seulement la contrainte de périodicité. De plus, l'heuristique classe les opérations par rapport à leurs périodes ou des multiples de leurs périodes.

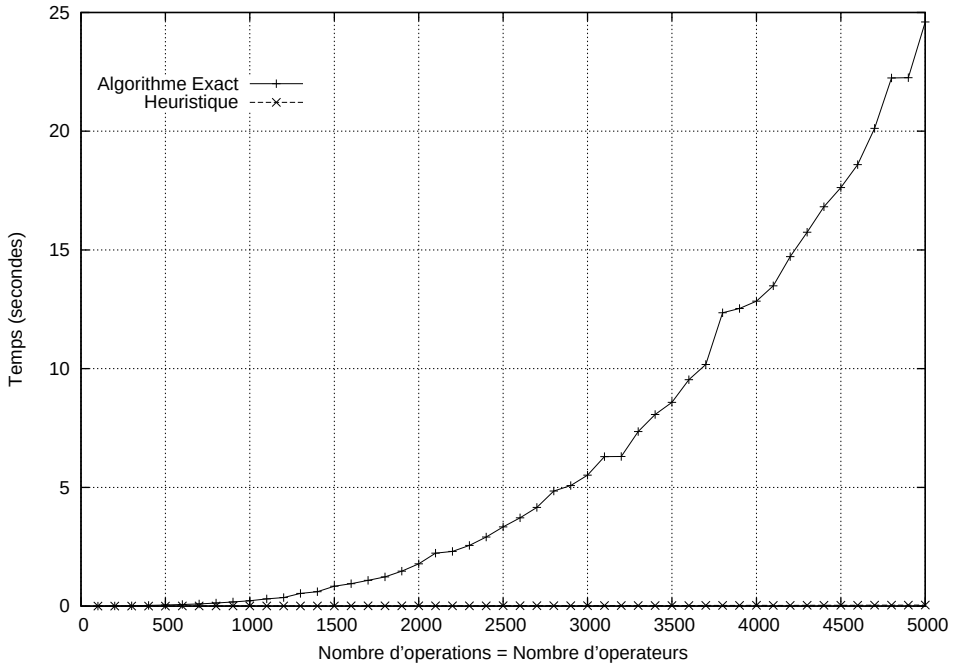

FIG. 3.17 – *Etude de performances comparant l'algorithme exact et heuristique*

Les listes d'opérations ont été générées de manière aléatoire et le temps nécessaire pour leur génération n'est pas pris en compte lors du calcul de temps d'exécution ni pour l'algorithme exact, ni pour l'heuristique. On a choisi un nombre d'opérateurs égal au nombre d'opérations afin d'augmenter le nombre de problèmes générés aléatoirement qui ont une solution. Cela permet aussi de compter le nombre d'opérateurs réellement utilisés (sur lequel il y a au moins une opération distribuée) par l'algorithme exact et l'heuristique, cela est montré et dans le tableau 3.3.

On compare le temps d'exécution pour l'algorithme et l'heuristique pour le même exemple. Cette comparaison est donnée par la figure 3.17 qui présente les résultats des simulations numériques. On observe que pour un petit nombre d'opérations, les deux mettent des temps comparables pour trouver une solution et plus on augmente le nombre d'opérations, plus

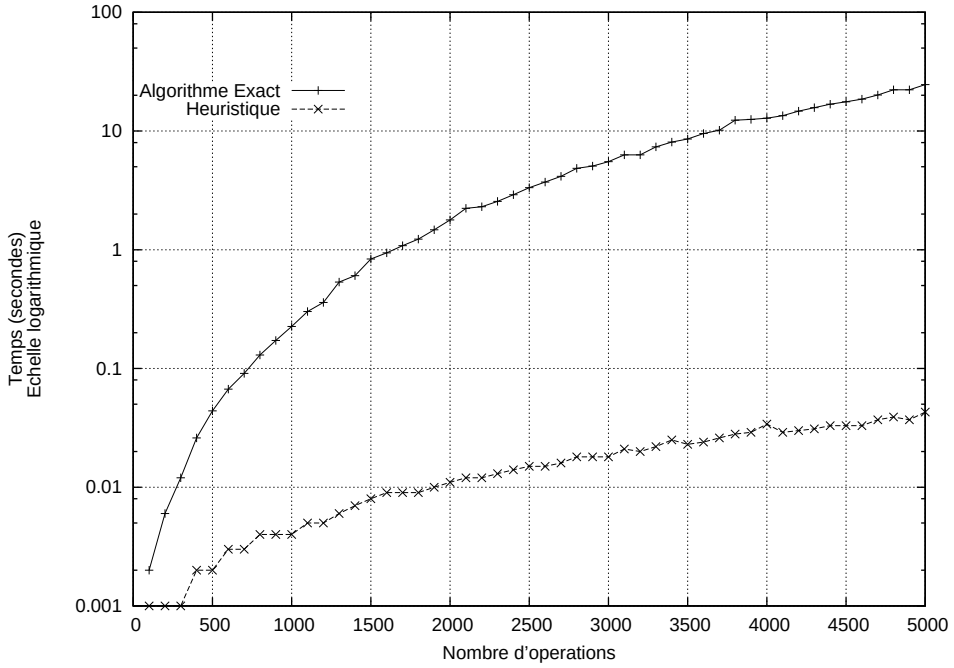

FIG. 3.18 – *Etude de performances échelle logarithmique comparant l'algorithme exact et heuristique*

l'heuristique est intéressante. Afin de mieux illustrer les différences, on donne aussi les résultats des simulations suivant une échelle logarithmique (voir figure 3.18).

En conclusion, l'heuristique est plus intéressante du point de vue du temps d'exécution, mais aussi de l'équilibrage de charge des opérateurs puisque l'algorithme exact utilise au maximum certains opérateurs (taux d'utilisation à 100 %) et d'autres ne sont jamais utilisés.

3.3 Contraintes de précédences et de latences

On commence dans ce chapitre par présenter le modèle utilisé afin de spécifier l'algorithme et l'architecture dans le cas des systèmes temps réel avec contraintes de précédences et de latences. On passe ensuite à la complexité du problème d'implantation et on finit par proposer une heuristique comparée à un algorithme exact.

3.3.1 Modèle

On utilise le même modèle que celui présenté dans le chapitre 3.2.1 pour le graphe d'architecture et le graphe d'algorithme sauf que les contraintes imposées aux opérations sont des contraintes de latence.

L'exemple 29 propose un ordonnancement et une distribution pour un graphe d'architecture et un graphe d'algorithme avec des contraintes de latence. On remarque que les

Nombre opérations	Opérateurs - algorithme exact	Opérateurs - heuristique
100	62	64
1000	628	703
2000	1256	1405
3000	1868	2112
4000	2455	2786
5000	3071	3490
6000	3702	4214
7000	4272	4894
8000	4857	5549
9000	5528	6291
10000	6076	6958

TAB. 3.3 – *Comparaison entre le nombres d'opérateurs utilisés par l'algorithme exact et par l'heuristique*

opérations appartenant aux trois contraintes de latence ont des durées minimales sur des opérateurs différents et cela facilite le choix de la distribution dans le sens où il suffit d'ordonnancer chaque opération sur l'opérateur pour lequel elle a une durée minimale.

Exemple 29 *On considère le graphe d'algorithme donné dans la figure 3.19 et le graphe d'architecture donné dans la figure 3.20. On considère les durées d'exécution données dans le tableau 3.4 et on définit les contraintes de latence suivantes :* $L_{AC} = 8$ *et* $L_{DF} = 7$*. La durée de communication sur le médium* m *est égale à* 1*. La figure 3.21 donne une implantation possible.*

	P_1	P_2
A	1	5
B	2	7
C	4	5
D	10	2
E	5	2
F	5	1

TAB. 3.4 – *Durées d'exécution des opérations du graphe donné dans la figure 3.19*

3.3.2 Complexité du problème d'implantation

Afin de prouver la complexité de notre problème d'implantation, on définit le problème de décision associé et on compare celui-ci à un problème de décision déjà prouvé NP-complet. De cette manière on peut conclure que notre problème d'implantation est NP-difficile.

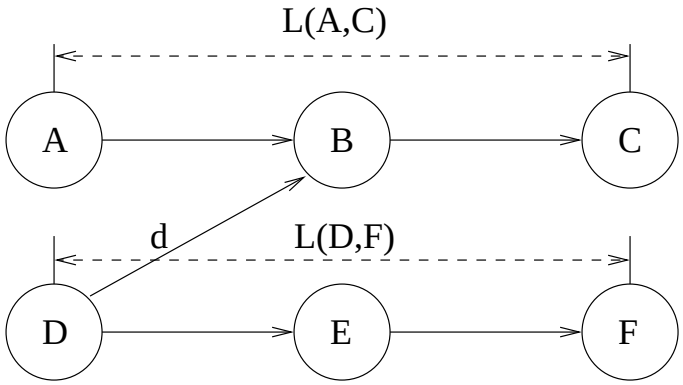

FIG. 3.19 – *Graphe d'algorithme*

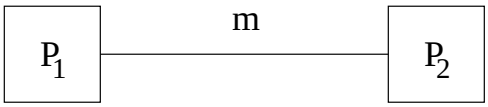

FIG. 3.20 – *Graphe d'architecture*

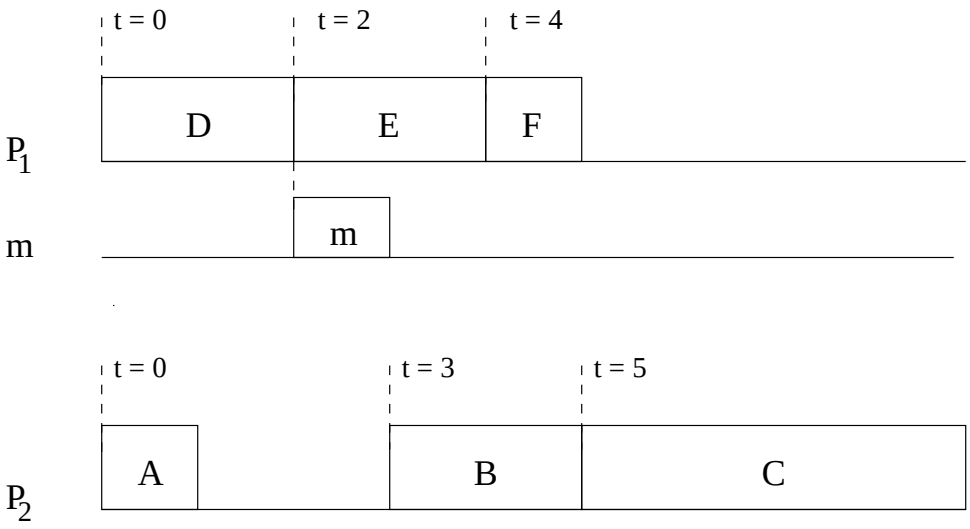

FIG. 3.21 – *Solution possible pour un système illustrant l'exemple 29*

On note par PPL le problème d'implantation avec contraintes de latences et de précédences définies sur un graphe d'algorithme, dans le cas multiprocesseur où les opérateurs sont définis à l'aide d'un graphe d'architecture. Il est facile de vérifier que $PPL \in$ NP étant donné qu'on peut déduire en temps polynomial si un ensemble de paires (s_A,p), deviné par un algorithme non-détérministe, est une solution pour PPL.

Théorème 34 *Le problème PPL est NP-difficile au sens fort.*

Preuve Le théorème 22 prouve que la contrainte de périodicité est un cas particulier de contrainte de latence. En conséquence le problème PPP qui est prouvé NP-difficile au sens fort dans le théorème 32 est un sous-cas de problème PPL. On obtient que le problème PPL est aussi un problème NP-difficile au sens fort. Le théorème est prouvé ⊡

3.3.3 Heuristique et algorithme exact

Les contraintes de latence sont celles qui aident à décider l'opérateur à utiliser pour une paire d'opérations sur laquelle on a défini une contrainte de latence. Le fait d'ordonnancer des opérations appartenant à une contrainte de latence sur des opérateurs différents allonge le temps écoulé entre la première opération et la dernière opération d'une contrainte de latence avec le temps de communication de la donnée entre les opérateurs.

L'heuristique utilise l'ensemble de travail $\mathcal{W}$ qui contient les opérations disponibles et on note par s_{lp} et C_{lp} la date de début et la durée d'exécution de la dernière opération ordonnancée sur l'opérateur p. On initialise au début de l'heuristique toutes ces valeurs à 0.

On note par $\mathcal{L}_X$ (voir chapitre 2.2) l'ensemble des latences qui sont en relation X avec d'autres contraintes de latence et qui n'ont pas la différence entre la latence et la somme des durées d'exécution des opérations appartenant à la paire égale à 0. On considère cet ensemble ordonné, si $L(A,B)$ est avant $L(C,D)$ dans l'ensemble $\mathcal{L}_X$ alors $L_{AB} < L_{CD}$.

On note $\mathcal{C}_{\mathcal{L}_X}$ la classe des opérations qui appartiennent à des contraintes de latence L en relation X. Pour chaque classe on détermine le meilleur opérateur, c'est-à-dire celui pour lequel la somme des durées d'exécutions d'opérations appartenant à une contrainte de latence est la plus petite.

L'heuristique utilise aussi l'algorithme de marquage donné au chapitre 2 que l'on rappelle ci-dessous :

Algorithm 3

Initialisation : si (A,B) a une contrainte de latence $L(A,B)$ alors $marque(B) = L(A,B)$ et $marque(A) = \infty$, sinon $marque(A) = marque(B) = \infty$. De plus, si B appartient à plusieurs paires avec des contraintes de latence alors $marque(B) = min_{(C,B)\in\mathcal{L}}\{L(C,B)\}$ et $marque(A) = \infty$. Soit $\mathcal{W} = \mathcal{L}$.

Pas 1 : pour $(A,B) \in \mathcal{W}$ et pour chaque opération $C \in V\backslash\{A,B\}$, il y a trois possibilités :
(a) si $A \in \mathcal{D}(C)$ alors $marque(C) = marque(C)$;

(b) si $A \notin \mathcal{D}(C)$ et $B \in \mathcal{D}(C)$, alors $marque(C) = min(marque(C),marque(B))$;
(c) si $A \notin \mathcal{D}(C)$ et $B \notin \mathcal{D}(C)$ alors $marque(C) = marque(C)$.
Soit $\mathcal{W} = \mathcal{W} \backslash \{(A,B)\}$.

Pas 2 : si $\mathcal{W} \neq \emptyset$ alors on va au Pas 1, sinon l'algorithme s'arrête.

L'heuristique s'inspire de l'algorithme donné pour le problème dans le cas monoprocesseur dans le chapitre 2.

Heuristique 3

Initialisation: $\mathcal{W} = \bigcup_{A \in V \ \ et \ Prec(A)=\emptyset} \{A\}$ et $s_{lp} = 0, C_{lp} = 0$.

Pas 1 *(opération avec contrainte de latence)* :
si $\exists A \in \mathcal{W}$ tel que $marque(A) \neq \infty$ alors on choisit A tel que $marque(A) = min_{B \in \mathcal{W}}\{marque(B)\}$. Si $\exists L(C_1,D_1) \in \mathcal{L}_X$ et $\exists L(C_i,D_i) \in \mathcal{L}_X, \forall i = 2,3,\ldots n$ tel que $A \in \bigcup_{E \in \Gamma^+_{C_i D_i}(C_1,D_1)} \mathcal{M}(E,D_1), \forall i = 2,3,\ldots n$ alors soit $B \in \mathcal{W}$ l'opération avec la plus petite marque tel que $\exists i \in \{2,3,\ldots,n\}$ avec $B \in \bigcup_{E \in \Gamma^+_{C_1 D_1}(C_i,D_i)} \mathcal{M}(E,D_i)$. Si l'opération B existe alors $s_B = s_{lp} + C_{lp}$, on enlève B de $\mathcal{W}$, on choisit l'opérateur déjà utilisé par sa classe ou le meilleur si aucune opération de la classe n'a pas été encore ordonncées, toutes les opérations qui deviennent ordonnançables sont ajoutées à $\mathcal{W}$ et on va au Pas 4. Si l'opération B n'existe pas alors $s_A = s_{lp} + C_{lp}$, on enlève A de $\mathcal{W}$, on choisit l'opérateur déjà utilisé par sa classe ou le meilleur si aucune opération de la classe n'a pas été encore ordonnancées, toutes les opérations qui deviennent ordonnançables sont ajoutées à $\mathcal{W}$ et on va au Pas 4.

Pas 2 *(opération qui n'est pas première dans une contrainte de latence)*:
si $\exists A \in \mathcal{W}$ tel qu'il n'y a pas d'opération $B \in V$ avec $(A,B) \in \mathcal{L}$ alors $s_A = s_l + C_l$, on l'enlève de $\mathcal{W}$, toutes les opérations qui deviennent ordonnançables sont rajoutées à $\mathcal{W}$ et on va au Pas 4.

Pas 3 *(opération qui est première dans une contrainte de latence)* :
si $\exists A \in \mathcal{W}$ tel qu'il y a $B \in V$ avec $(A,B) \in \mathcal{L}$ alors $s_A = s_{lp} + C_{lp}$ avec $L(A,B) = max_{(C,D) \in \mathcal{L} \ \ et \ C \in \mathcal{W}}\{L(C,D)\}$, on l'enlève de $\mathcal{W}$, on choisit celui avec la somme minimale, toutes les opérations qui deviennent ordonnançables sont rajoutées à $\mathcal{W}$ et on va au Pas 4.

Pas 4: si $\mathcal{W} = \emptyset$ alors l'algorithme s'arrête, sinon on va au Pas 1.

On donne un exemple d'utilisation de l'heuristique pour un graphe d'algorithme et un graphe d'architecture. Dans cet exemple on obtient une implantation qui satisfait les contraintes de latence.

Exemple 30 *On considère le graphe d'algorithme donné dans la figure 3.22 et le graphe d'architecture donné dans la figure 3.23. On considère les durées d'exécution données dans*

le tableau 3.5 et on définit les contraintes de latence suivantes: $L_{AC} = 8$, $L_{DF} = 18$ *et* $L_{GI} = 7$. *Les durées de communication sur les média sont égales à* 1. *On détermine pour chaque contrainte de latence le meilleur opérateur: pour* L_{AC} *on obtient l'opérateur* P_1 *avec la somme égale à* 7, *pour* L_{DF} *l'opérateur* P_2 *avec* 5 *et pour* L_{GI} *l'opérateur* P_2 *avec* 4. *La figure 3.24 donne une implantation obtenue en utilisant l'heuristique 3.*

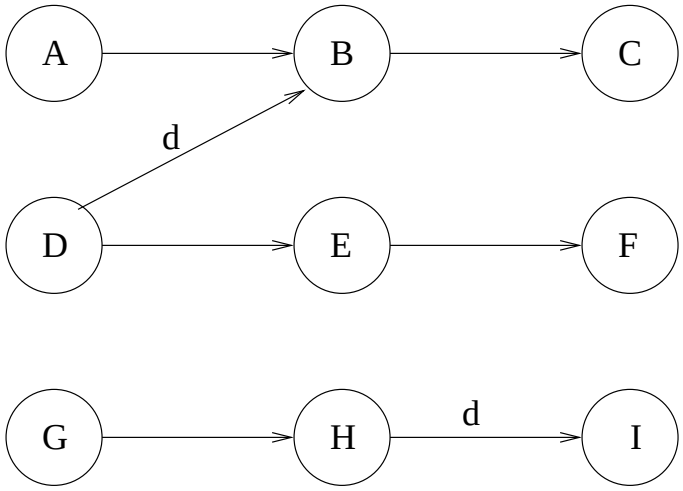

FIG. 3.22 – *Graphe d'algorithme*

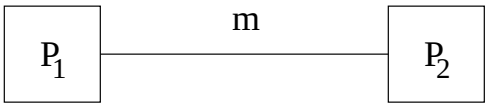

FIG. 3.23 – *Graphe d'architecture*

On suppose que les opérations de notre graphe d'algorithme sont numérotées de 1 à n où n est le nombre d'opérations. De même on suppose que les opérateurs de notre graphe d'architecture sont numérotés de 1 à m où m est le nombre d'opérateurs.

L'algorithme exact parcourt tous les nombres en base n où n est le nombre d'opérations. Pour chaque nombre en base n, la valeur $j < n$ occupant la $i^{\text{ème}}$ position dans le nombre donne la place où se trouve la $i^{\text{ème}}$ opération sur l'opérateur sur lequel elle a été ordonnancée et distribuée, position qu'on appelle «rang». Afin d 'ordonnancer une opération i du rang j, c'est-à-dire de fixer sa date de début, on choisit parmi toutes les opérations avec le rang $j-1$ celle qui à la fin la plus tôt calculée à partir de la fin du dernier prédécesseur ordonnancé de l'opération i. L'opérateur sur lequel est ordonnancée l'opération choisie parmi les opérations de rang $j-1$ est l'opérateur sur lequel l'opération i est distribuée. Ainsi l'opération i est ordonnancée et distribuée. Les opérations ayant le rang 0 ont la date de début 0. En avançant dans les rangs l'algorithme construit un ensemble ordonné de paires tel que la $i^{\text{ème}}$ paire

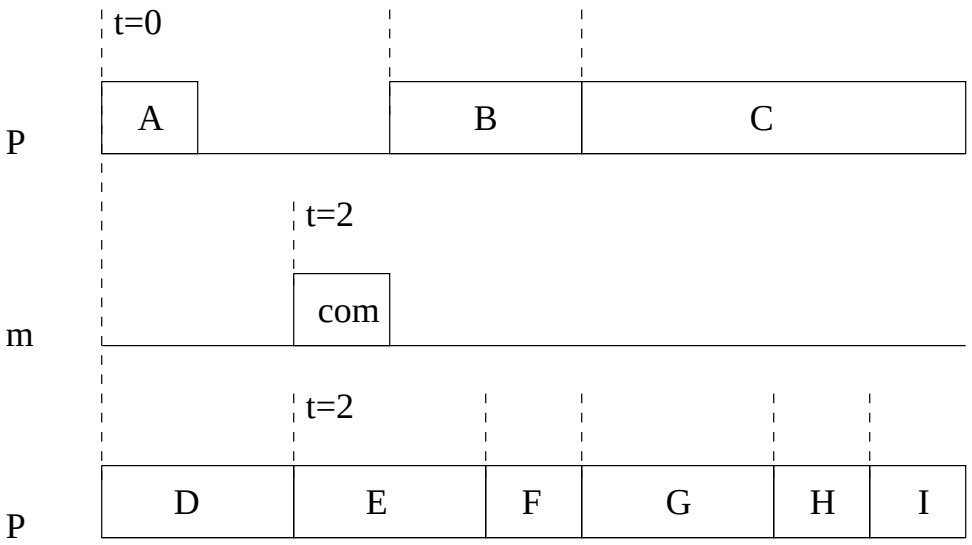

FIG. 3.24 – *Implantation obtenue en utilisant l'heuristique 3*

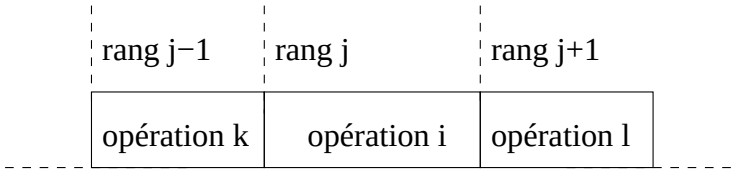

FIG. 3.25 – *Ordonnancement illustrant la définition du rang*

	P_1	P_2
A	1	5
B	2	7
C	4	5
D	10	2
E	5	2
F	5	1
G	1	2
H	2	1
I	4	1

TAB. 3.5 – *Durées d'exécution des opérations du graphe donné dans la figure 3.22*

corresponde toujours à la $i^{\text{ème}}$ opération à distribuer et ordonnancer. On appelle un ensemble ayant cette forme une solution potentielle de notre problème. La $i^{\text{ème}}$ paire est formée de la date de début de l'opération i et de l'opérateur sur lequel l'opération i est distribuée et ordonnancée. Cet ensemble contient donc n paires. Si les dates de début correspondant à une solution potentielle satisfont toutes les contraintes alors cette solution potentielle est une solution du problème. En parcourant tous les nombres en base n selon ce principe, l'algorithme exact parcourt toutes les combinaisons d'ordonnancement possibles pour chaque opérateur, donc toutes les solutions potentielles.

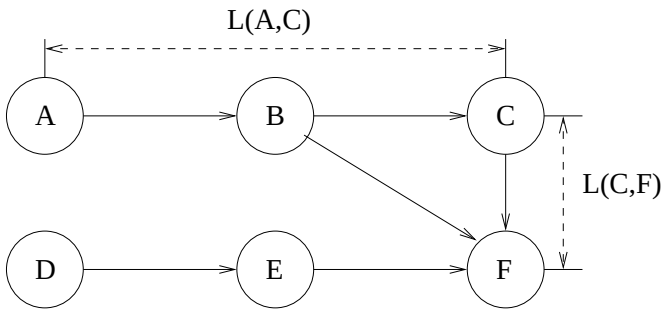

FIG. 3.26 – *Graphe d'algorithme illustrant l'exemple 31*

Exemple 31 *Cet exemple illustre la construction d'un ordonnancement à partir d'une solution potentielle par l'algorithme exact 9. Soit* $\mathcal{G}_{al} = (\mathcal{V}_{al},\mathcal{E}_{al})$ *le graphe d'algorithme donné dans la figure 3.26 et* $\mathcal{G}_{ar} = (\mathcal{V}_{ar},\mathcal{E}_{ar})$ *le graphe d'architecture donné dans la figure 3.23. Toutes les opérations ont la durée égale à* 1.

Le graphe d'algorithme a six opérations A,B,C,D,E,F *donc* 001122 *est une solution potentielle. L'algorithme construit un ordonnancement en choisissant des opérateurs différents pour les opérations qui correspondent au même rang. Par exemple, l'opération* A *et l'opération* B *ont le même rang et elles vont être ordonnancées sur des opérateurs différents comme*

C et D ou E et F. L'opération A peut être ordonnancée sur P_1 à l'instant $t = 0$, l'opération B doit attendre la fin de A afin d'être ordonnancée à l'instant $t = 1$ sur l'opérateur P_2. L'opération D a le rang 1 et aucun prédécesseur, donc elle va être ordonnancée à l'instant $t = 1$ sur P_1. L'opération C a le rang 1 et ses prédécesseurs finissent à l'instant $t = 2$, donc elle va être ordonnancée à l'instant $t = 2$. Finalement on obtient l'ordonnancement donné dans la figure 3.27.

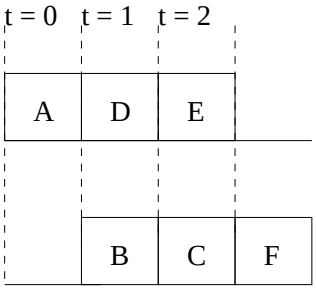

FIG. 3.27 – *Ordonnancement obtenu à partir d'une solution potentielle*

Algorithme 9

Initialisation: On commence par le nombre $u = 0000\ldots0$. On va au Pas 1.
Pas 1 Si u n'est pas égal à $(n-1)(n-1)\ldots(n-1)$ alors on va au Pas 2. Sinon, l'algorithme s'arrête avec la réponse «aucune solution possible ».
Pas 2 A partir de u on construit une solution potentielle. Si elle satisfait les contraintes de précédences et de latences alors l'algorithme s'arrête. Sinon $u := u + 1$ et on va au Pas 1.

Théorème 35 *L'algorithme 9 est de complexité $O(n\,m)$, où n est le nombre d'opérations et m est le nombre d'opérateurs.*

Preuve L'algorithme parcourt n fois l'ensemble d'opérations et pour chaque opération on teste au maximum m opérateurs. On obtient une complexité $\mathcal{O}(n\,m)$. Le théorème est prouvé ⊡

Exemple 32 *On considère le graphe d'algorithme donné dans la figure 3.26 et le graphe d'architecture donné dans la figure 3.23. On définit les contraintes de latence suivantes : $L_{AC} = 3$ et $L_{CF} = 2$. Les durées d'exécution des opérations est égale à 1 sur les deux opérateurs et on considère les durées de communications nulles. La figure 3.28 donne la solution qui satisfait les contraintes de précédences et de latences, solution obtenue en utilisant l'algorithme 9.*

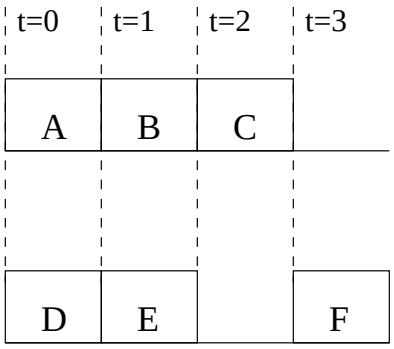

FIG. 3.28 – *Implantation obtenue en utilisant l'algorithme 9*

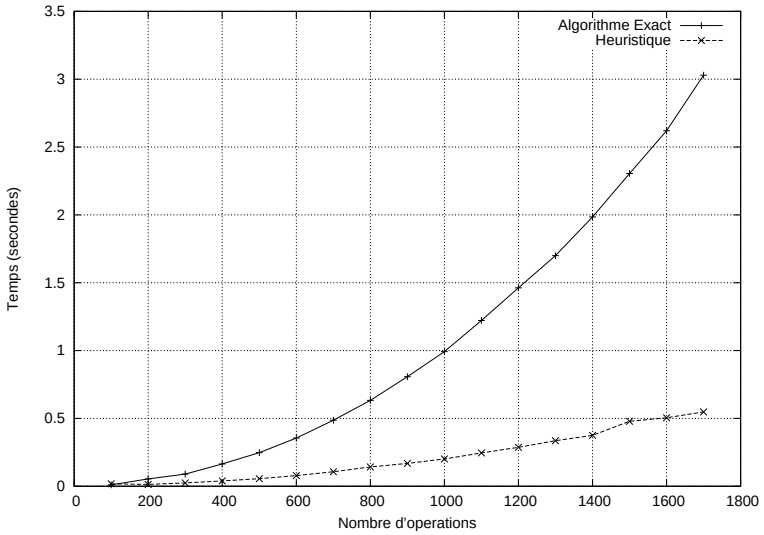

FIG. 3.29 – *Etude de performances comparant l'algorithme exact et l'heuristique*

3.3.4 Etude de performances

On compare les performances de l'heuristique 3 à l'algorithme exact 9. Dans ce cas, l'algorithme exact et l'heuristique partent d'une liste d'opérations dont on connaît les prédécesseurs et les successeurs.

Les graphes de précédences ont été générés d'une manière aléatoire et le temps nécessaire pour leur génération n'est pas pris en compte lors du calcul de temps d'exécution ni pour l'algorithme exact, ni pour l'heuristique. On compare le temps d'exécution pour l'algorithme et l'heuristique pour le même exemple. Pour chaque nombre d'opérations, on génère une seule instance du problème. Ces résultats ont été obtenues en utilisant un Intel Xeon CPU avec une vitesse d'horloge de 2,6 GHz et 1Go de RAM.

La figure 3.29 donne la comparaison entre le temps d'exécution de l'heuristique et de l'algorithme exact. On remarque que l'heuristique est clairement plus rapide que l'algorithme exact. Cette figure prend en compte les meilleurs temps d'exécution pour l'algorithme exact, obtenues pour des solutions trouvées au début de l'énumération. Comparer les pires temps d'exécution de l'algorithme exact à ceux de l'heuristique n'est pas réaliste puisque le temps d'exécution de l'heuristique est trop petit et donc non mesurable. Le tableau 3.6 donne de tels exemples de temps d'exécution de l'algorithme exact. Dans quelques cas le temps d'exécution pour l'heuristique n'est pas spécifié car l'heuristique n'a pas trouvé de solution.

Nombre d'opérations	Temps (algorithme exact)	Temps (heuristique)
2	0.001	0
4	0.002	0
5	0.010	0
6	0.134	–
7	2.450	0
8	56.297	–
9	1549.256	0
10	5000	0

TAB. 3.6 – *Pires temps d'exécution de l'algorithme exact*

3.4 Contraintes de précédences, de périodicités et de latences

On commence ce chapitre par présenter le modèle utilisé afin de spécifier l'algorithme et l'architecture dans le cas des systèmes temps réel avec contraintes de précédences, de périodicités et de latences. On passe ensuite à la complexité du problème d'implantation et on finit par proposer une heuristique comparée à un algorithme exact.

3.4.1 Modèle

On utilise le même modèle que celui présenté dans le chapitre 3.2.1 pour le graphe d'architecture et le graphe d'algorithme sauf que les contraintes imposées aux opérations sont les contraintes de périodicité et de latences.

3.4.2 Complexité du problème d'implantation

Pour prouver la complexité de notre problème, on se rapporte au problème d'implantation avec contraintes de périodicités et de précédences, dans le cas multiprocesseur. On note par PPP le problème d'implantation avec contraintes de périodicités et de précédences, dans le cas multiprocesseur et par $PPPL$ le problème d'implantation avec contraintes de périodicités, de latences et de précédences, dans le cas multiprocesseur. Le théorème suivant donne la complexité de $PPPL$:

Théorème 36 *Le problème $PPPL$ est NP-difficile au sens fort.*

Preuve On remarque que PPP est sous-cas de $PPPL$. En effet, il suffit de ne pas définir des contraintes de latence pour le problème $PPPL$ et on obtient PPP. Étant donné que PPP est NP-difficile au sens fort alors $PPPL$ est NP-difficile au sens fort. Le théorème est prouvé ⊡

3.4.3 Heuristique et algorithme exact

L'heuristique dans ce cas doit prendre en compte toutes les contraintes : de précédences, de périodicités et de latences. Pour les deux heuristiques données dans le cas avec contraintes de périodicité et respectivement le cas avec contraintes de latence, on définit des classes des opérations par rapport soit aux périodes, soit aux latences. Le théorème suivant prouve que toutes les opérations appartenant à la même contrainte de latence appartiennent à la même classe si on considérait les classes dues aux périodes.

Théorème 37 *Les opérations appartenant à la même contrainte de latence appartiennent à la même classe si on considérait les classes dues aux périodes.*

Preuve Le théorème 24 prouve que les opérations appartenant à la même contrainte de latence doivent avoir la même période afin que la contrainte de latence soit satisfaite. En conséquence les opérations appartenant à la même contrainte de latence appartiennent à la même classe si on considérait les classes dues aux périodes. Le théorème est prouvé ⊡

En conséquence si on définit les classes dues aux périodes, on conserve les classes pour les latences.

L'heuristique utilise l'algorithme de marquage pour prendre en compte les contraintes de latence.

Heuristique 4

Pas 1 : on choisit parmi les opérations pour chaque classe avec la plus petite période de $\mathcal{W}$ celle qui a la plus petite durée d'exécution (en choisissant parmi toutes les durées d'une opération A sur tous les opérateurs. On ordonnance ces opérations. On met à jour $\mathcal{W}$, s_{lp} et C_{lp} ;

Pas 2 : pour chaque opérateur p, on cherche une opération $A \in \mathcal{W} \bigcap \mathcal{P}er$ avec A_1 déjà ordonnancée tel que :

$$s_{A_i} + T_A - s_{lp} - C_{lp} = min_{B_i \in \mathcal{W} \bigcap \mathcal{P}er \text{ and } B_1 \text{ déjà ordonnancée}} \{s_{B_i} + T_B - s_{lp} - C_{lp}\}$$

où A_i est la dernière répétition ordonnancée de A. Si on trouve plusieurs opérations A pour le même opérateur alors le système n'est pas ordonnançable avec l'heuristique. On choisit l'opérateur avec la date de fin $s_{A_i} + T_A - s_{lp} - C_{lp}$ la plus petite et on va au Pas 3 ;

Pas 3 : si $\exists C \in \mathcal{W} \bigcap C_{T_A}$ avec C_1 pas ordonnancée, C et A appartenant à la même classe et $s_{lp} + C_{lp} + C_C \leq s_{A_i} + T_A$, pour l'opération A et l'opérateur p trouvés précédemment, alors on a $s_C = s_{lp} + C_{lp}$ et on va au Pas 5. Sinon, on va au Pas 4

pas 4 : on a $s_A = s_{lp} + C_{lp}$ t.q. $s_{A_{i+1}} = s_{A_i} + T_A$ et on va au Pas 5 ;

pas 5 : s'il y a des opérations dans $\mathcal{W}$ appartenant à une classe dont aucune opération a été encore ordonnancée alors on choisit l'opération avec la plus petite période appartenant à la classe celle qui la plus petite durée d'exécution (en choisissant parmi toutes les durées sur tous les opérateurs) et on va au Pas 6 ;

Pas 6 : s'il y a des opérations appartenant à une classe dont les opérateurs utilisés ne permettent plus l'ordonnancement des opérations sans la première répétition ordonnancée, alors on choisit un nouvel opérateur pour cette classe et les opérateurs déjà utilisés par cette classe ne sont plus utilisables. L'opération de la classe qui a la plus petite période et la plus petite durée d'exécution est ordonnancée avec la date de début égale à la plus grande date de fin de ses prédécesseurs. On va au Pas 2.

On présente maitenant l'algorithme exact de type branch & bound. On suppose que l'on a m opérateurs mis dans une liste dans un ordre quelconque. On suppose que l'on a n opérations mises dans une liste selon un ordre total tel que pour une opération quelconque i tous ses prédécesseurs ont un indice plus petit qu'elle et toutes les opérations, qui deviennent disponibles en même temps qu'elle et qui ont une période plus grande qu'elle, ont un indice plus grand. L'exemple suivant suit ce type d'ordre.

Exemple 33 *On considère le graphe d'algorithme donné dans la figure 3.30 et le graphe d'architecture donné dans la figure 3.31. On définit les contraintes de latence suivantes :* $L_{AC} = 3$, $L_{DF} = 4$ *et* $L_{GI} = 5$. *On obtient la liste ordonnée d'opérations* {A,B,C,E,G,H,I}.

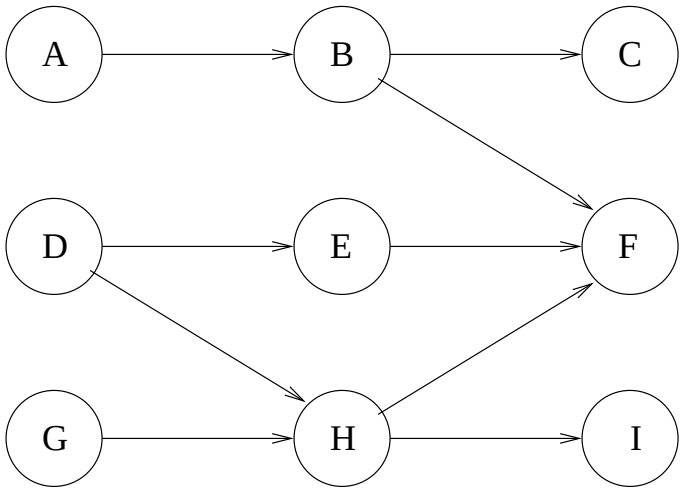

FIG. 3.30 – *Graphe d'algorithme*

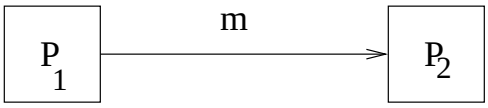

FIG. 3.31 – *Graphe d'architecture*

Algorithme 10

Initialisation : On initialise l'indice pour les opérations $i = 1$ et l'indice pour les opérateurs $j = 1$, la date de début de la dernière opération ordonnancée sur chaque opérateur $s(j) = 0$ et la durée de l'opération $C(j) = 0$;

Pas 2 : Si $i \leq n$ alors on ordonnance l'opération i sur l'opérateur j, $s(j) = s(j) + C(j)$ et $C(j) = C_i$, sinon on s'arrête. S'il y a au moins une contrainte qui n'est pas satisfaite alors $i = i + 1$ et on va au Pas 2. Sinon on va au Pas 3 ;

Pas 3 : Si $j > n$ alors $i = i - 1$ et j prends la valeur $j_i - 1$ où j_i est l'indice de l'opérateur sur lequel l'opération i a été ordonnancée et on va au Pas 2. Sinon on va au Pas 4 ;

Pas 4 : $j = j + 1$ et on va au Pas 2.

Théorème 38 *L'algorithme 10 a la complexité $\mathcal{O}(n\,m)$ où n est le nombre d'opérations et m est le nombre d'opérateurs.*

Preuve L'algorithme parcourt n fois l'ensemble d'opérations et pour chaque opération on teste au maximum m opérateurs. On obtient une complexité $\mathcal{O}(n\,m)$. Le théorème est prouvé ⊡

3.4.4 Etude de performances

On compare l'heuristique 4 à l'algorithme exact 10 afin d'étudier ses performances.

On suppose avoir appliqué les propriétés d'héritage des périodes (voir Corollaire 3) et donc avoir toutes les opérations périodiques. En plus, l'algorithme exact et l'heuristique partent des listes d'opérations où chaque liste contient les opérations disponibles à un moment donné en utilisant en parcours en largeur du graphe de précédences. De cette manière, afin de vérifier si les contraintes sont satisfaites lors de la distribution et l'ordonnancement d'une opération, l'algorithme exact vérifie seulement les contraintes de périodicités et de latences. Donc, comme dans le cas des opérations avec contraintes de précédences et de latences, on n'utilise pas un algorithme exact branch & bound.

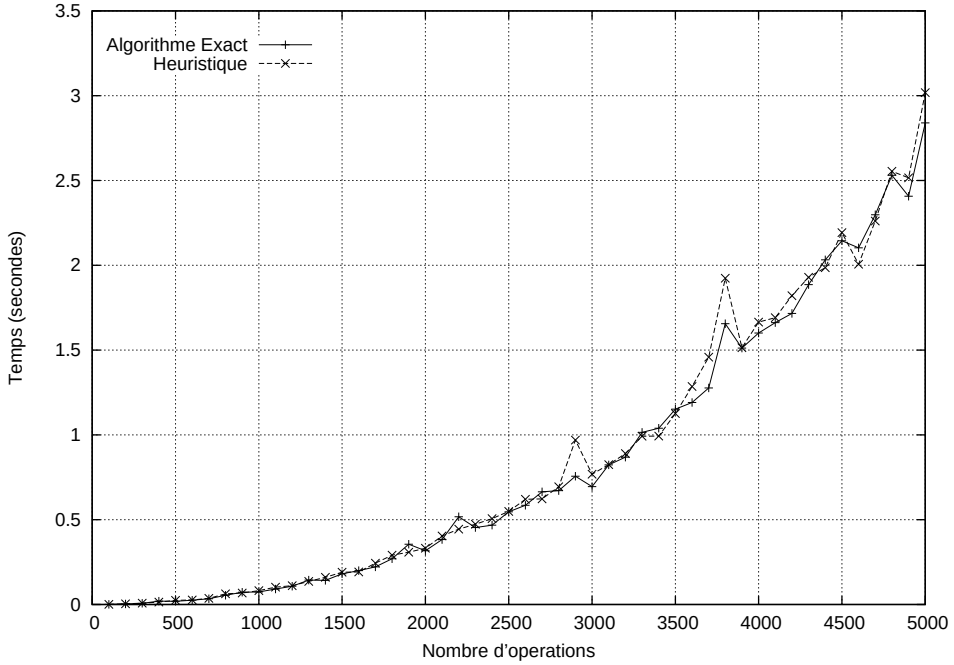

FIG. 3.32 – *Résultats comparant l'algorithme exact et l'heuristique*

Les graphes de précédences ont été générés d'une manière aléatoire et le temps nécessaire pour leur génération n'est pas pris en compte lors du calcul du temps d'exécution ni pour l'algorithme exact, ni pour l'heuristique. On compare le temps d'exécution pour l'algorithme et l'heuristique pour le même exemple. Cette comparaison est donnée par la figure 3.32 qui présente les résultats des simulations numériques. La présence de contraintes de latence parmi les contraintes augmente le temps d'exécution dû aux recherches des chemins ce qui rends les perfomances de l'heuristique très proches de celles de l'algorithme exact.

Conclusion et perspectives

Cette thèse présente un nouveau modèle pour décrire des systèmes temps réel avec contraintes de précédences, de périodicités et de latences. Il permet à la fois de poser clairement les problèmes de distribution et d'ordonnancement, et aussi de préciser les choix faits parmi les approches possibles utilisées dans la communauté du temps réel. Pour le cas monoprocesseur nous posons d'abord le problème d'ordonnancement des systèmes avec contraintes de précédences et de périodicités pour lequel le modèle proposé permet d'imposer des contraintes de précédence entre opérations avec périodes différentes. On donne des résultats concernant la cohérence entre les contraintes de précédences et de périodicités. Toujours dans le cas monoprocesseur, après avoir motivé l'intérêt d'une nouvelle contrainte temps réel de latence, nous posons le problème d'ordonnancement des systèmes avec contraintes de précédences et de latences, ces dernières pouvant être multiples. Pour finir on établit des relations entre des paires d'opérations sur lesquelles on peut imposer des contraintes de latence. On donne des résultats concernant la cohérence entre les contraintes de précédences et de latences. Nous posons finalement dans le cas monoprocesseur le problème d'ordonnancement des systèmes avec contraintes de précédences, de périodicités et de latences. On donne des résultats concernant la cohérence entre les contraintes de périodicités et de latences. Pour le cas multiprocesseur nous posons d'abord le problème d'implantation (distribution et ordonnancement) des systèmes avec contraintes de précédences et de périodicités, en tenant compte du coût des transferts de données entre opérations distribuées sur des processeurs différents. On donne des résultats concernant la cohérence entre les contraintes de périodicités et les transferts de données. Toujours pour le cas multiprocesseur nous posons le problème d'implantation des systèmes avec contraintes de précédences et de périodicités. On donne des résultats concernant la cohérence entre les contraintes de latence et les transferts de données. Nous posons finalement pour le cas multiprocesseur le problème d'implantation des systèmes avec contraintes de précédences, de périodicités et de latences.

Cette thèse dans le cas monoprocesseur donne pour chaque problème traité un algorithme d'ordonnancement optimal et prouve l'existence d'une condition ou d'un test d'ordonnançabilité. Pour le problème d'ordonnancement des systèmes avec contraintes de précédences et de périodicités on prouve l'existence d'une hyper-période et on donne des résultats concernant l'héritage des périodes pour des opérations qui ne sont pas périodiques. Ces derniers résultats sont ensuite étendus pour le problème d'ordonnancement des systèmes avec contraintes de précédences, de périodicités et de latences. Dans le cas multiprocesseur on propose pour chaque problème traité une heuristique d'implantation des opérations sur les processeurs qui

est comparée à un algorithme exact.

Enfin, cette thèse dans le cas monoprocesseur donne un résultat de complexité concernant le problème d'ordonnancement des systèmes avec contraintes de précédences et de périodicités. Dans le cas multiprocesseur on donne pour chaque problème traité un résultat de complexité qui montre que le problème est NP-difficile au sens fort. Toujours dans le cas multiprocesseur on calcule la complexité de chaque algorithme exact.

Une première perspective ouverte par cette thèse vient de l'étude complète sur l'interaction entre les contraintes de précédences et de périodicités et surtout sur les opérations non-périodiques qui soit ont des prédécesseurs périodiques, soit ont des successeurs périodiques. Ces opérations sont des opérations sporadiques, respectivement des opérations apériodiques. Ces résultats concernant l'héritage de périodes forment une base théorique pour des futurs résultats d'ordonnançabilité pour ce type d'opérations, résultats qui manquent dans la littérature.

Comme cela est dit dans l'introduction, le plus grand défi à la suite de cette thèse est l'introduction de la préemption qui impose la modification du modèle proposé actuellement, mais qui, on l'espère, permettra de trouver aussi une condition nécessaire d'ordonnancement pour les cas où on prend en compte la préemption. La modification du modèle concerne surtout la contrainte de périodicité qui actuellement est une contrainte entre les débuts des répétitions successives d'opérations. Il faut étudier si cette définition a toujours de l'intérêt par rapport à la préemption et trouver dans le cas contraire la définition sur laquelle la préemption a vraiment une influence (modifier l'ordonnançabilité d'un système par exemple). Il faut choisir entre les différents cas possibles : début - fin des répétitions successives, fin - fin des répétitions successives et fin - début des répétitions successives.

On s'attend aussi à ce que la préemption ait une influence sur la définition de la latence qui est actuellement une contrainte entre le début d'une opération et la fin d'une autre opération liées par une contrainte de latence. Les différents cas possibles sont : fin - fin, fin - début et début - début.

Une autre perspective concerne les contraintes relatives. La définition de la latence permet d'imposer une contrainte de latence sur deux opérations quelconques. Son caractère général permet d'étendre les résultats obtenus sur l'ordonnançabilité de tels systèmes pour d'autres contraintes relatives.

En combinant la périodicité et la latence on peut prendre en compte la gigue (variations faibles de périodicité) dans notre problème. Cela nous permettra d'étendre notre étude vers la périodicité classique.

Le lien entre notre modèle et le modèle classique permettra de proposer de nouvelles heuristiques pour le cas multiprocesseur du problème classique et aussi d'obtenir des résultats de complexité pour des problèmes utilisant le modèle classique dont la complexité n'est pas encore établie (prouvée).

Bibliographie

[1] K. Jeffay, D.F. Stanat, and C.U. Martel. On non-preemptive scheduling of periodic and sporadic tasks. *IEEE*, 1991.

[2] J.A. Stankovic, M. Spuri, and M. Di Natale. Implications of classical scheduling results for real-time systems. *IEEE Computer*, 1995.

[3] R. H. Rodney and K. V. Muralidhar. On non-preemptive scheduling of recurring tasks using inserted idle times. *Information and Computation*, 1995.

[4] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 1973.

[5] J. Leung and M. L. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 1980.

[6] E. Grolleau and A. Choquet-Geniet. Cyclicité des ordonnancements de systèmes de tâches périodiques différées. *Real-time and Embedded Systems*, 2000.

[7] J.P. Lehoczky, L. Sha, and Y Ding. The rate monotonic sheduling algorithm: exact characterization and average case bahavior. *Proceedings of the IEEE Real-Time Systems Symposium*, 1989.

[8] J. Orozco, R. Santos, J. Santos, and E. Ferro. Hybrid rate-monotonic/reward-based scheduling of real-time embedded systems. *IEEE Real-Time Embedded System Workshop*, 2001.

[9] J.P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. *IEEE*, 1990.

[10] N.C. Audsley, Burns A., M.F. Richardson, and A.J. Wellings. Hard real-time scheduling : The deadline-monotonic approach.

[11] S. Pailler and A. Choquet-Geniet. Ordonnancement temps réel d'applications comportant des tâches à durées variables. *Real-time and Embedded Systems*, 2002.

[12] K. Tindell. Adding time-offsets to schedulability analysis.

[13] K.W. Tindell. Using offset information to analyse static priority pre-emptively scheduled tasks sets.

[14] J. Goossens and R. Devilliers. The non-optimality of the monotonic assignments for hard real-time offset free systems. 1997.

[15] A. Choquet-Geniet, D. Geniet, and F. Cottet. Exhaustive computation of the scheduled task execution sequences of a real-time application. *4ème International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, 1996.

[16] T.P. Baker. Stack-based scheduling of realtime processes. *The Journal of Real-time Systems*, 1991.

[17] N.C. Audsley, A. Burns, M.F. Richardson, Tindell K., and A.J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineerung Journal*, 1993.

[18] C. D. Locke. Software architecture for hard real-time applications: cyclic executives vs. fixed priority executives. *Real-time Systems (Netherlands)*, march 1992.

[19] G. Bernat and A. Burns. Weakly-hard real-time systems. *IEEE Transactions on Computers(50)4*, 2001.

[20] G. Bernat and A. Burns. Combining (n,m)-hard dealines and dual-priority scheduling. *Real-Time Systems Symposium*, 1997.

[21] E.-M. Poggi, Y.-Q. Song, A. Koubaa, and Z. Wang. Matrix-dbp for (m,k)-firm rela-time guarantee. *10ème Real-Time Systems, Paris*, 2003.

[22] X. Yuan and A.K. Agrawala. A decomposition approach to non-preemptive scheduling in hard real-time systems. *IEEE*, 1989.

[23] J.H.M. Korst, E.H.L. Aarts, and J.K. Lenstra. Scheduling periodic tasks. *INFORMS Journal on Computing 8*, 1996.

[24] J.H.M. Korst, E.H.L. Aarts, and J.K. Lenstra. Scheduling periodic tasks with slack. *ORSA Journal on Computing.*

[25] Gerber R., S. Hong, and M. Saksena. Guaranteeing real-time requirements with resource-based calibration of periodic processes. *IEEE Transactions on Software Engineering(21)7*, 1995.

[26] D.-I. Kang, Gerber R., and M. Saksena. Performance-based design of distributed real-time systems.

[27] R. Gerber, W. Pugh, and M. Saksena. Parametric dispatching of hard real-time tasks. *IEEE Transactions on Computers(44)3*, 1995.

[28] W. Lindsay and Ramanathan P. Dbp-m, a technique for meeting end-to-end (m,k)-firm constraints guarantee requirements in point-to-point networks. *IEEE Conference on Local networks*, 1999.

[29] E.L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. Technical report, University of California, 1971.

[30] J.K. Lenstra and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discrete Math.*, 1977.

[31] W.A. Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 1961.

[32] J. Valdes, R.E. Tarjan, and Lawler E.L. The recognition of series parallel digraphs. *SIAM J. Comput.*, 1982.

[33] E.L. Lawler. Recent results in the theory of machine scheduling. *Mathematical Programming: the State of the Art, Springer-Verlag*, 1983.

[34] J. Carlier and P. Chrétienne. Problèmes d'ordonnancement. *Masson*, 1988.

[35] A. Blum, P. Chasalani, D. Coppersmith, B. Pulleyblank, and P. Raghavan. The minimum latency problem. Technical report, IBM Research Division, 1993.

[36] J. Xu and D.L. Parnas. Scheduling processes with release times, deadlines, precedence and exclusion relations. Technical report, York University, 1988.

[37] M.G. Harbour, M.H. Klein, and J.P. Lehoczky. Fixed priority scheduling of periodic tasks with varying execution priority. *IEEE*, 1991.

[38] M.G. Harbour, M.H. Klein, and J.P. Lehoczky. Timing analysis for fixed-priority scheduling of hard real-time systems. *IEEE Transactions on Software Engineering*, 1994.

[39] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence contraints. *The Journal of Real-Time Systems*, 1990.

[40] M. Spuri and J.A. Stankovic. How to integrate precedence contraintes and shared resources in rea-time scheduling. *IEEE Transactions on Computers (43)*, 1994.

[41] E.A. Lee and Messerschmitt D.G. Synchronous data flow. *IEEE Transactions on Computers*, 1987.

[42] S. Baruah, S. Goddard, and K. Jeffay. Feasibility concerns in pgm graphs with bounded buffers. *Proc. of the Third Intl. Conference on Engineering of Complex Computer Systems*, 1997.

[43] S. Goddard. *Constraints on data-flow*. PhD thesis, University of North Carolina, 2000.

[44] J.H.M. Korst. *Periodic Multiprocessor Scheduling*. PhD thesis, University of Eindhoven, 1992.

[45] J. Sun and J.W.S. Liu. Bounding completion times of jobs with arbitrary release times and variable execution times. *Real-Time Systems Symposium*, 1996.

[46] F. Baccelli, B. Gaujal, and D. Simon. Analysis of preemptive periodic real-time systems using the (max, plus) algebra. *IEEE Transactions on Control Systems Technology*, 2002.

[47] P. van Beek and K. Wilken. Fast optimal instruction scheduling for single-issue processors with arbitrary latencies. *Springer-Verlag*, 2001.

[48] A. Munier. The cyclic scheduling problem with linear precedence constraints. *Discrete Applied Mathematics*, 1996.

[49] J. Leung and Whitehead J. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation(4)*, 1982.

[50] R. Garey, Michael and S. Johnson, David. *Computers and intractability : a guide to the theory of NP-completeness*. W.H. Freeman, 1979.

[51] M.W. Mutka and J.-P. Li. A tool for allocating periodic real-time systems to a set of processors. *Journal Systems Software(29)*, 1995.

[52] J. Santos, E. Ferro, J. Orozco, and R. Cayssials. A heuristic approach to the multitask-multiprocessor assignement problem using empty-slots method and rate monotonic scheduling. *Journal of Real-Time Systems(13)*, 1997.

[53] P. Altenbernd and H. Hansson. The slack method: A new method for static allocation of hard real-time tasks. *Journal of Real-Time Systems*, 1997.

[54] J. Orozco, R. Cayssials, J. Santos, and E. Ferro. Precedence constraints in hard real-time systems. *3rd International Conference on Engineering of Complex Computer Systems*, 1997.

[55] T. Grandpierre, C. Lavarenne, and Y. Sorel. Optimized rapid prototyping for real time embedded heterogeneous multiprocessors. *Codes'99 7th International Workshop on Hardware/Software Co-Design*, 1999.

[56] M. Richard, P. Richard, and F. Cottet. Affectation optimale des priorités des tâches et des messages dans les systèmes distribués temps réel. *Real-Time Systems, Paris*, 2002.

[57] B.P. Dave, G. Lakshminarayana, and N.K. Jha. Cosyn: Hardware-software co-synthesis of embedded systems. *ACM*, 1997.

[58] D.-I. Oh and T.P. Baker. Utilization bounds for n-processor rate monotone scheduling with static processor assignment. *Journal of Real-Time Systems*, 1998.

[59] K. Hong and J. Leung. On-line scheduling of real-time tasks. *Real-Time Systems Symposium*, 1988.

[60] S.K. Dhall and C.L. Liu. On a real-time scheduling problem. *Operations Research(26)*, 1978.

[61] J. Goossens, S. Baruah, and S. Funk. Real-time scheduling on multiprocessors. *Real-Time Systems, Paris*, 2002.

[62] K. Tindell and J. Clark. Holistic schedulability for distributed hard real-time systems. *Microprocessing and Microprogramming*, 1994.

[63] M. Richard, P. Richard, and F. Cottet. Allocating and scheduling tasks in multiple fieldbus real-time systems. *Emerging Technologies and Factory Automation*, 2003.

[64] Y. Oh and S.H. Son. Allocating fixed-priority periodic tasks on multiprocessor systems. *Real-Time Journal*, 1995.

[65] D.-T. Peng, K.G. Shin, and T.F. Abdelzaher. Assignement and scheduling communicating periodic tasks in distributed real-time systems. *IEEE Transactions on Software Engineering*, 1997.

[66] J. Jonsson and J. Vasell. Evaluation and comparison of task allocation and scheduling methods for distributed real-time systems. *IEEE Workshop on Real-Time Applications*, 1996.

[67] J.P. Penny, P.J. Ashton, and A.L. Wilkinson. Data recording and monitoring for analysis of system reponse times. *The Computer Journal(29)5*, 1986.

[68] S. Saez, J. Vila, and A. Crespo. Using exact feasibility tests for allocating real-time tasks in multiprocessor systems. *10th Euromicro Workshop on Real-Time Systems*, 1998.

[69] J.P. Beauvais and A.M. Déplanche. Affectation de tâches dans un système temps réel réparti. *Technique et Science Informatique(17)*, 1998.

[70] P. Bratley, M. Florian, and P. Robillard. Scheduling with earliest start and due date contraints on multiple machines. *Naval Research Logistic Quarterly(22)*, 1975.

[71] M. Hamdaoui and P. Ramanathan. Evaluating dynamic failure probability for streams with (m,k)-firm constraints. *IEEE Transactions on Computers(46)12*, 1995.

[72] S.B. Shukla and D.P. Agrawal. A framwork for mapping periodic real-time applications on multicomputers. *IEEE Transactions on Parallel and Distributed Systems(5)7*, 1994.

[73] K. Ramamritham. Allocation and scheduling of precedence-related periodic tasks. *IEEE Transactions on Parallel and Distributed Systems*, 1995.

[74] C. Martel. *Deterministic and Stochastic Scheduling*. M.A.H. Dempster, 1985.

[75] A. Aggarwal and A.K. Chandra. On communication latency in pram computations. Technical report, IBM Research Division, 1989.

[76] S. Baruah. The non-preemptive scheduling of periodic tasks upon multiprocessors. *TR*, 2002.

[77] K. Tindell, A. Burns, and A. Wellings. Allocation hard real-time tasks (a np-hard problem made easy). *Real-Time Systems*, 1992.

[78] Albert Benveniste and Gérard Berry. The synchronous approach to reactive and real-time systems. *Proceedings of the IEEE*, 79(9):1270–1282, September 1991.

[79] J.B. Dennis. First version of a data flow procedure language. Technical report, Massachusetts Institute of Technology, 1975.

[80] P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problem. *SIAM Journal Discrete Mathematics*, 1989.

[81] R.B. Hagmann. Low latency logging. Technical report, Palo Alto Research Center, 1991.

[82] L. Cucu, R. Kocik, and Y. Sorel. Real-time scheduling for systems with precedence, periodicity and latency constraints. *Real-time and Embedded Systems*, 2002.

[83] L. Cucu and Y. Sorel. Schedulability condition for systems with precedence and periodicity constraints without preemption. *Real-time and Embedded Systems*, 2003.

[84] L. Cucu and Y. Sorel. Schedulability condition for real-time non-preemptive systems with precedence and latency contraints. *RR INRIA à apparaître*.

[85] R. Balakrishnan and K. Ranganathan. *A Textbook of Graph Theory*. Springer, 2000.

[86] S.K. Baruah, R.R. Howell, and L.E. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor. *Journal of Real-Time Systems*, 1990.

[87] Y Sorel. Massively parallel computing systems with real time constraints, the "algorithm architecture adequation" methodology. *Proceedings of Massively Parallel Computing Systems, Italy*, 1994.

[88] T. Grandpierre, C. Lavarenne, and Y. Sorel. Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors. *7th International Workshop on Hardware/Software Co-Design, Rome*, 1999.

[89] A. H. Land and A. G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 1960.

[90] M. Gondran and M. Minoux. *Graphes et algorithmes*. Eyrolles, 1985.

RESUME : Après un état de l'art sur l'ordonnancement en général et l'ordonnancement temps réel en particulier permetttant de préciser les notions utilisées par la suite et après avoir motivé l'intérêt d'une nouvelle contrainte temps réel de latence, nous proposons un modèle qui formalise les systèmes temps réel avec contraintes de précédences, de périodicités et de latences. Dans ce modèle, les précédences sont définies par un graphe orienté acyclique infiniment répété. Pour le cas monoprocesseur, on étudie trois problèmes d'ordonnancement : celui des systèmes avec contraintes de précédences et de périodicités, celui des systèmes avec contraintes de précédences et de latences et enfin celui des systèmes avec contraintes de précédences, de périodicités et de latences. Pour chaque problème on étudie la cohérence entre les différentes contraintes, on donne des conditions d'ordonnançabilité et on propose un algorithme prouvé optimal dans le sens où s'il y a un ordonnancement, l'algorithme le trouvera. On passe ensuite au cas multiprocesseur où l'architecture est définie par un graphe non-orienté. On étudie trois problèmes d'implantation (distribution et ordonnancement) dans les mêmes cas qu'en monoprocesseur en tenant compte des temps de communications. On prouve que ces trois problèmes sont NP-difficiles et on propose, donc, des heuristiques. Les performances de chaque heuristique sont comparées à celles d'algorithme exacte de type "branch and bound", en utilisant des simulations numériques.

Mots-clé : temps réel, graphes, ordonnancement, ordonnançabilité, distribution, algorithmes, heuristiques, complexité, multiprocesseur, monoprocesseur.

ABSTRACT: After a state of art on classical scheduling and on real-ime scheduling, particularly, allowing to define the notions used afterwards and after motivating a new real-time constraint of latency, we propose a model which describes the real-time systems with precedence, periodicity and latency constraints. In this model, the precedences are defined using a directed acyclic graph. In the monoprocessor case, we study three problems of scheduling: systems with precedence and periodicity constraints, systems with precedence and latency constraint and systems with precedence, periodicity and latency constraints. For each problem we study the relation between the precedences and the periodicity, respectively the latency, we give schedulability conditions and we propose scheduling algorithms which are proved optimal (if there is a schedule, the algorithm will find it). For the multiprocessor case the architecture is defined by a non-oriented graph. We study three multiprocessor scheduling problems in the same cases as in monoprocessor, taking into account communication times . For each problem the model takes into account the communications . We prove that each problem is NP-hard and we propose heuristics. The performances of each heuristic is compared to those of an exact algorithm of type branch and bound, using numerical simulations.

Keywords: real-time, graphs, scheduling, schedulability, distribution, algorihms, heuristics, complexity, multiprocessor, monoprocessor.