THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :
Institut National des Sciences Appliquées de Toulouse (INSA Toulouse)

Discipline ou spécialité :

Systèmes Informatiques et Systèmes Embarqués

Présentée et soutenue par :
Ahmad AL SHEIKH

le : 28 Septembre 2011

Titre :

Resource allocation in hard real-time avionic systems
-
Scheduling and routing problems

Ecole doctorale :
Systèmes (EDSYS)

Unité de recherche :
LAAS-CNRS

Directeur(s) de Thèse :
Olivier BRUN (LAAS-CNRS)
Pierre-Emmanuel HLADIK (LAAS-CNRS)

Rapporteurs :

Sanjoy BARUAH (University of North Carolina)
Yves SOREL (INRIA)

Membre(s) du jury :

Frédéric BONIOL (ONERA)
Joël GOOSSENS (Université Libre de Bruxelles)

Acknowledgements

First and foremost, my appreciation goes to my doctoral advisors, Dr. Olivier Brun and Dr. Pierre-Emmanuel Hladik for their support throughout my PhD. They have successfully provided me with a fulfilling and motivating work environment in which I have conducted my research. I thank them for this unique experience for which I am deeply grateful.

I would also like to offer my appreciation to Dr. Balakrishna Prabhu for his support and contribution in many occasions. His intervention had a great impact on the direction this thesis has took.

I would like to thank the Jury for having accepted to participate in my PhD defense. Beginning with Professor Sanjoy Baruah and Professor Yves Sorel who have consecrated much of their time to review my dissertation and give detailed feedback on my work, to Professor Frédéric Boniol and Professor Joël Goossens who participated in examining my work and providing valuable insights to the improvement of the quality of this thesis.

I would like to acknowledge all the participants in the research project SATRIMMAP for the support they have given me throughout the 3 years of the PhD. This support was of utmost importance and was essential to the accomplishment of the objectives set before us.

I am extremely grateful for the mates I got to know and share offices with in the Laboratory of Architecture and Analysis of Systems. Among them, I would like to mention Rémi Sharrock, Jean-Marie Codol and Aurélien Gonzalez, for whom I offer my fondest regards for all of the time we have passed together. I hope our friendship continues on.

I wish to thank my friends outside work and whom I get to call my second family: Alaa Allouch, Houssam Arbess, Yahya Salma, Nadim Nasreddine and several others. You were there next to me when I was in need. Without your company, life wouldn't have been the same abroad.

Finally, I would like to dedicate this thesis to my parents, Mustafa and Mervat, who have raised, taught, supported and guided me throughout my life.

Abstract

The last couple of years have seen a profound evolution in embedded architectures with the introduction of Integrated Modular Avionics (IMA). By offering to embedded applications a standardized execution and communication support, these architectures have allowed the consequent reduction of physical weight and complexity. This low level reduction of complexity is opposed by an increased difficulty in application conception and integration, as managing resource sharing is through numerous configuration parameters. This thesis is devoted to two resource allocation problems that arise in conception phases of IMA-based architectures.

The multiprocessor scheduling problem is first addressed for strictly periodic tasks, or in other words tasks that execute indefinitely in equidistant time intervals. The objective is supposed to be the maximization of the minimal idle time between two tasks while avoiding overlap in temporal executions. This allows guaranteeing a minimal evolution margin for the task executions. An integer linear programming based formulation is first proposed for this NP-hard problem, integrating all associated temporal and resource constraints. To extend scalability, a heuristic inspired from Game Theory is equally introduced. In this heuristic, each task adopts a scheduling that maximizes its proper utility function (related to the evolution margin of tasks). The convergence of this algorithm towards an equilibrium point, where no task has an interest in modifying its strategy, is shown in addition to the presence of a globally optimal equilibrium. The obtained numerical results show that this algorithm is much faster than the exact method and gives a good approximation. To further ameliorate the quality of obtained solutions, multi-start methods can be applied to this algorithm to supply probabilistic guarantees on the optimality of attained equilibria.

In the second part of the thesis, the message routing problem between avionic functions in the AFDX network is considered. This network allows the transmission of Ethernet frames in what is called virtual links (VL). Each VL can be seen as a multicast tree allowing data transmission from one point of the network to several others. An exact node-link linear formulation is first introduced. This is followed by the proposition of a two-level heuristic that compromises between the fair load distribution and delay minimization in the network. The obtained results show that solutions obtained by the heuristic can be very close to those of the exact method while significantly ameliorating communication delays.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Résumé étendu

Dans le domaine avionique, les architectures embarquées connaissent depuis une dizaine d'années une mutation profonde avec l'apparition des architectures modulaires intégrées (IMA). En offrant aux applications embarquées un support d'exécution et de communication standard et mutualisé, ces architectures ont permis une réduction du poids et de la complexité de l'architecture physique. Cette réduction de la complexité au niveau bas s'est cependant traduite par une difficulté accrue de conception et d'intégration des applications car il faut gérer le partage des ressources au moyen de nombreux paramètres de configuration. Cette thèse est consacrée à deux problèmes d'allocation de ressources qui se posent dans le processus de conception d'une architecture IMA.

Nous étudions tout d'abord le problème de l'ordonnancement multiprocesseur de tâches strictement périodiques, c'est-à-dire des tâches qui s'exécutent à intervalles de temps constants sur un horizon de temps infini. L'objectif est de maximiser la durée minimale d'inactivité entre deux exécutions de tâches tout en garantisant que les intervalles pendant lesquels elles s'exécutent ne se chevauchent pas. Ceci garantit une marge d'évolution minimale des budgets de temps alloués aux tâches. Nous proposons en premier une formulation exacte sous la forme d'un programme linéaire en nombres entiers intégrant de nombreuses contraintes temporelles et de ressource.

Pour permettre le passage à l'échelle, une heuristique inspirée de la théorie des jeux a été développée dans laquelle chaque tâche adapte à son tour son ordonnancement pour maximiser sa propre fonction d'utilité (qui est liée aux marges d'évolution des tâches). Nous montrons la convergence de cet algorithme vers un point d'équilibre dans lequel aucune tâche n'a intérêt à modifier unilatéralement sa stratégie et établissons qu'il existe au moins un équilibre qui est globalement optimal. Les résultats numériques obtenus montrent que cet algorithme est beaucoup plus rapide que la méthode exacte et fournit une bonne approximation. Pour améliorer encore la qualité des solutions, nous utilisons cette heuristique dans un algorithme multi-start qui offre des garanties probabilistes sur l'optimalité des équilibres atteints.

Dans une seconde partie, nous considérons le problème du routage des messages échangés entre les fonctions avioniques. Le réseau utilisé permet la transmission de trames Ethernet dans des liens

virtuels (VL) qui se partagent les ressources du réseau de transmission. Chaque VL peut être vu comme un arbre multicast permettant la transmission de données depuis un point du réseau vers un ou plusieurs autres. Nous proposons tout d'abord une formulation exacte du problème de routage des VL sous la forme d'un programme linéaire en nombres entiers de type noeuds-liens. Nous étudions ensuite une heuristique à deux niveaux qui permet de trouver un compromis entre une distribution équitable de la charge dans le réseau et la minimisation du délai de communication. Les résultats obtenus avec cette heuristique montrent qu'elle permet d'avoir des solutions très proches de la méthode exacte tout en améliorant significativement les délais de communication.

1 Présentation de la plate-forme IMA

Un système avionique est composé d'un ensemble de logiciels, d'unités de traitement, de bus de communication, de senseurs et d'actuateurs qui doivent réaliser diverses fonctionnalités soumises à des contraintes critiques de type temps réel. La conception de tels systèmes nécessite de nombreux efforts en particulier pour réduire son poids et sa consommation. À cela s'ajoute une demande constante de prise en charge de nouvelles fonctionnalités pour en améliorer la qualité. Cela a conduit à la standardisation de leurs différents éléments, mais aussi à une augmentation de leur complexité.

A la fin des années 1990, une nouvelle architecture pour les plate-formes avioniques, baptisée *Integrated Modular Avionics* (IMA), a été définie. La nouveauté introduite avec ces plate-formes consiste principalement à mutualiser les ressources d'exécution et de communication entre différents sous-systèmes. Ces derniers doivent alors être soumis à des mécanismes de ségrégation assurant leur intégrité et leur isolation. La mise en commun des ressources permet ainsi de réduire leur nombre et de standardiser les supports d'exécution et de communication.

Cependant le partage des ressources induit aussi de nouvelles difficultés dans la conception des systèmes, en particulier en ce qui concerne le partage du temps, de l'espace mémoire et du réseau entre les différents sous-systèmes.

1.1 Composants de la plate-forme IMA

L'IMA est spécifiée par l'ARINC 651 [3]. Les ressources sont désignées d'une manière générique en tant que *Line Replaceable Unit* (LRU). Elles sont principalement de trois types : *Core Processing Modules* (CPM) qui sont les modules responsables de l'exécution des applications ; *Input/Output Modules* (IOM) qui assurent les communication avec les éléments qui ne sont pas IMA, comme le *Remote Data Concentrator* (RDC) qui convertit les messages entre le système et les senseurs et actuateurs ; commutateurs et passerelles qui assurent le support physique de la communication.

Les CPM sont regroupés dans des cabinets qui sont des unités avec une alimentation propre. L'architecture logicielle est basée sur le standard APEX [8] (*APplication EXecutive*). Celui-ci offre une interface générique aux applications pour accéder aux services du système d'exploitation.

Les application avioniques sont responsables de plusieurs fonctionnalités dans un avion. Elles sont décomposées en un ensemble de fonctions qui ont leur propre flux d'exécution et de données. Ces fonctions résident sur les modules d'exécution et sont partitionnées. Dans la suite, nous ne parlerons plus que de partition pour désigner un sous-ensemble cohérent de fonctions associées à des contraintes de temps et de ressource.

1.1.1 Ségrégation des partitions

Pour éviter la propagation de faute, deux mécanismes principaux sont mis en œuvre dans l'IMA : la ségrégation spatiale et la ségrégation temporelle. La première consiste à allouer de manière statique les ressources d'exécution (mémoire, nombre de port de communication, etc.) en assurant une intersection nulle. Quant à la ségrégation temporelle, elle consiste à allouer des tranches de temps sans recouvrement pour que les partitions s'exécutent. Ces tranches sont fixes et déterminées en fonction des périodes, durées et échéances des partitions.

En avionique, l'ordonnancement des partitions est donc pré-défini de manière à respecter les différents aspects temps réel du systèmes et assure que l'exécution de chaque fonction est possible dans une durée impartie. Dans le cadre de ce travail, nous verrons que les partitions sont caractérisées par un budget temporel et une période qui impose une stricte répétition des tranches de temps allouées à une partition. De plus, la séquence d'ordonnancement sur un module est définie sur une *Major Frame* (MAF) qui a une durée fixe égale au plus petit commun multiple des périodes des partitions. La MAF contient l'ensemble des dates auxquels les partitions s'exécutent et est jouée périodiquement par le système d'exploitation.

1.1.2 Réseau de communication

La communication entre les modules est spécifiée par l'ARINC 659 [4] et s'appuie sur un réseau commuté Ethernet redondant et fiabilisé, nommé AFDX ("Avionics Full DupleX switched Ethernet"). Le routage à travers les commutateurs du réseau est basé sur les adresses MAC et aussi sur la notion de *Virtual Link* (VL). Un VL est un chemin logique unidirectionnel à travers le réseau qui a une unique source et peut avoir de multiple récepteurs [14]. Ces liens sont définis par des tables statiques dans chaque commutateur.

Pour assurer un comportement prédictible, chaque VL a une bande passante limitée et deux caractéristiques : le *Maximum Frame Size* (MFS) qui fixe la taille maximale d'une trame qui peut être transportée à travers le VL et le *Bandwidth Allocation Gap* (BAG) qui impose une durée minimale entre deux envois successifs de trames.

Les messages produits par une partition qui ont le même chemin dans le réseau AFDX peuvent être regroupés dans une même message afin de minimiser le nombre de bits transférés.

1.2 Allocation de ressources dans les systèmes avioniques

Le travail mené dans cette thèse est réalisé dans le cadre du projet de recherche “Safety and Time Critical Middle-ware for future Modular Avionics Platform” (SATRIMMAP) financé par l’Agence National de la Recherche (ANR). Le consortium du projet est composé de six partenaires : Airbus, CEA LIST, IRIT, LAAS-CNRS, ONERA et QoS Design.

Le principal objectif de SATRIMMAP est d’apporter des solutions pour réduire les difficultés liées aux phases d’intégration et de configuration des avions. Pendant ces phases, des tables de configuration des ressources (entrée-sortie, mémoire, ordonnancement, etc.) sont définies. La principale difficulté est d’en garantir la cohérence, en particulier lors des changements du système ou suite à l’ajout de nouveaux éléments, comme l’ajout de nouvelle fonctions.

Le projet a pour ambition de fournir de nouveaux outils pour aider à la configuration et respecter les contraintes du système. Pour cela trois points ont été abordés:

- la configuration des architectures IMA,
- la formalisation d’un modèle de configuration pour les applications embarquées,
- la définition et réalisation d’outils pour aider à la configuration.

La figure 1 décrit les différents outils dédiés à l’allocation des ressources qui ont été réalisés pendant le projet. L’outil d’analyse de latence ne concerne pas le travail présenté dans cette thèse et a été développé par un autre partenaire. Il a pour but d’évaluer les délais dans le réseau et de produire un retour qui précise si la configuration est valide ou non.

La présente thèse se focalise autour de deux autres problèmes liés aux outils de configuration :

- La distribution des partitions sur les modules d’exécution et leur ordonnancement temporel sous contraintes.
- Le routage des messages, sous la forme de VL, sur le réseau AFDX.

2 Etat de l’art

L’allocation des ressources sur une plate-forme avionique s’inscrit dans le contexte des systèmes temps réel dont nous retiendrons la définition donnée par J.-P. Elloy dans [57] :

> “est qualifié de temps réel le comportement d’un système informatique lorsqu’il est assujetti à l’évolution dynamique d’un procédé qui lui est connecté, et qu’il doit piloter ou suivre en “réagissant” à tous ses changements d’état.” La terminologie “temps réel” cache donc la notion d’un temps de réaction relatif aux dynamiques du procédé à contrôler : le système de contrôle doit réagir en temps contraint aux évolutions du procédé sachant que la valeur de ces temps de réaction peut être à la fois conditionnée par les dynamiques internes du procédé et le respect d’une cadence de production.”

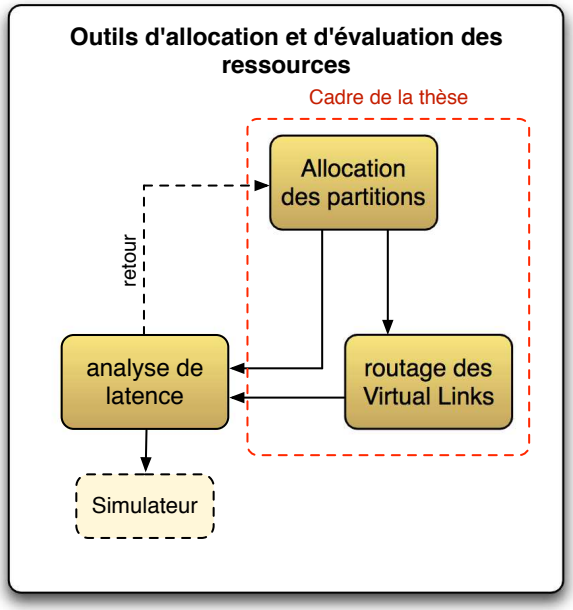


Figure 1: Outillage pour l'allocation de resource.

Ainsi une application temps réel ne doit pas simplement répondre à des exigences fonctionnelles, c'est-à-dire produire des valeurs correctes, mais aussi à des contraintes temporelles, c'est-à-dire produire des résultats "à temps".

De tels systèmes sont couramment déployés dans des domaines aussi variés que l'avionique, les télécommunications, l'automobile, le spatial mais aussi le contrôle de chanes de production, de processus chimiques, etc. D'une manière générale, ces systèmes sont constitués d'un ensemble de calculateurs spécialisés qui reçoivent des données de capteurs, puis les traitent et calculent des commandes adéquates, qu'ils envoient vers des actionneurs. Outre les aspects fonctionnels et temporels, d'autres problèmes se posent pour les systèmes temps réel, spécialement dans le domaine des systèmes embarqués: optimisation du volume et du poids du matériel, réduction de la consommation électrique pour accrotre l'autonomie, limitation des ressources requises en fonction des capacités du matériel disponible, par exemple, taille de la mémoire, puissance du calculateur, etc.

Les principales caractéristiques de la plate-forme IMA présentées dans la partie précédente permettent de la classer parmi les systèmes temps réel – nous utiliserons dans cette partie le terme de tâche pour désigner les entités logicielles d'exécution – :

- A contraintes temporelles dures : le système ne doit jamais violer les contraintes temporelles.
- A contraintes périodiques strictes : la durée séparant deux débuts d'exécution d'une tâche doit être strictement égale à sa période.
- Non-préemptif : lorsqu'une tâche commence son exécution elle ne peut être interrompue.

- Partitionnés : les tâches sont allouées de manière statique sur les supports d'exécution et sont toujours exécutées au même endroit.
- Distribués et à latence contrainte : les tâches sont réparties sur différents processeurs et les durées des échanges de données entre les tâches doivent être matrisées.
- Hors-lignes : l'ordonnancement est pré-calculé avant l'exécution du système.

De plus, le problème d'allocation considéré s'inscrit dans le cadre des systèmes distribués contraint pour lesquels :

- Les différentes ressources de calcul sont asynchrones : les modules ne disposent pas d'une horloge commune.
- Les capacités des ressources sont cumulatives : la consommation d'une ressource est représentée par la somme des besoins des tâches allouées à cette ressource, par exemple les besoins mémoire.
- Les tâches sont sujettes à des contraintes de localisation : l'allocation des tâches peut être contraint à un sous-ensemble de modules ou bien par la nécessité de la localiser avec une autre tâche.

Trouver un ordonnancement qui respecte l'ensemble de ces contraintes est un problème NP-difficile [24]. Remarquons que ce type d'ordonnancement ne nécessite pas d'être vérifié *a posteriori* puisqu'il est correct par construction.

2.1 Littérature sur l'ordonnancement non-préemptif et strictement périodique

Le problème d'ordonnancement multiprocesseur partitionné, non-préemptif, strictement périodique n'a pas beaucoup été abordé dans la littérature avant ces dernières années. La combinaison des aspects non-préemptif et strictement périodique rend le problème particulièrement complexe. Dans la majorité des travaux, seule une périodicité relative, c.-à-d. que les activations sont périodiques mais pas les dates de début d'exécution, est considérée introduisant ainsi des gigues entre les exécutions successives des tâches périodiques [93].

Parmi les premiers travaux autour de la périodicité stricte, ceux de Korst [91] ont été conduits sur des systèmes de traitement vidéo temps réel. Le problème abordé est celui de l'ordonnancement non-préemptif strictement périodique dans le but de minimiser le nombre de ressources d'exécution. Pour cela, Korst propose diverses heuristiques gloutonnes. Il établit aussi une condition nécessaire et suffisante d'ordonnançabilité pour deux tâches strictement périodiques. Dans [92], Korst *et al*. montrent que le problème est NP-complet au sens fort, même dans le cas d'un unique processeur, mais qu'il est résoluble en un temps polynomial si les périodes et les temps d'exécution sont divisibles. Ils proposent aussi une heuristiques basée sur l'allocation successive de tâches sur les processeurs suivant des règles de priorité.

Kermia et Sorel [88] ont aussi travaillé sur une heuristique pour l'ordonnancement de tâches non-préemptives strictement périodiques sur une architecture multiprocesseur. Leur objectif est de minimiser le temps de cycle de l'ordonnancement tout en respectant les latences et les contraintes de

précédence. Leur heuristique est constituée de plusieurs algorithmes, dont l'un alloue les tâches sur les processeurs pour ensuite construire l'ordonnancement. L'allocation est faite de manière à favoriser les tâches qui ont des périodes égales ou multiples sur le même processeur. La comparaison avec un algorithme de *branch and cut* exact montre l'efficacité de cette approche. Meumeu et Sorel [116] puis Marouf et Sorel [114] proposent différentes conditions d'ordonnançabilité dans un contexte similaire.

Dans [55], Eisenbrand *et al*. considèrent les problème des tâches strictement périodiques pour minimiser le nombre de processeurs. Ils montrent que si les périodes sont harmoniques, c'es-à-dire toutes divisibles entre elles, il existe une 2-approximation pour le problème de minimisation. Dans [56], les auteurs abordent le même problème avec des contraintes supplémentaires du domaine avionique. Supposant les périodes harmoniques, ils proposent une formulation en programmation linéaire en nombres entiers et une heuristique qui permettent de résoudre le problème sur des instances de taille industrielle.

2.2 Cadre théorique de la thèse

La recherche d'une solution pour un problème d'ordonnancement strictement périodique constitue le cœur de cette thèse. Le but est de trouver des algorithmes capables de produire une solution dans le contexte avionique en tenant compte des multiples contraintes du système. Bien que les périodes des tâches soient généralement harmoniques dans les systèmes industriels, ce travail propose une solution plus générale. Pour cela, différents algorithmes ont été développés sans faire d'hypothèse sur les périodes des tâches.

Le problème d'ordonnancement n'a pas simplement à satisfaire les contraintes de temps du système, mais doit aussi prendre en considération les contraintes non-fonctionnelles telles que la localisation et l'utilisation des ressources. De plus, nous verrons que le critère d'optimisation n'est pas classique et cherche à augmenter la marge d'évolution des budget de temps de chaque partition.

La présence de ces contraintes rend difficile à résoudre ce problème [68]. C'est pourquoi les différentes méthodes employées dans ce travail puisent dans plusieurs cadres théoriques, en particulier l'optimisation linéaire [59], la recherche locale et la théorie des jeux non-coopératifs [120].

3 Ordonnancement multi-processeur de tâches strictement périodiques

Le problème que nous abordons dans cette partie concerne l'ordonnancement des fonctions avioniques indépendantes, et sans contraintes de précédence, sur les unités de traitement. La particularité de ce problème est qu'il porte sur l'ordonnancement de tâches strictement périodiques, c'est-à-dire des tâches qui s'exécutent de façon non-préemptive à intervalles de temps constants sur un horizon infini. De plus, la fonction objectif maximise la durée minimale d'inactivité entre deux exécutions de tâches différentes tout en garantisant que les intervalles pendant lesquels elles s'exécutent ne se chevauchent pas. Ceci permet de garantir une marge d'évolution minimale des budgets de temps alloués aux tâches.

Nous proposons tout d'abord une formulation sous la forme d'un programme linéaire en nombres entiers intégrant de nombreuses contraintes temporelles et de ressource de ce problème [10]. Pour permettre le passage à l'échelle, nous proposons également une heuristique inspirée de la théorie des jeux

dans laquelle chaque tâche adapte à son tour son ordonnancement pour maximiser sa propre fonction d'utilité (qui est liée aux marges d'évolution des tâches) [12]. Nous montrons la convergence de cet algorithme vers un point d'équilibre dans lequel aucune tâche n'a intérêt à modifier unilatéralement sa stratégie et établissons qu'il existe au moins un équilibre qui est globalement optimal. Les résultats numériques obtenus montrent que cet algorithme est beaucoup plus rapide que la méthode exacte et fournit une bonne approximation. Pour améliorer encore la qualité des solutions, nous utilisons cette heuristique dans un algorithme multi-start qui permet d'obtenir des garanties probabilistes sur l'optimalité des équilibres atteints.

3.1 Ordonnancement uniprocesseur

Etant donné un ensemble $\Pi = \{1, \ldots, N\}$ de N tâches strictement périodiques, nous cherchons un ordonnancement non-préemptif de ces tâches permettant de garantir qu'il n'y a aucun recouvrement temporel dans leurs exécutions. Chaque tâche $i \in \Pi$ est caractérisée par sa période T_i et par son budget de temps b_i, qui représente la durée d'exécution maximale d'une instance (WCET). Posons $\mathcal{T}_i = \{0, 1, 2, \ldots, T_i - 1\}$. Nous notons $t_i \in \mathcal{T}_i$ la date de première exécution de la tâche i, aussi appelée offset. Nous définissons $\mathbf{t} = [t_1, \ldots, t_N]$ comme le vecteur des offsets, et $\mathcal{T} = \times_{i=1}^{N} \mathcal{T}_i$ comme l'ensemble des vecteurs d'offsets possibles.

Le problème que nous considérons consiste à affecter une date de première exécution à chaque tâche de telle manière qu'il n'y ait aucun recouvrement dans le temps de leurs exécutions. Les tâches étant strictement périodiques, pour un vecteur d'offsets $\mathbf{t}$ donné, l'instance k de la tâche i (ou $k$ème exécution) s'exécute dans l'intervalle

$$I_k(t_i) = [t_i + k\,T_i, t_i + k\,T_i + b_i]\,. \tag{1}$$

Les exécutions des tâches i et j ne se recouvrent pas si et seulement si $I_k(t_i) \cap I_l(t_j) = \emptyset$ pour tout $k, l \in \mathbb{Z}$, d'où la définition suivante.

Définition 1 *Un vecteur d'offsets $\mathbf{t} \in \mathcal{T}$ est dit admissible si et seulement si*

$$I_k(t_i) \cap I_l(t_j) = \emptyset,\ \forall k, l \in \mathbb{Z}, \tag{2}$$

pour toutes tâches $i, j \in \Pi$, $i \neq j$.

Puisque nous voulons garantir qu'il n'y a pas de recouvrement entre deux exécutions, il est naturel de s'intéresser à la distance minimale entre deux débuts d'exécution. En effet, il n'y aura pas de recouvrement si cette distance est supérieure à la durée de la tâche qui s'exécute en premier. Considérons l'instance k de la tâche i et l'instance l de la tâche j. En utilisant le théorème de Bachet-Bezou, on montre aisément que si la tâche i s'exécute en premier, i.e. $t_i + kT_i \leq t_j + lT_j$, alors la durée minimale entre les deux débuts d'exécution est $(t_j - t_i)\%g_{i,j}$, où $g_{i,j}$ est le *plus grand commun diviseur* de T_i et T_j et où le symbole $\%$ est utilisé comme notation abrégée de l'opérateur modulo, i.e. $a\%b$ doit être lu comme $a \mod b$. De même, si la tâche j s'exécute en premier, i.e. $t_i + kT_i \geq t_j + lT_j$, alors la durée minimale entre les deux débuts d'exécution est $(t_i - t_j)\%g_{i,j}$. On en déduit ainsi une condition nécessaire d'admissibilité d'un vecteur d'offsets, qui est présentée formellement dans le Lemme 1.

Lemme 1

$$\min_{k,l\in\mathbb{Z}} |(t_j + lT_j) - (t_i + kT_i)| = \min\left[(t_j - t_i)\%g_{i,j}, (t_i - t_j)\%g_{i,j}\right] \tag{3}$$

Nous déduisons de ce résultat une condition nécessaire et suffisante pour l'ordonnançabilité de deux tâches qui est décrite dans le théorème suivant.

Théorème 2 *Les exécutions de deux tâches i et j ne se recouvrent pas si et seulement si*

$$b_i \leq (t_j - t_i)\%g_{i,j} \quad \textit{et} \quad b_j \leq (t_i - t_j)\%g_{i,j}, \tag{4}$$

ou, de façon équivalente, si et seulement si

$$b_i \leq (t_j - t_i)\%g_{i,j} \leq g_{i,j} - b_j. \tag{5}$$

Le terme $\frac{(t_j - t_i)\%g_{i,j}}{b_i}$ représente le facteur maximal par lequel la durée d'exécution b_i de la tâche i peut être multipliée sans interférer avec les exécutions de la tâche j. On peut voir ce terme comme la marge d'évolution sur la durée b_i par rapport à la tâche j. Si ce terme est supérieur ou égal à 1 pour toutes tâches $i, j \neq i$, cela signifie que le vecteur d'offsets $\mathbf{t}$ est admissible. Une condition suffisante pour que l'ordonnancement $\mathbf{t}$ soit admissible est donc que $\min_{i\neq j} \frac{(t_j - t_i)\%g_{i,j}}{b_i} \geq 1$. Pour déterminer un ordonnancement admissible, il apparaît donc naturel de chercher à maximiser ce minimum. Ainsi, en introduisant

$$d_{ij}(\mathbf{t}) = \min\left(\frac{(t_j - t_i)\%g_{i,j}}{b_i}, \frac{(t_i - t_j)\%g_{i,j}}{b_j}\right), \tag{6}$$

le problème d'ordonnancement peut être formulé de la façon suivante :

$$\begin{aligned} &\text{maximiser } \min_{i,j\neq i} d_{ij}(\mathbf{t}), \\ &\text{sous la contrainte } \mathbf{t} \in \mathcal{T}. \end{aligned} \tag{OPT}$$

Remarquons de plus que si la valeur optimale de ce problème est strictement supérieure à 1, cela permet de garantir une marge d'évolution minimale sur les budgets de temps alloués aux tâches qui peut être utile à l'avenir si les traitements effectués par ces tâches évoluent. Il est possible d'écrire ce problème d'ordonnancement uniprocesseur sous la forme d'un programme linéaire en nombres entiers (*cf.* [10] pour plus de détails):

$$\begin{aligned} &\text{maximiser} && \alpha \\ &\text{sous les contraintes} \\ &&& \mathbf{t} \in \mathcal{T}, \\ &&& (t_j - t_i) - q_{j,i}\, g_{i,j} \geq \alpha\, b_i,\ \forall (i,j) \in \Pi^2, \\ &&& (t_j - t_i) - q_{j,i}\, g_{i,j} \leq g_{i,j} - \alpha\, b_j,\ \forall (i,j) \in \Pi^2, \\ &&& t_i \in [0, T_i),\ \forall i \in \Pi, \end{aligned}$$

où la variable $q_{j,i}$ représente en fait le quotient entier $\left\lfloor \frac{t_j - t_i}{g_{i,j}} \right\rfloor$. Bien que ce programme linéaire en nombres entiers puisse être résolu numériquement, il n'est possible de résoudre que des exemples de tailles modestes en un temps raisonnable – rappelons en effet que le problème est NP complet au sens fort. Dans les paragraphes suivants, nous présentons un algorithme d'approximation relativement rapide pour générer des vecteurs d'offsets qui sont optimaux sous certaines conditions. Comme le montrent les résultats numériques, le principal avantage de cet algorithme est qu'il est beaucoup plus rapide que les méthodes exactes basées sur la formulation linéaire en nombres entiers proposées ci-dessus tout en fournissant en même temps des solutions de bonne qualité.

3.2 Algorithme de la meilleure réponse dans le cas uniprocesseur

L'algorithme de la meilleure réponse que nous proposons pour résoudre le problème d'optimisation (OPT) est inspiré d'un algorithme du même nom en théorie des jeux [63]. Dans cet algorithme, nous identifions les tâches à des joueurs jouant un jeu séquentiel. Dans ce jeu, chaque tâche adapte à son tour sa stratégie (i.e. son offset) en fonction des offsets des autres tâches. Le jeu se poursuit jusqu'à ce que le vecteur d'offsets converge vers un point d'équilibre, appelé équilibre de Nash, dans lequel plus aucune tâche n'a intérêt à changer sa stratégie. Notons t_j^n la stratégie de la tâche j au début de l'itération n et supposons qu'à cette itération c'est au tour de la tâche i de jouer. Cette tâche va calculer son offset de manière à maximiser sa distance relative par rapport aux autres tâches en résolvant le problème suivant :

$$\begin{aligned} &\text{maximiser} && \min_{j \neq i} d_{i,j}(x, \mathbf{t}_{-i}^n) \\ &\text{sous la contrainte} && x \in \mathcal{T}_i, \end{aligned} \tag{SCHD-i}$$

où, suivant la notation habituelle en théorie des jeux, $\mathbf{t}_{-i} = [t_1, t_2, \ldots, t_{i-1}, t_{i+1}, \ldots, t_N]$ est le vecteur d'offsets de tous les joueurs autres que i. Le joueur i va alors fixer t_i^{n+1} à la valeur x donnant la solution optimale du problème SCHD-i, c'est-à-dire sa meilleure réponse. Si cette dernière n'est pas unique, nous supposons que la tâche va retenir le plus petit offset parmi ceux donnant la meilleure réponse. On notera que le joueur i résoud en fait le même problème que le problème global (OPT), sauf qu'il ne prend en compte que les termes affectés par son offset.

Dans la suite, définissons

$$\alpha_i^n = \min_{j \neq i} d_{i,j}(\mathbf{t}^n) \tag{7}$$

$$\mathcal{S}_i(\mathbf{t}_{-i}^n) = \operatorname*{argmax}_{x \in \mathcal{T}_i} \min_{j \neq i} d_{i,j}(x, \mathbf{t}_{-i}^n), \tag{8}$$

où α_i^n est l'utilité du joueur i après l'itération n, c'est-à-dire la marge d'évolution relatif à ce joueur, et $\mathcal{S}_i(\mathbf{t}_{-i})$ est l'ensemble des meilleures réponses de ce joueur. Nous supposons de plus que si le joueur i ne peut améliorer son utilité α_i^n, il ne change pas sa stratégie, i.e. $t_i^{n+1} = t_i^n$. Cette hypothèse, bien que non restrictive, est utile pour démontrer la convergence de l'algorithme. Le pseudo-code de l'algorithme de la meilleure réponse est décrit dans l'Algorithme 1. A l'étape 4 de l'algorithme, $n\%N + 1$ donne l'index du joueur dont c'est le tour d'adapter sa stratégie à l'itération n.

Algorithm 1 Meilleure Réponse Uniprocesseur

Require: $\mathbf{t}^0$
$n \leftarrow 0$
while $\mathbf{t}^n \neq \mathbf{t}^{n-N}$ **do**
  for $i = 1$ to N **do**
    if $i = n\%N + 1$ and $\max_x \min_{j \neq i} d_{i,j}(x, \mathbf{t}^n_{-i}) > \alpha^n_i$ **then**
      $t^{n+1}_i \leftarrow \min \operatorname{argmax}$ (SCHD-i)
    else
      $t^{n+1}_i \leftarrow t^n_i$
    end if
  end for
  $n \leftarrow n + 1$
end while
Return $\mathbf{t}^n$

Deux propriétés importantes de cet algorithme sont formulées dans les théorèmes suivants.

Théorème 3 *L'algorithme de la meilleure réponse converge vers un point d'équilibre.*

Théorème 4 *Il existe au moins un point d'équilibre qui est aussi une solution optimale du problème.*

En conséquence des résultats précédents, nous déduisons que si le point de départ est choisi de manière appropriée, l'algorithme de la meilleure réponse convergera vers une solution globalement optimale. La proposition suivante fournit de plus une borne supérieure sur le nombre d'itérations.

Proposition 5 *Soit* $\alpha_{max} = \max_i \min_{j \neq i} \frac{g_{i,j}}{b_i + b_j}$ *et* $\Delta = \min_{j,k} \frac{1}{ppcm(b_j, b_k)}$. *L'algorithme de la meilleure réponse converge en au plus* $\binom{N+K}{K} N$ *itérations, où* $K = \lceil \alpha_{max} \Delta^{-1} \rceil$.

Cette borne supérieure sur le nombre d'itérations est une estimation pessimiste qui est exponentielle en le nombre de tâches. En pratique, pour toutes les expérimentations effectuées, l'algorithme a toujours convergé en quelques dizaines d'itérations au pire.

Un point critique de cet algorithme concerne le calcul de la meilleure réponse d'un joueur. La meilleure réponse de la tâche i peut bien sûr être calculée en effectuant une recherche linéaire, ce qui nécessite $O(T_i)$ opérations. On peut en fait déterminer la meilleure réponse de façon beaucoup plus efficace en faisant l'hypothèse suivante.

Conjecture 6 *Les offsets peuvent prendre des valeurs réelles, i.e.* $\mathcal{T}_i = [0, T_i)$.

Sous cette hypothèse, introduisons pour la tâche i l'ensemble

$$\mathcal{I}_i(\mathbf{t}_{-i}) = \bigcup_{(j,k) \in (\Pi \setminus \{i\})^2} \left\{ x : \frac{(x - t_j)\%g_{i,j}}{b_j} = \frac{(t_k - x)\%g_{i,k}}{b_k} \right\}$$

des points d'intersection où deux tâches interférent. On a alors le résultat suivant.

Théorème 7 $\mathcal{S}_i(\mathbf{t}_{-i}) \subset \mathcal{I}_i(\mathbf{t}_{-i}) \subset \mathcal{T}_i$.

Ainsi, la solution de (SCHD-i) peut être obtenue en restreignant la recherche aux points de l'ensemble $\mathcal{I}_i(\mathbf{t}_{-i})$. Dans la mesure où le problème original n'est défini que pour des offsets entiers, l'heuristique décrite dans l'Algorithme 1 va chercher la meilleure réponse de la tâche i dans les entiers directement inférieurs et supérieurs aux points de l'ensemble $\mathcal{I}_i(\mathbf{t}_{-i})$, plutôt que d'examiner tous les T_i points possibles.

Il est de plus possible de générer très efficacement les points de $\mathcal{I}_i(\mathbf{t}_{-i})$ grâce à la méthode décrite dans l'Algorithme 2. Le lecteur intéressé se reportera à [12] pour plus de détails sur cet algorithme.

Algorithm 2 Calcul des points d'intersection

REQUIRE : $\mathbf{t}_{-i}$
$\mathcal{I}_i(\mathbf{t}_{-i}) = \emptyset$
for all $j \neq i$ **do**
  for all $k \neq i$ **do**
    for $l = \left\lfloor \frac{t_j - t_k - \min\left(\frac{g_{i,j}}{b_i}, \frac{g_{i,k}}{b_k}\right)(b_i + b_k)}{g_{i,j,k}} \right\rfloor + 1$ to $\left\lfloor \frac{t_j - t_k}{g_{i,j,k}} \right\rfloor$ **do**
      $\tau(l) = (t_j - l\hat{m}g_{i,j} - \frac{t_j - t_k - lg_{i,j,k}}{1 + \frac{b_k}{b_i}}) \% c_{i,j,k}$
      for $r = 0$ to $\left\lfloor \frac{T_i - \tau(l)}{c_{i,j,k}} \right\rfloor$ **do**
        $\mathcal{I}_i(\mathbf{t}_{-i}) \leftarrow \mathcal{I}_i(\mathbf{t}_{-i}) \cup (\tau(l) + rc_{i,j,k})$
      end for
    end for
  end for
end for

Signalons enfin que l'on peut obtenir la borne suivante sur la cardinalité de $\mathcal{I}_i(\mathbf{t}_{-i})$

$$|\mathcal{I}_i(\mathbf{t}_{-i})| \leq \sum_{j \neq i} \sum_{k \neq i} \frac{\min\left(\frac{g_{i,j}}{b_i}, \frac{g_{i,k}}{b_k}\right)(b_i + b_k)T_i}{g_{i,j}g_{i,k}}$$

d'où l'on peut déduire que si les tâches sont au moins ordonnançables deux à deux, i.e. $(b_i + b_j) \leq g_{i,j}, \forall i, j \neq i$, alors une condition suffisante pour que l'algorithme ci-dessus soit plus efficace qu'une recherche linéaire est que $b_i > N^2$.

3.3 Ordonnancement multiprocesseur

Nous montrons dans ce paragraphe comment l'algorithme de la meilleure réponse pour l'ordonnancement uniprocesseur s'étend naturellement au cas multiprocesseur. Soit $\mathcal{P} = \{1, \ldots, P\}$ un ensemble de P processeurs, le processeur k étant caractérisé par sa capacité mémoire M_k et par le nombre maximal H_k de tâches qu'il peut accueillir. Un ordonnancement n'est plus seulement décrit par la donnée du vecteur d'offsets $\mathbf{t}$, mais également par l'affectation d'un processeur à chacune des tâches. Cette affectation peut être représentée par un vecteur de variables binaires $\mathbf{a} = (a_{i,k})_{i \in \Pi, k \in \mathcal{P}}$ telles que $a_{i,k} = 1$ si la tâche i

est affectée au processeur k, et $a_{i,k} = 0$ sinon. On peut alors formuler le problème d'ordonnancement multiprocesseur comme un programme linéaire en nombres entiers de la façon suivante (certaines contraintes spécifiques à l'application avionique sont omises ici, cf [10]):

$$\text{maximiser}_{\mathbf{a},\mathbf{t}}\,\alpha \tag{9}$$

$s.t.$

$$\sum_{p_k \in \mathcal{P}} a_{i,k} = 1 \quad ,\forall i \in \Pi, \tag{10}$$

$$\sum_{i \in \Pi} a_{i,k}\, m_i \leq M_k \quad ,\forall k \in \mathcal{P}, \tag{11}$$

$$\sum_{i \in \Pi} a_{i,k} \leq H_k \quad ,\forall k \in \mathcal{P}, \tag{12}$$

$$(t_j - t_i) - q_{j,i}\, g_{i,j} \geq \alpha\, b_i - (2 - a_{i,k} - a_{j,k})\, Z \quad ,\forall k, \forall (i,j), \tag{13}$$

$$(t_j - t_i) - q_{j,i}\, g_{i,j} \leq g_{i,j} - \alpha\, b_j + (2 - a_{i,k} - a_{j,k})\, Z \quad ,\forall k, \forall (i,j), \tag{14}$$

$$a_{i,k} \in \{0,1\} \quad ,\forall k, \forall i, \tag{15}$$

$$t_i \in [0, T_i) \quad ,\forall i \in \Pi, \tag{16}$$

$$q_{j,i} \in \mathbb{Z} \quad ,\forall (i,j), \tag{17}$$

Comme dans le cas uniprocesseur, la fonction objectif (9) représente le minimum des marges d'évolution des tâches. Les contraintes (10) imposent le choix d'un seul processeur par tâche, tandis que les contraintes (11) et (12) sont celles associées aux capacités mémoire et en nombre de tâches des processeurs. Les contraintes (13) et (14) expriment la condition d'ordonnançabilité (4) pour les couples de tâches affectées au même processeur (Z est une grande constante qui permet de garantir que ces contraintes ne sont actives que si $a_{i,k} = a_{j,k} = 1$). Les contraintes (15), (16) et (17) décrivent le domaine des variables. Cette formulation linéaire n'est évidemment utilisable que pour des problèmes de tailles modestes.

3.4 Algorithme de la meilleure réponse dans le cas multiprocesseur

L'algorithme de la meilleure réponse pour l'ordonnancement uniprocesseur s'étend au cas multiprocesseur de la façon suivante. A son tour, une tâche i va calculer sa meilleure réponse sur chacun des processeurs, les uns après les autres. Puis, elle va sélectionner le processeur lui permettant de maximiser son utilité qui est définie de la façon suivante

$$\alpha_i^n = \min_{\{j : j \neq i, a_{i,k}^n = a_{j,k}^n \forall k\}} d_{i,j}(\mathbf{t}^n), \tag{18}$$

et qui représente la marge d'évolution de la tâche i par rapport aux tâches ordonnancées sur le même processeur qu'elle et pour des vecteurs d'offsets $\mathbf{t}^n$ et d'allocation $\mathbf{a}^n$ donnés. Exactement comme dans

le cas uniprocesseur, nous supposons que la tâche i ne change pas sa stratégie (processeur et offset) si elle ne peut pas améliorer son utilité. Le pseudocode pour l'heuristique de meilleure réponse multiprocesseur est décrit dans l'Algorithme 3, où $\mathbf{a}_i^n = \left(a_{i,k}^n\right)_{k\in\mathcal{P}}$ est le vecteur des variables d'affectation de la tâche i à l'itération n.

Algorithm 3 Algorithme de la meilleure réponse multiprocesseur

REQUIRE $\mathbf{t}^0, \mathbf{a}^0$
$n \leftarrow 0$
while $\mathbf{t}^n \neq \mathbf{t}^{n-N}$ and $\mathbf{a}^n \neq \mathbf{a}^{n-N}$. **do**
  for $i = 1$ to N **do**
    if i = n % N + 1 **then**
      $\mathbf{a}_i^{n+1} \leftarrow \mathbf{0}$
      for $k = 1$ to M **do**
        $z \leftarrow \max_x \min_{\{j:j\neq i, a_{j,k}=1\}} d_{i,j}(x, \mathbf{t}_{-i}^n)$
        if $z > \alpha_i^n$ **then**
          $\alpha_i^n \leftarrow z$
          $c \leftarrow k$
          $t_i^{n+1} \leftarrow \min \operatorname{argmax} \min_{\{j:j\neq i, a_{j,k}=1\}} d_{i,j}(x, \mathbf{t}_{-i}^n)$
        end if
      end for
      $a_{i,c}^{n+1} = 1$
    else
      $t_i^{n+1} \leftarrow t_i^n$
      $\mathbf{a}_i^{n+1} \leftarrow \mathbf{a}_i^n$
    end if
  end for
  $n \leftarrow n + 1$
end while
RETURN $\mathbf{t}^n$

Comme dans le cas uniprocesseur, on peut montrer que cet algorithme converge vers un point d'équilibre et qu'il existe au moins un point d'équilibre qui est globalement optimal.

3.5 Un exemple de résultat

Pour illustrer les résultats obtenus, considérons le problème d'ordonnancement uniprocesseur avec 20 tâches non harmoniques dont les caractéristiques temporelles sont indiquées dans le tableau 1. Pour cet exemple, l'algorithme de la meilleure réponse fournit une marge d'évolution minimale égale à $\alpha = 1.41$ en 2.83 secondes, tandis que la résolution avec CPLEX [81] de la formulation linéaire en nombres entiers ne permet d'obtenir qu'une marge de $\alpha = 1.11$ au bout d'une heure de calcul. De manière générale, l'ensemble des expérimentations effectuées ont montré que l'algorithme de la meilleure réponse permet d'obtenir des solutions de bonne qualité, et généralement admissibles, en des temps calculs très inférieurs à une résolution exacte basée sur la programmation linéaire en nombres entiers.

Table 1: Exemple uniprocesseur avec 20 tâches non harmoniques (hyperpériode de 756000).

Tâche	1	2	3	4	5	6
Budget de temps	10	30	30	10	10	10
Période	1200	1200	3600	1200	1200	1500

7	8	9	10	11	12	13
10	10	30	10	10	45	40
4200	1000	2000	4000	1200	2400	2000

14	15	16	17	18	19	20
40	60	80	30	10	60	40
4000	3000	3000	2700	200	1800	1800

3.6 Algorithme multi-start

La qualité des solutions obtenues avec l'algorithme de la meilleure réponse peut encore être améliorée en utilisant une méthode multi-start [115]. En effet, l'espace des solutions du problème d'ordonnancement est divisé en région d'attraction, l'algorithme de la meilleure réponse conduisant vers le même équilibre pour tous les points initiaux appartenant à la même région. Une méthode multi-start va permettre de générer aléatoirement des points de départ de l'heuristique appartenant à des régions d'attraction différentes. Des régles d'arrêt Bayesiennes [30] sont utilisées pour arrêter l'exploration aléatoire de l'espace des solutions lorsque des garanties probabilistes suffisantes sont obtenues sur l'optimalité de la meilleure solution trouvée.

En pratique, le nombre de points d'équilibre k (associé à la v.a. K) ainsi que le volume relatif ϕ_i de chaque région $i = 1, \ldots, k$ (associé à la v.a. Φ_i) sont inconnus. Les points de départ générés par la méthode du multi-start étant uniformément distribués sur l'espace des solutions, le $i^{\text{ème}}$ point d'équilibre a une probabilité ϕ_i d'être découvert. On suppose que la distribution *a priori* du nombre d'équilibres est uniforme sur $[1, \infty)$ et que les volumes relatifs des régions d'attraction ont également une distribution uniforme. Cela conduit à une densité de probabilités *a priori* donnée par $p(k, \phi_1, \ldots, \phi_k) \propto (k-1)!$. Etant donné la nombre s d'échantillons générés (points de départ de l'heuristique), le nombre w d'équilibres observés et le nombre s_i d'occurences du $i^{\text{ème}}$ point d'équilibre, on peut déterminer les quantités suivantes [30] :

- Espérance *a posteriori* du nombre d'équilibres : $E(K|\{s_1, \ldots, s_w\}) = \frac{w(s-1)}{s-w-2} \quad (s \geq w+3)$.
- Espérance *a posteriori* du volume relatif de la $j^{\text{ème}}$ région :

$$E(\Phi_{s_j}|\{s_1, \ldots, s_w\}) = \frac{(s_j+1)(s+w)}{s(s-1)} \quad (s \geq w+2).$$

- Espérance *a posteriori* du volume relatif total des régions observées (v.a. Ω):

$$E(\Omega|\{s_1,\ldots,s_w\}) = \frac{(s-w-1)(s+w)}{s(s-1)} \quad (s \geq w+2).$$

Deux régles d'arrêt pertinentes consistent soit à terminer la recherche quand le nombre d'équilibres observés est suffisamment proche du nombre d'équilibres estimé, soit à la terminer quand le volume relatif total des régions observées dépasse un certain pourcentage. Une autre approche, que nous avons utilisée dans nos travaux, consiste à utiliser le concept de perte de terminaison qui associe un cot à l'écart entre une quantité inconnue estimée et sa vraie valeur. Ainsi, la perte de terminaison peut être prise proportionnelle au volume relatif total des régions non observées comme suggéré dans [30].

Dans le cas uniprocesseur, la méthode multi-start s'exécute en quelques minutes. Elle nous a permis de réduire l'écart relatif moyen à l'optimum de 7.8% avec l'heuristique à 0.25% pour les exemples harmoniques, et de 4.2% à 0% pour les exemples non harmoniques. Dans le cas multiprocesseur, nous avons obtenu un temps de calcul moyen de 16 minutes. L'écart à l'optimum est passé de 14.17% à 3.84% pour les instances harmoniques, et de 18.5% à 9.6% pour les instances non harmoniques. Nous avons observé que dans le cas multiprocesseur les points d'équilibre optimaux peuvent se trouver dans de petites régions d'attraction, ce qui peut nécessiter un critère d'arrêt plus strict que celui que nous avons utilisé.

4 Routage des Liens Virtuels

Après l'ordonnancement des partitions, il est nécessaire de router les flux de messages échangés entre les partitions dans le réseau AFDX. Ces messages sont groupés dans un ou plusieurs liens virtuels, ou VL (pour Virtual Link). Le routage d'un VL correspond en fait à un arbre multicast (*cf.* Figure 2) entre le module de traitement où est située la partition émettrice et les modules où sont situés les partitions réceptrices. Chaque VL est caractérisé par deux quantités (*cf.* Figure 3) : le BAG (Bandwidth Allocation Gap), qui définit l'intervalle de temps minimal séparant l'émission de deux trames consécutives, et la taille maximale des trames Ethernet envoyées, notée MFS (pour Maximum Frame Size). La bande-passante d'un VL, en tenant compte d'une entête de 67 octets, est donc

$$bw = \frac{(67+MFS)}{BAG}, \tag{19}$$

4.1 Description du problème

Une fois les partitions ordonnancées sur les modules, le concepteur doit définir les VL nécessaires ainsi que toutes leurs caractéristiques, puis spécifier leur routage dans le réseau de commutation AFDX en remplissant des tables de configuration. Ce processus de routage doit évidemment prendre en compte les ressources réseaux disponibles.

Le problème revient à déterminer un arbre multicast par VL. Dans le cas où il y a un seul VL, ce problème est équivalent à celui de l'arbre de Steiner [69, 79] qui est bien connu pour être NP difficile.

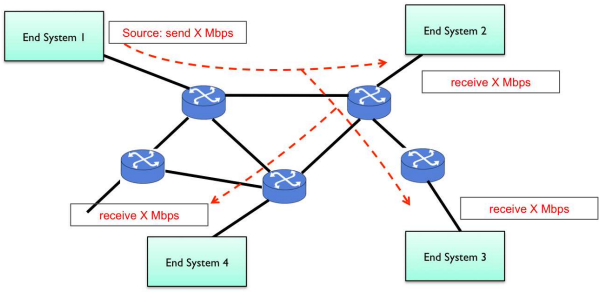


Figure 2: Lien virtuel issu d'un module et acheminant des données vers 3 autres.

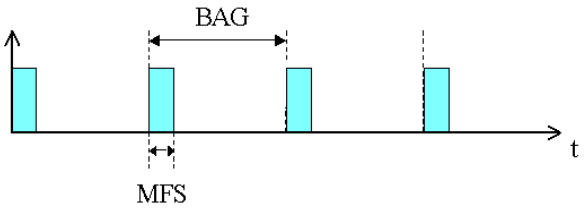


Figure 3: Un VL envoie des trames de taille maximale MFS octets toutes les BAG ms.

Un autre cas particulier est celui où chaque VL a une seule destination, ce qui conduit à un problème de monoroutage qui est également NP difficile [127]. Dans le cas général, le problème a été relativement peu traité, les travaux les plus proches concernant la détermination d'un arbre de routage multicast dans un réseau déjà chargé [136].

Nous avons proposé deux méthodes permettant l'automatisation du routage des VL. Ces méthodes sont décrites ci-dessous.

4.2 Définition formelle

Nous représentons le réseau AFDX par un graphe $(\mathcal{N}, \mathcal{E})$ ayant un ensemble $\mathcal{N} = \{1, \ldots, N\}$ de noeuds (modules de traitement inclus), et un ensemble $\mathcal{E}$ de liens. On note c_e la capacité du lien $e \in \mathcal{E}$. Ce réseau doit acheminer un ensemble $\mathcal{V} = \{1, \ldots, V\}$ de VL, chaque VL v étant caractérisé par

- un noeud source (module de traitement) $src(v) \in \mathcal{N}$,
- un ensemble de noeuds destination (modules de traitement) $dest(v) \subseteq \mathcal{N} \setminus \{src(v)\}$,
- et une bande-passante b_v, définie conformément à l'équation (19).

L'objectif est de minimiser le taux d'utilisation maximal des liens du réseau, c'est-à-dire,

$$\text{Minimiser } \left[\rho = \max_{e \in \mathcal{E}} \left(\frac{y_e}{c_e}\right)\right],$$

où y_e représente le trafic total sur le lien e. Ce critère d'optimisation a été choisi en accord avec nos partenaires du projet SATRIMMAP suite à des études ayant montré que le délai réseau n'intervient que pour une faible part dans le délai de transmission au pire d'un message, le choix du BAG et les périodes des partitions réceptrices étant les paramètres dominants [100]. Ce critère a de plus l'avantage de garantir une certaine marge d'évolution sur les ressources du réseau, qui peut être nécessaire si de nouveaux VL doivent être déployés.

4.3 Formulation noeud-lien

Notons $\Gamma^+(n)$ et $\Gamma^-(n)$ l'ensemble des liens entrants et sortants du noeud n respectivement, comme illustré sur la Figure 4.

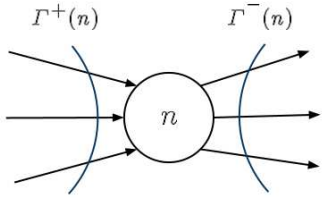


Figure 4: Liens entrants et sortants du noeud n.

En adoptant une formulation noeud-lien, le problème peut être écrit comme un programme linéaire en nombres entiers :

$$\min \rho$$

$$s.t.$$

$$\sum_{e\in\Gamma^+(n)} x_v^e - \sum_{e\in\Gamma^-(n)} x_v^e = h_{n,v} \qquad ,\forall n, v, \tag{20}$$

$$y_v^e\, M \geq x_v^e \qquad ,\forall v, e, \tag{21}$$

$$y_v^e \leq x_v^e \qquad ,\forall v, e, \tag{22}$$

$$\sum_{e\in\Gamma^+(n)} y_v^e \leq 1 \qquad ,\forall n, v, \tag{23}$$

$$y_e = \sum_{v\in\mathcal{V}} y_v^e\, b_v \qquad ,\forall e, \tag{24}$$

$$y_e \leq c_e\, \rho \qquad ,\forall e, \tag{25}$$

$$y_v^e \in \{0, 1\} \qquad ,\forall v, e, \tag{26}$$

$$x_v^e \in \{0, \ldots, K_v\} \qquad ,\forall v, e, \tag{27}$$

Dans cette formulation, la variable entière x_v^e correspond au nombre de destinations du VL v qui seront jointes en passant par le lien e. Les constantes $h_{n,v}$ intervenant dans les contraintes de conservation (20) sont données par

$$h_{n,v} = \begin{cases} -K_v & \text{si } i = src(v), \\ 1 & \text{si } i \in dst(v), \\ 0 & \text{sinon.} \end{cases}$$

où K_v est le nombre de destinations du VL v. La variable binaire y_v^e est calculée à partir de x_v^e grâce aux contraintes (21) et (22). Elle indique si le lien e est utilisé par le VL v. Les contraintes (23) imposent un routage en arbre pour les VL. La variable y_e dont la valeur est définie par la contrainte (24) correspond au trafic sur le lien e. Elle est utilisée pour calculer la charge maximale ρ des liens dans la contrainte (25). Cette formulation permet de résoudre de manière exacte des problèmes de tailles assez conséquentes dans des temps raisonnables. Elle n'apporte toutefois aucune garantie sur la longueur des arbres multicast générés.

4.4 Heuristique à deux niveaux

On peut reformuler le problème en utilisant une formulation lien-chemin de la façon suivante :

$$\min \ \rho \qquad \text{(OPT-VL)}$$

$$s.t.$$

$$\sum_{T \in \mathcal{T}^v} x_T = 1 \qquad ,\forall v \in \mathcal{V}, \qquad (28)$$

$$y_e = \sum_{v \in \mathcal{V}} \sum_{T \in \mathcal{T}^v} \delta_e^T \, x_T \, b_v \qquad ,\forall e \in \mathcal{E}, \qquad (29)$$

$$y_e \leq c_e \, \rho \qquad ,\forall e \in \mathcal{E}, \qquad (30)$$

$$x_T \in \{0,1\} \qquad ,\forall T \in \mathcal{T}^v, \forall v \in \mathcal{V}, \qquad (31)$$

où $\mathcal{T}^v$ est l'ensemble des arbres de routage possibles pour le VL v. La variable de décision binaire x_T indique si l'arbre T est choisi pour router le VL v. La constante booléenne δ_e^T indique si l'arbre T passe par le lien e et est utilisée pour calculer la charge y_e de ce lien.

L'inconvénient de cette formulation est qu'il n'est en général pas possible d'énumérer tous les arbres de routage de $\mathcal{T}^v$. Notre idée est alors de décomposer la résolution en deux phases. Dans une première phase, nous déterminons pour chaque VL un ensemble d'arbres multicast candidats au routage de ce VL. Ces arbres candidats doivent être choisis de manière à ne pas compromettre l'optimalité de la solution obtenue et donc en tenant compte de la charge des liens. D'autre part, on souhaite retenir des arbres candidats dont la longueur n'est pas trop grande pour minimiser le délai de transmission des messages. Pour cela on suit une approche de type "mille-feuille" en dupliquant chaque VL v en R copies ayant chacune une demande en bande-passante égale à b_v/R. Puis à chaque itération on route un VL, la copie $i = 1, \ldots, R$ du VL v étant routée à l'itération $k = (i-1)V + v$ suivant l'arbre $T \in \mathcal{T}^v$ minimisant

$$\sum_{e \in \mathcal{E}} D_e(y_e + \delta_e^T \, \frac{b_v}{R}) - \sum_{e \in \mathcal{E}} D_e(y_e) \qquad (32)$$

où $D_e(y_e)$ représente le cot du lien e en fonction du trafic sur ce lien[1]. Dans la mesure où elle est additive, la fonction cot (32) permet de prendre en compte la longueur des arbres choisis, et en même temps elle permet de prendre en compte la charge des liens à travers les cots $D_e(y_e)$. Déterminer l'arbre T minimisant la fonction objectif (32) revient en fait à résoudre un problème de Steiner dans lequel le lien e à un cot $\Delta_e = D_e(y_e + b_v/R) - D_e(y_e)$. Pour cela nous utilisons une heuristique inspirée de [94].

Après avoir routé chaque copie de chaque VL, on dispose d'au plus R arbres de routage candidats par VL. La seconde phase consiste alors à résoudre le problème (OPT-VL) à l'aide d'un solveur linéaire en nombres entiers en restreignant $\mathcal{T}^v$ aux arbres de routage séléctionnés dans la première phase pour chaque VL v.

Dans toutes les expérimentations effectuées, nous avons observé que cette heuristique a toujours fourni une solution optimale, tout en permettant une réduction significative de la longueur des arbres. Toutefois, l'heuristique en 2 phases a des temps de calcul supérieurs à l'approche exacte basée sur la formulation noeud-lien, même si ceux-ci restent très raisonnables (environ 20 secondes pour 1000 VL sur la topologie considérée). De plus, sur un benchmark constitué de 4894 VL, nous avons observé que le nombre importants de variables et de contraintes de la formulation noeud-lien n'a pas permis de résoudre le problème par manque de mémoire de la machine utilisée, alors que sur la même machine l'heuristique en 2 phases a fourni une solution en environ une demi-heure. Cette dernière approche a de plus l'avantage de permettre l'intégration d'autres contraintes dans la phase de sélection des chemins, par exemple pour interdire certains liens à certains VL.

5 Conclusion

Cette thèse aborde deux problèmes fondamentaux pour l'allocation de ressources dans les systèmes avioniques. L'un concerne l'ordonnancement et le placement des applications avioniques sous contraintes de ressource et l'autre aborde le routage des messages à travers le réseau avionique.

Dans un premier temps, le travail sur l'ordonnancement a été formulé sous la forme d'un problème linéaire en nombres entiers. Le critère d'optimisation est original et cherche à maximiser la marge d'évolution temporelle pour les différentes partitions. Ainsi, une solution optimale est caractérisée par un facteur multiplicatif sur les budgets de temps des partitions tel que l'ordonnancement soit faisable pour cette valeur. Une valeur plus grande que 1 implique la faisabilité du problème d'ordonnancement, mais donne aussi une estimation de la marge d'évolution temporelle du système. La formulation exacte, bien qu'assurant l'optimalité, est montrée inefficace pour des grands problèmes.

En plus de cette formulation linéaire, une heuristique basée sur la théorie des jeux a été proposée. Suivant le principe de l'algorithme de la meilleure réponse, elle a été repensée spécifiquement pour le problème d'ordonnancement strictement périodique. Le principe consiste à ce que chaque partition essaie à tour de rôle de faire évoluer son offset et son allocation afin d'accrotre la marge d'évolution globale. Les résultats expérimentaux obtenus montrent de bonnes performances en temps et en qualité. Les comparaisons avec la méthode exacte sur de petites instances ont montré une faible erreur par rap-

[1] Nous avons utilisé $D_e(y_e) = \frac{1}{c_e - y_e}$, i.e. le délai moyen d'une file M/M/1 [89].

port à l'optimum. L'heuristique a aussi permis de résoudre des problèmes de grande taille, là où la solution exacte était incapable d'apporter une réponse.

La qualité de l'algorithme de la meilleure réponse est très sensible à son initialisation. Pour pallier ce problème, une méthode de type multi-start avec un critère d'arrêt bayésien a permis de réduire grandement cette sensibilité et de réduire fortement l'erreur relative des solutions produites.

La dernière partie de cette thèse est consacrée au problème du routage des messages échangés entre les fonctions avioniques. Une formulation exacte du problème sous la forme d'un programme linéaire en nombres entiers de type noeuds-liens a été proposée, ainsi qu'une heuristique à deux niveaux qui permet de faire un compromis entre une distribution équitable de la charge dans le réseau et la minimisation du délai de communication. Les résultats expérimentaux montrent que l'heuristique permet d'obtenir des solutions très proches de la méthode exacte tout en améliorant significativement les délais de communication.

5.1 Perspectives

Le problème d'allocation traité dans cette thèse n'aborde pas celui de la reconfiguration. Cela sera envisageable, par exemple, en proposant des ordonnancements qui garantissent la robustesse du système en cas de panne d'un module tout en assurant la réallocation des partitions. Ce travail devrait passer par la définition des scénarii de pannes et par l'expression d'un critère d'optimisation prenant en considération la robustesse. L'algorithme de la meilleure réponse semble une base solide pour explorer cette voix.

En ce qui concerne le problème de routage des liens virtuels, des futurs travaux pourraient explorer des moyens pour obtenir une meilleure sélection des arbres candidats. Cela consisterait à envisager d'autres approches que le problème d'arbre de Steiner en considérant plusieurs objectifs qui soient des compromis entre la charge des liens et les délais de bout-en-bout. De plus, l'hypothèse d'un modèle de file M/M/1, qui est un moyen efficace pour mesurer la performance, pourrait être relaxée en implémentant directement les aspects spécifiques du réseau AFDX.

Une autre piste envisageable serait de considérer le problème de dimensionnement des liens virtuels. Ce serait alors une phase intermédiaire entre le routage et l'ordonnancement. Le problème consisterait à trouver un regroupement des messages dans un ou plusieurs liens virtuels et à calculer leur valeur de BAG et de MFS tout en respectant les délais de bout-en-bout et en minimisant la bande passante.

Les propriétés de l'algorithme de la meilleure réponse pourrait être aussi le sujet d'une étude plus approfondie. Bien qu'il soit prouvé qu'il converge et que la solution optimale peut être obtenue, l'analyse du temps de convergence pourrait être affinée ainsi que la connaissance de l'erreur induite. Pour ce dernier point, le but serait de dériver une borne supérieure sur le prix de l'anarchie [95], c'est-à-dire le ratio entre la pire solution que l'on peut obtenir et l'optimum.

Enfin, d'une manière plus générale, l'algorithme de la meilleure réponse pourrait être généralisé pour des problèmes d'optimisation de type maxmin. Pour cela, son comportement devrait être mieux

analysé en fonction des conditions d'utilisation et de nouvelles contraintes, par exemple de précédence, pourraient y être ajoutées.

Introduction

Resource allocation in IMA-based avionic systems

The last couple of years have seen a great standardisation effort in the avionic domain, in the aim of supplying applications a modular execution and communication support. Such architectures are called Integrated Modular Avionics (IMA) [152] architectures and are aimed at reinforcing the resource sharing concept between embedded applications. IMA architectures employ a high-integrity, partitioned environment that hosts multiple avionic functions of different criticalities on a shared computing and communication platform. They also provide benefits of flexibility and scalability, and allow for weight and power savings.

Avionic applications, such as those responsible for temperature and pressure conditioning, are embedded into available processing modules, thus providing some services. These applications share resources which are allocated to them and execute under the ARINC 653 [8] specifications. Configuration tables are explicitly completed, indicating all the parameters required for the proper functioning of the system. These configuration tables define the resource allocations in the system and the subsequent application executions. In the case of any modification or the introduction of new components, such as a new application or hardware, system experts have to go through another cumbersome modification of the configuration tables. All this renders the integration or reconfiguration processes uneasy to manage, as any conflict between the applications' parameters or even with the real-time requirements of the system, may lead to hazardous situations.

Contributions

This thesis is devoted to the proposition of decision aiding tools, which are capable of facilitating the choices to be made by system experts. More specifically, it aims at reducing the complexity of configuration phases where resource allocation has to be realized. These resource allocation problems are constituted of:

- The allocation of applications to available processing modules. This includes their respective temporal scheduling on the modules, in a manner respecting all system constraints and requirements.
- The definition of a message flow scheme in the avionic network, between the various communicating applications. This is carried out after successfully achieving the previously mentioned application distribution upon the modules.

The major part of this thesis addresses the allocation and scheduling problem for the avionic applications, which is characterized by the strict periodicity and system specific constraints. A first approach consists of an exact Linear Programming formulation [9, 10, 11] which, given a specified optimization criterion, solves the scheduling problem to optimality. The proposed optimization is not classical, but aimed at provisioning a temporal flexibility for the avionic applications' executions, so that spare processing power can be allocated to meet the resource demand growth with minimal configuration modification.

As will be shown, the proposed exact formulation has its limitations. This necessitates regarding alternate methods, which are able to solve the problem efficiently within bearable time limits. For this reason, an efficient heuristic inspired from Game Theory, is introduced [12, 13]. Avionic applications are specified as players who, each at his turn, try to change their strategies (allocation/scheduling) to enhance the temporal evolution capability of the system, in a manner obtaining an optimization equivalent to that discussed earlier. This method proves its convenience under several circumstances.

For what concerns the second part of this thesis, it focuses on the communication aspect of the IMA architecture. Data and message transmission between applications belonging to different modules is achieved via the avionic AFDX network, which is based on the well-known full duplex Ethernet. Messages are grouped into what is called Virtual Links (VL). These virtual links can be thought of as multicast flows where one and only one source application addresses one or several destinations. Data in a VL is divided into frames, which are limited in size, and are then transmitted in the AFDX network following a predefined transmission rate.

The final configuration of the VLs consists of specifying the routes and links traversed by each one. This is also a decision making phase where, based on the affected allocation of applications to modules, system experts have to route several VLs at the same time while respecting some constraints, such as transmission delay constraints or even load minimization in the network. The thesis proposes two methods for solving this problem. The first method is an exact node-link formulation based on Linear Programming. The second is a two level heuristic based on a link-path formulation. In a first step it filters out undesirable multicast trees for the VLs, and then uses the reduced set of trees to assign routes for each of the VLs.

Organization

The rest of this thesis is organized as follows:

- Chapter 1 discusses the technical context of this thesis, from the introduction of modular avionics to the identification of the various resource allocation problems in IMA-based avionic systems.
- Chapter 2 represents the state of the art on real-time and embedded systems. This chapter focuses mainly on advances in scheduling theory, along with the classification of the scheduling problem associated with resource allocation in avionics.
- Chapter 3 gives an exact Mixed Integer Linear Programming formulation for the thesis' scheduling problem. In this chapter the limitation of such methods is pointed out.
- Chapter 4 identifies the necessity to develop efficient heuristics for the scheduling problem, and hence the proposition of an algorithm inspired from Game Theory.
- Chapter 5 addresses the routing problem in the avionic network. It discusses some state of the art on the subject and proposes methods for solving this problem.
- The Conclusion concludes the thesis and draws some perspectives.

CHAPTER 1

Resource allocation in avionic systems

This thesis is mainly concerned with resource allocation problems in avionic platforms. These problems are considered critical for system designers during the various conception and integration phases of these platforms. This chapter hence discusses currently supported architectures, their evolution, advantages and drawbacks. The various software development and design phases are also introduced in order to better shed light on the level at which this thesis intervenes.

One major resource allocation concern is closely related to the periodic scheduling problem in distributed systems. This periodic scheduling consists of scheduling tasks, or functional operations, that have to be periodically executed at a constant rate over time. Another concern is the network resource allocation for communication and data transmission. In this problem, routing tables for independent transmission tunnels have to be filled out.

As indicated above, this chapter presents the context of this thesis from a technical point of view. For this purpose, the evolution of avionic systems and the introduction of modular avionics are first discussed in Sections 1.1 and 1.2 respectively. In addition, the latter section identifies an important aspect concerning the partitioning of avionic applications. An overview on software development phases are then discussed in Section 1.3. Section 1.4 introduces the research project "SATRIMMAP" under which this thesis was supported. Section 1.5 clearly describes the main problems and objectives considered in this thesis and the contributions brought forward. This chapter is finally concluded in section 1.6.

1.1 Evolution of avionic systems

An avionic system [141, 142] is a set of software, processing units, buses, sensors and actuators, achieving some avionic functions. Such a system should respect real-time and criticality constraints [16, 132], and is responsible for:

- processing information originating from various sensors,

- presenting coherent and adapted information to pilots,
- calculating control orders based on defined rules,
- realizing other functions related to aerial traffic control, passenger comfort, on-board power generation and distribution, etc.

Great effort has always been made to reduce the weight and power consumption in civil aviation aircrafts [152]. Most of which related to the huge volume of wiring. In addition, an increasing demand for installing new on-board functionalities, constitutes a major motive for further advancements in avionics. This leads to an ever-growing complexity, where the evolution of the avionic embedded systems and the amplification of the integrated functions number imply a huge increase in the exchanged data quantity, and thus, in the number of connections between functions.

The overgrowing complexity of embedded equipments limits evolution possibilities. Generally, like Moore's law on processor power, it was shown that the complexity of avionic systems double every five years [37]. Meanwhile, the evolution of the aviation market has led to greater pressure for the reduction of costs, especially those related to specific communication electronics. Thereby, the modular avionics concept was introduced, and aims to share processing and communication resources. Its main objectives are:

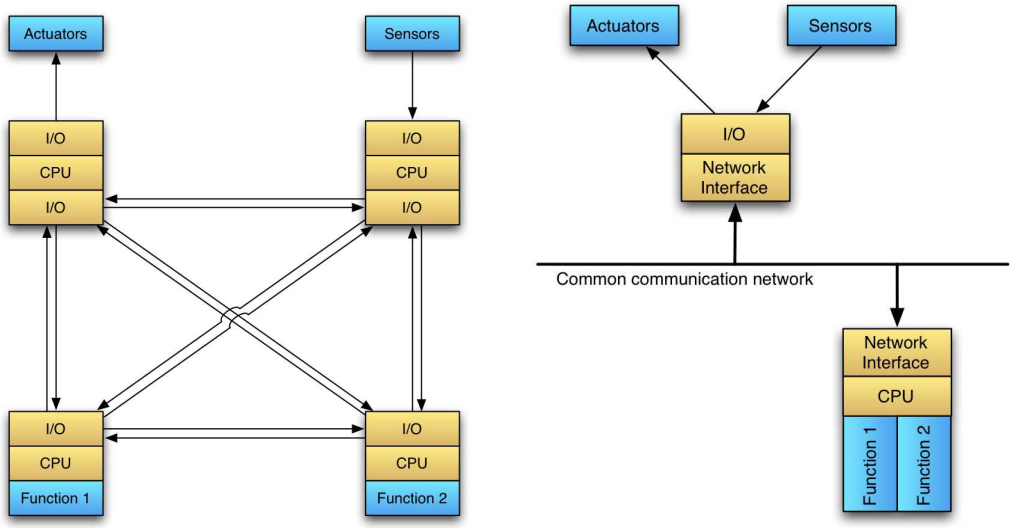
- data transmission with strong temporal constraints,
- reliability in information exchange following the client/server model,
- cost reduction by reusing commercial off-the-shelf (COTS) products, but with certification constraints.

Avionic systems are also subject to very strict constraints to guarantee their proper operation since a single failure in a critical equipment can lead to catastrophic consequences. Avionic systems are very demanding in terms of strict real-time constraints, limited complexity, size, volume and reduced weight. They furthermore have to support delicate operation conditions (temperature, pressure, vibration, electromagnetic environment, etc.).

In the following section, the Integrated Modular Avionics (IMA) architecture which constitutes the workspace of this thesis is introduced.

1.2 The Integrated Modular Avionics architecture

The Integrated Modular Avionics (IMA) concept is, as was the case before, based on the presence of several sub-systems, such as flight control, landing gear and others [142]. However, such IMA architectures employ a high-integrity, partitioned environment that hosts multiple avionic functions of different criticalities on a shared computing platform [152].



(a) Federated architectures. (b) IMA architectures.

Figure 1.1: Example on resource allocation in federated and modular avionic architectures.

In classic avionics, equipments are distributed all over the aircraft in proximity of the sub-systems they command. This so-called federated architecture had rather limited resource sharing and dependencies between systems were well understood [133]. All these equipments are interconnected and are associated to the cockpit from which they receive orders. Identifiable inconveniences of this architecture include the weight imposed by the large volume of wiring, and the high equipment cost.

The approach to managing resources, such as computing, communication and I/O can be identified as the fundamental architectural difference between federated and IMA systems [152]. In contrast to federated avionic architectures, which implement independent collection of dedicated computing resources for each avionic function, those based on IMA provide a shared computing, communications and I/O pool that is partitioned for use by multiple avionic functions. Figure 1.1 for example shows how a set of four computing processors in a federated architecture can be replaced by one in an IMA-based one.

Since IMA makes use of shared computing resources, the total number of computing processors and associated circuitry is reduced. So is the associated infrastructure, such as power, cooling, and redundancy mechanisms. In addition, dedicated communication channels are replaced with a common one, and hence less wiring and dedicated I/O interfaces. All this translates into better power and weight savings for the overall aircraft.

The IMA architecture was introduced to solve problems arising from the evolution of previous architectures. It is based on an architecture of type “systems”, physically distributed in the aircraft [43]. A main originality resides in the hardware platform standardization, where developers of an avionic function no longer need to waste time on developing computing resources. This standardization of

resources makes it easier for developers to focus on their software, and eases certification efforts. This leads to reduced development expenses and design cycle times [152].

1.2.1 Architecture components

The IMA physical architecture is defined by the ARINC 651 [3] specification. Resources are regrouped in generic modules called Line Replaceable Units (LRU), that are in turn grouped into cabinets. *Intra-cabinet communication* is realized via specific buses, generally defined by ARINC 659 [4]. *Inter-cabinet communication* is achieved using a network based on full-duplex switched Ethernet [78], as Airbus has chosen for its A380. The aforementioned modules can be of three types:

- Core Processing Modules (CPM), which are modules responsible for application execution.
- Input/Output Modules (IOM), enabling communication with non-IMA compliant elements. An example is the Remote Data Concentrator (RDC) that converts message formats between on-board sensors and actuators, and core modules.
- Switches and gateways, representing the communication backbone.

The software architecture is described by the APEX (APplication EXecutive) standard that offers to applications a generic interface towards the operating system [8].

1.2.2 The avionics AFDX network

The Avionics Full Duplex Ethernet (AFDX) constitutes one of the major technological breakthroughs in the avionics of the A380 [6]. In effect, and for the first time for such aircraft category, the avionic system is based on a redundant and reliable Ethernet network.

With the introduction of the first AFDX specifications (around 1999), the best candidates were a combination between Ethernet and TCP/IP amongst technologies arising from computer networks [50, 58], in opposition to ATM amongst technologies in the telecommunication domain. Key criteria for the final choice were avionic specific constraints (security, temporal problems), the arrival of Ethernet switching and the size of the computer market as opposed to that of telecommunication equipments. The choice has therefore landed on the switched Ethernet (full-duplex mode) [61].

The AFDX standard defines the electrical and protocol specifications [7] for the exchange of data between *Avionics Subsystems*. One thousand times faster than its predecessor, the ARINC 429 [5], it built upon the original AFDX concepts introduced by Airbus.

As shown in Figure 1.2, an AFDX network comprises the following components:

- **Avionics Subsystem:** It represents the traditional systems on board an aircraft, such as the flight control computer, global positioning system, the pressure monitoring system, etc. A processing unit, or an *Avionics Computer System*, provides a computational environment for the Avionics Subsystems. Each Avionics Computer System contains an embedded *End System* that connects the Avionics Subsystem to an *AFDX Interconnect*.

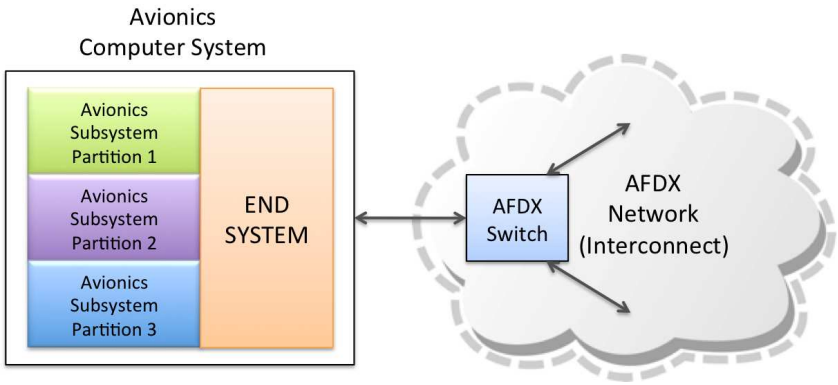


Figure 1.2: Composition of an Avionics system

- **AFDX End System:** It provides an "interface" between the Avionics Subsystems and the AFDX Interconnect. This interface exports an application program interface (API) to the various Avionics Subsystems, enabling them to communicate with each other through a simple message interface.
- **AFDX Interconnect:** It is a full-duplex, switched Ethernet interconnect. It generally consists of a network of switches that forward Ethernet frames to their appropriate destinations. This switched Ethernet technology is a departure from the traditional ARINC 429 unidirectional, point-to-point technology and the MIL-STD-1553 bus technology [103].

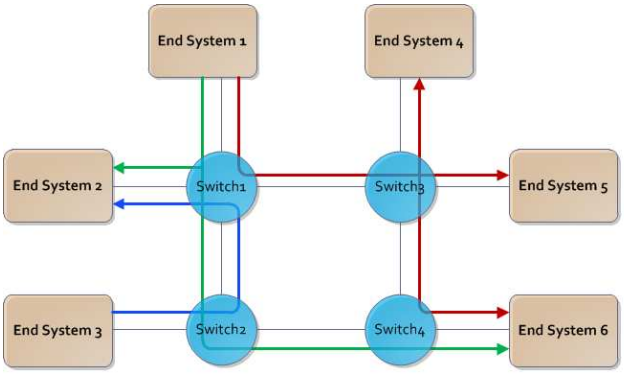
It should be noted that, in addition to the AFDX network, some non-AFDX compliant equipment (utilizing CAN [82] buses, ARINC 429, etc.) can be found on board the aircraft. Therefore communication between different interfaces is done via the Input/Output modules such as the RDC.

For what concerns the AFDX switches, they route based on not only MAC addresses, but also on what is known as *Virtual Links* (VL) which are unidirectional logic paths sourced by only one transmitting End System but can have multiple receivers [14]. These links are defined in a table inside each switch. Refer to Figure 1.3 for an example. In this figure, ports simply indicate on what links data is received and sent.

1.2.3 Partition segregation

Applications in avionics are software responsible for one or several functionalities in an airplane. An example is the position measurement application which is responsible for the acquisition and display of the altitude, longitude, orientation, etc. of the plane. *Functions* are code blocks of an application that carry out a part of its functionality. Hence, an application consists of a set of functions that all together lead to its proper execution, each of which may have a certain input and output.

Functions resident on a processing module are partitioned with respect to space (memory partitioning) and time (temporal partitioning). A *partition* is therefore a program unit of the application



Switch 1 configuration table

	Input port	Output port
VL ID=1	1	4
VL ID=2	1	2, 3
VL ID=3	3	2

Figure 1.3: A simple AFDX network with three Virtual Links.

(including a set of functions) designed to satisfy these partitioning constraints. This partitioning is crucial as to prevent faults from propagating throughout the system. Functions become in this manner totally independent, where faulty ones can be isolated without affecting much of the system integrity.

1.2.3.1 Partition spatial constraints

Each partition is allocated a set of spatial resources (memory, non-volatile memory (NVM), etc.) in a static manner, that is to say that the module integrator has the task of assigning maximum allowed spatial resources to each partition while respecting space segregation between them.

Consequently, the role of the operating system (OS) here is to provide protection for partition data against any modification from the other partitions. The OS also has to monitor application activity as compared to allowed resources which are statically allocated through configuration tables. Figure 1.4 demonstrates memory segregation between two partitions.

1.2.3.2 Partition temporal constraints

A module may host several partitions running with different periods following a certain scheduling scheme. All information concerning period, duration and deadline of partitions are defined statically.

This scheduling must lead to a temporal allocation for each partition allowing the proper execution of all functions defined in this partition.

In avionics, scheduling of partitions is predefined in a manner respecting the real-time aspect of the system, that is ensuring proper execution of all functions before their respective deadlines. The OS has to make sure that this scheduling is carried out flawlessly and monitor irregularities.

Define the following characteristics:

- Partitions have no priority.

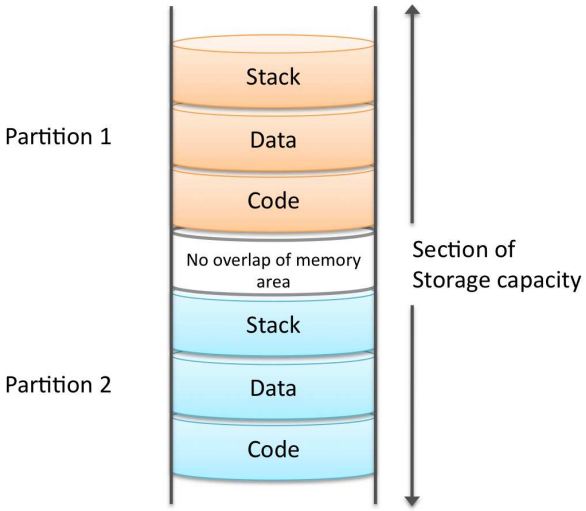


Figure 1.4: Memory segregation between partitions.

- Partition scheduling is predetermined and repetitive with a strict periodicity.
- Core module level OS exclusively ensures the temporal allocation for the partitions.

Each partition is allocated a window for execution. At the end of this window the partition is suspended and execution is given to another partition. These windows are defined in a manner ensuring the efficient performance of the system while respecting real-time aspects.

The *Major Frame* (MAF) as can be seen in Figure 1.5 is a fixed duration which is periodically repeated throughout the module's runtime operation (it represents the periodicity of the scheduling). This schedule periodicity can be defined as the least common multiple (LCM) of the module's partition periods; given that all partitions are released together for the first time, and provides a sufficient time-horizon for ensuring a correct temporal scheduling [34].

It should be also noted that a granularity of $100\mu sec$ is defined. All partition periods, and the MAF, should be a multiple of this granularity.

Major partition attributes are:

- **Duration:** Each function requires a certain amount of time to fully execute, hence a partition duration can be defined from the duration of its functions.
- **Deadline:** After the activation of a partition, the execution should be completed before a predefined hard deadline.
- **Period:** The period defines the periodicity of partitions. This periodicity is strict (i.e. partition executions should be exactly one period apart).

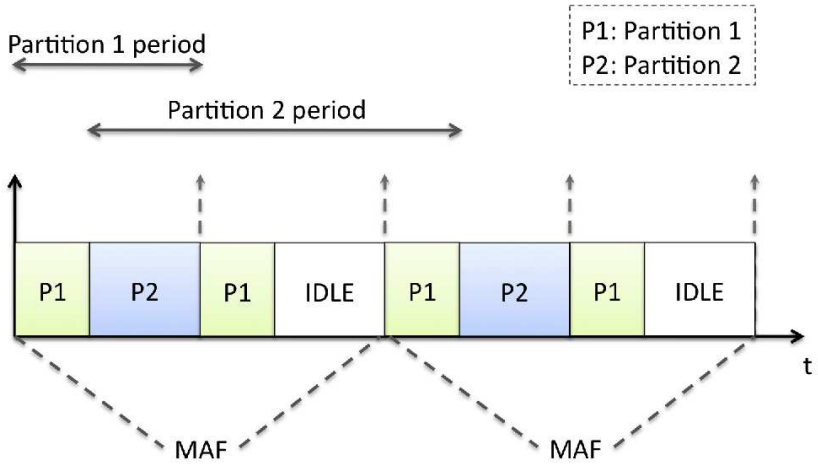


Figure 1.5: Partition execution in a MAF.

- **Allocated resources:** Represents the global resources allocated to each partition, such as memory.

1.2.3.3 Partition communication

Communication between partitions is essential for the exchange of data, given that spatial partitioning prohibits sharing of resources and data areas. This exchange can only be done via inter-partition communication.

Inter-partition communication is based on the sending and receiving of API messages, which are, in the case the communicating partitions are located on different Core Processing Modules, placed into AFDX frames and sent over the AFDX network. However, if the two communicating partitions are located on the same module, then communication can be achieved using for example buffers or blackboards. Otherwise, in the case of data exchange between partitions and other non-AFDX compliant equipments, such as sensors, messages are converted to the appropriate formats in the RDC, based on the type of interfaces the destinations are using.

Messages having the same path in the AFDX network may be grouped into one message to minimize number of transferred bits and hence ensure faster delivery.

1.2.3.4 Partition redundancy

An important aspect of the IMA architecture is redundancy, where applications are re-placed as mirrors on other modules connected to different switches to ensure the safety of the system. The following are three types of redundancy that may exist:

- **Standby:** The redundant application is idle, it does not receive or send any data or messages.

- **Cold:** The redundant application has the same input as the master one, computes the data but does not send any messages or data.
- **Warm:** The redundant application has the same input as the master one, computes and sends data and messages, and hence, the user receives redundant information.

1.3 Overview on the software development process design in avionics

To properly indicate the contributions of this thesis, the software (avionic applications) development process for the IMA architecture is hereafter described. For the sake of simplicity, this process is reformulated through the V-Model [129] which is a generic software development process. The reformulation aims at apprehending the implicated design process.

The V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing. It deploys a well-structured method in which each phase can be implemented by the detailed documentation of the previous phase.

The V-Model consistes of a number of phases presented in Figure 1.6. The verification phases are on the left hand side of the V, the coding phase is at the bottom and the validation phases are on the right hand side.

1.3.1 Requirements analysis phase

In the requirements analysis phase, the requirements of the proposed system are collected by analyzing the needs exerted on it. This phase concerns establishing what the ideal system has to perform. However it does not determine how the software will be designed or built. Usually, a document called the system requirements document is generated.

The system requirements document will typically describe the system's functional, physical, interface, performance, data, security requirements, etc. as expected by the user and defined by the system designer. It is one which the analysts use to communicate their understanding of the system. The document is carefully reviewed as it would serve as the guideline for the system designers in the system design phase. The acceptance tests are designed in this phase.

1.3.2 System design phase

System design is the phase where system engineers analyse and understand the business of the proposed system by studying the system requirements document. They will then design and define the system structure which will be compliant with the system and platform constraints. If any of the requirements are not feasible, the issue is directly highlighted.

In addition, the system designer will generate the detailed functional document which details a specific functional need or more particularly the implementation and the expected characteristics from the software (avionic applications).

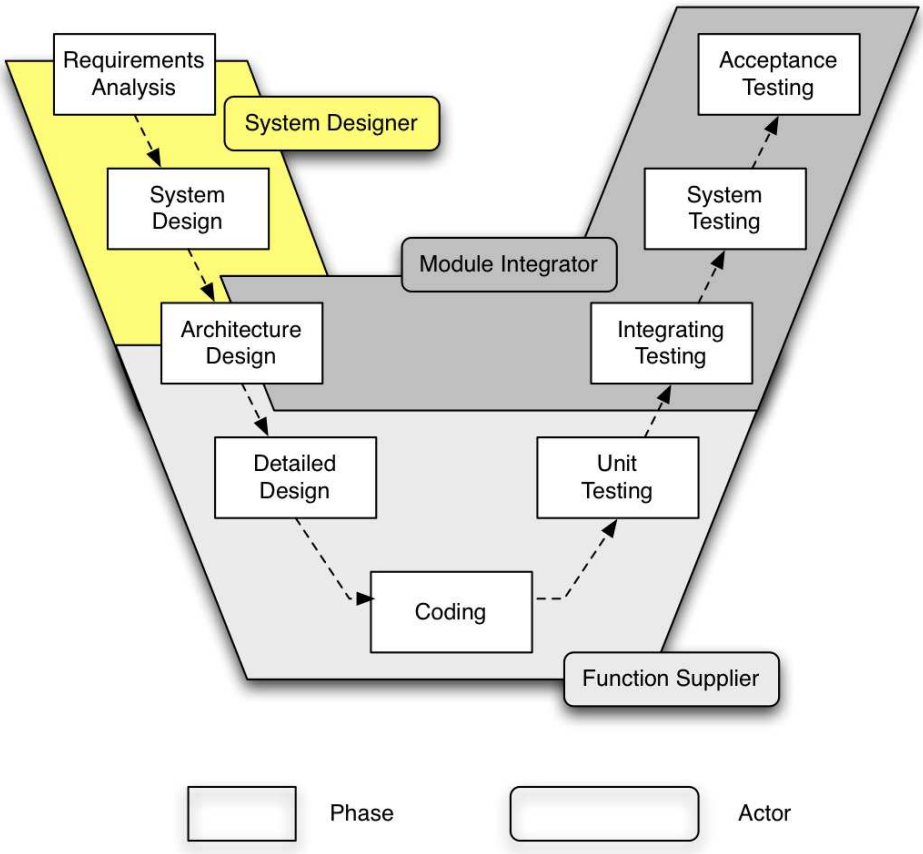


Figure 1.6: A representation of the integration process view from IMA with the V-Model approach.

1.3.3 Architecture design phase

This can also be related to as high-level design phase. The baseline in selecting the architecture is that it should realize all which typically consists of the list of modules, brief functionality of each module, interface relationships, dependencies, etc.

In this phase, and after the definition of various elements of the architecture (processing modules, memory, etc.), resource allocations are realized. Upon coordination with the function supplier, the module integrator intervenes in defining an initial, high level resource allocation on the modules, and in the network. That is to say he fulfills the role of distributing and scheduling functions upon the processing modules and, based on this allocation, specifies message transmission routes in the AFDX network.

1.3.4 Detailed design phase

This can also be referred to as low-level design phase and realized by the function supplier. The designed system is broken up into smaller units and each of them is explained so that the programmer can start coding directly. Upon the reception of the requirements and functional documents from the system designer, the function supplier will refine them into the software requirement document.

According to the complexity of the function, the function supplier will produce the software description document that will enable him to trace the requirements on its implementation choices. The functions, and subsequently partitions, are broken up into functional threads, called processes. These processes should verify proper executions in the schedules defined in the preceding phase.

1.3.5 Coding phase

In this phase, and after the definition of the partitions' processes, the code is written by the function supplier for the application that he is responsible of.

1.3.6 Unit testing phase

Unit testing implies the first stage of dynamic testing process. It involves analysis of the written code with the intention of eliminating errors. It also verifies that the codes are efficient and adheres to the adopted coding standards. This may be carried out by software developers, or in other words function suppliers.

1.3.7 Integration testing phase

In this phase the separate modules will be tested together by the module integrator to expose faults in the interfaces and in the interaction between integrated components. Testing is usually a black box as the code is not directly checked for errors.

1.3.8 System testing phase

System testing will compare the system specifications against the actual system. The system test design is derived from the system design documents and is used in this phase. Sometimes system testing is automated using testing tools. Once all the modules are integrated several errors may arise. Testing done at this stage is called system testing and is carried out by the module integrator. It also includes verification that the actual resource distribution is as it would be expected, or that the resources utilized by a function is within the resources provided by the configuration.

1.3.9 Acceptance testing phase

Acceptance testing is the phase of testing used to determine whether a system satisfies the requirements specified in the requirements analysis phase. The acceptance test design is derived from the requirements document. The acceptance test phase is the phase used by the customer (e.g. Airbus) to determine whether to accept the system or not.

1.4 The research project SATRIMMAP

This thesis is supported by the French National Agency for Research (ANR) and was conducted under the research project SATRIMMAP [134] which stands for Safety and Time Critical Middle-ware for future Modular Avionics Platforms. This project's consortium consists of six partners: Airbus, CEA LIST, IRIT, LAAS-CNRS, ONERA and QoS Design.

The main objective of "SATRIMMAP" is to raise the difficulties associated to integration or reconfiguration phases, where resource allocation tables and parameters have to be released, or even redefined, each time the system changes or a new element is introduced (e.g. introduction of a new function). This is to be achieved through the study and the definition of tool-sets. The ambition is also to ensure a maximum transparency between applications and the architecture support. The aforementioned tool-sets must ensure the criticality constraints of avionic systems, and in particular the determinism, the reliability and the predictability of the various components. For this purpose, the project concentrates on three points:

- the formalisation of a proper model for avionic applications,
- the configuration and reconfiguration of IMA architectures, and
- the definition of suitable tool-sets for facilitated configuration purposes.

These steps include the study of techniques and methods aimed at the performance evaluation for an IMA architecture.

Amongst the several tool-sets to be conceived in this project, this thesis is concerned with the resource allocation one shown in Figure 1.7. The latency checker, which is the responsibility of another project partner, thoroughly evaluates the delays in the network [101, 100]. Message transmissions between the various partitions are checked for any requirement violation, after which feedback is supplied specifying whether the proposed configuration (resource allocation) is viable or not. The complete system is afterwards simulated and checked for any irregularities.

1.5 Objectives of the study

As mentioned in the preceding section, the drawbacks of the IMA architecture necessitates its enhancement. In order to achieve this, many aspects of the architecture have to be tackled. This thesis concerns the decision support aspect in system configuration phases.

The several development phases represented in Section 1.3 through the V-Model include the efforts of the system designer, the module integrator and the function supplier. In summary, the system designer presents system requirements and specifications to be followed. After element definition, the module integrator allocates the partitions on the different resources upon negotiation with the function supplier, gives them a proper scheduling and then configures routing tables for ensuring a suitable message transmission in the network. The function supplier moves on to define the processes and to do the necessary software coding. Once done, several testing phases follow to ensure the good functioning of the system.

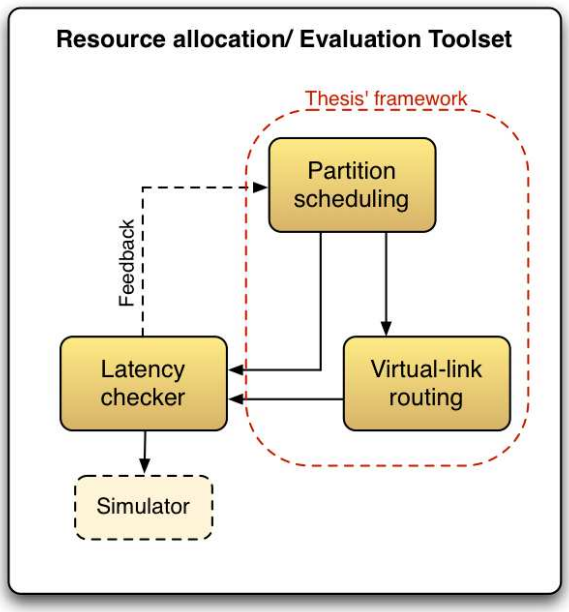


Figure 1.7: Resource allocation and evaluation tool-set.

Currently, the module integrator is faced with an important challenge, the resource allocation problem which represents a critical phase in the conception of IMA-based systems. Continuous advancements in avionics will give rise to significantly more complex systems in which the volume of constraints and requirements pose serious integration and configuration difficulties. System experts will inevitably require aiding or even decision-making tools for resource allocations. This thesis is devoted to the proposition of such tools which are capable of facilitating the choices to be made by the system experts. The two distinguishable aspects of the resource allocation problem for which this thesis contributes are:

1. The distribution of partitions on the core processing modules (CPM) and their respective temporal scheduling, in a manner respecting all system constraints and requirements.

2. The routing of message flows, in the form of Virtual Links (*cf.* Section 1.2.2), between partitions. This is carried out after successfully scheduling partitions on the modules, where based on this choice, the VLs are to be routed in the AFDX network.

Figure 1.8 shows the place of the proposed tools in the development phases already discussed.

1.5.1 Scheduling objectives

The scheduling objectives include the introduction of functional aspects that may aid the module integrator in allocating resources for the partitions. That is partition mapping and scheduling on the modules.

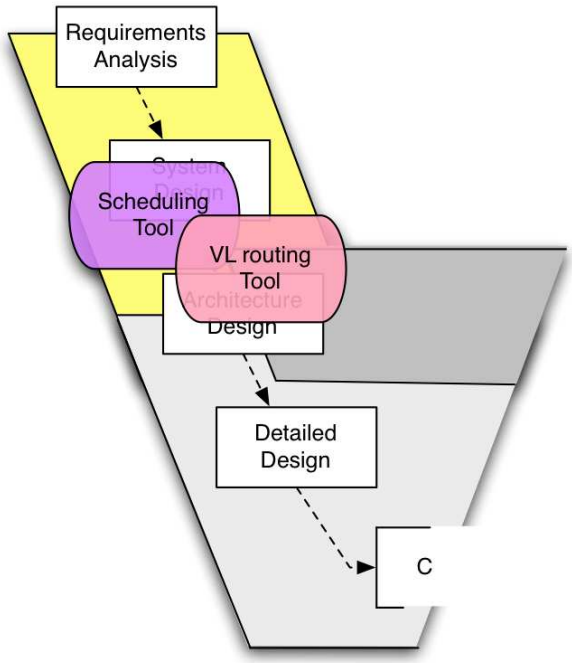


Figure 1.8: Position of the proposed tools in the development process.

Starting from a set of well defined avionic partitions, their distribution onto the resources of the IMA architecture, including several distributed processing modules, has to be achieved. This distribution should ensure a viable resource mapping and temporal scheduling (proper execution of all partitions).

The instances of partition deployment model are under IMA Integrator responsibility and must verify all partition stand-alone qualifications. The scheduling tool has to compute the better sharing of the resources between the partitions. This sharing must take into account application needs in terms of strict periodicity, time duration, memory capacity and also in terms of segregation (or redundancy) of partitions, co-localization of different partitions in the same module or not, and in the same cabinet or not.

To optimize solution for configuration, the mapping tool follows a major need to maximize the spare area for a partition, so, if an application is modified and a new budget should be allocated, this application must be modified or re-localized without re-qualifying all applications sharing the module.

1.5.2 Virtual Link routing objectives

Once partitions are successfully placed on the various modules, routing tables need to be completed. The Virtual Links, implicated in this process, are defined from the message exchange requirements between partitions. After their definition, source and destination End Systems can be interpreted for each

VL from the partition allocations on modules.

Routing in the AFDX network should respect not only network constraints, such as link capacities, but also transmission delay requirements for each VL. This routing should optimize the residual bandwidth so as to ensure future evolution possibilities, such as the introduction of a new VL, or the modification of an existing one.

1.6 Conclusion

In this chapter, key attributes of an avionic platform were briefly discussed to then introduce the main aspects of the IMA architecture, which constitutes the framework for this thesis. The different components of IMA were presented, from the AFDX network to the various system and software development phases.

The emphasis was on the need for design aiding tools that support system designers in filling out configuration tables, more specifically those related to resource allocation. Be it related to partition distribution and scheduling on modules, or even subsequent routing of Virtual Links. It should be noted however that the main contribution of this thesis concerns the former partition scheduling problem, which is considered as the primary objective. The latter Virtual Link routing follows as a secondary one where assertion on transmission delay requirements for VLs is performed by other project partners (*cf.* Latency checker in Figure 1.7). Their feedback will allow the interpretation of the global feasibility of the solutions supplied by the tools.

CHAPTER 2

State of the art

As indicated in the previous chapter, resource allocation in avionic platforms constitutes the main interest of this thesis. Real-time embedded systems are clearly implicated as they constitute a major classification for avionic systems. It is hence indispensable to define such systems and specify the aspects that apply in avionics. Among the resource allocation problems, this thesis is principally focused on the real-time scheduling aspect, and hence, the state of the art on this issue is hereafter presented. This shall lead to the proper identification of this thesis' main problematic amid all that already exists, including the associated theoretical concepts. In addition, methods to be implemented in later chapters are also to be discussed.

Section 2.1 introduces real-time systems in addition to their classification. Section 2.2 details scheduling problems including some generalities and variations. An overview on embedded systems is then included in Section 2.3. After a brief discussion on the complexity of scheduling problems in Section 2.4, some known algorithms for uniprocessor and multiprocessor scheduling are introduced in Section 2.5. Particularities of the scheduling problem, concerned with this thesis' resource allocation problem, is clearly indicated in Section 2.6, along with methods that may be implemented for its resolution. Section 2.7 finally concludes this chapter.

2.1 Introduction to real-time systems

Real-time systems are digital computing systems that allow the establishment of applications where timing constraints are to be satisfied. These real-time systems are reactive, due to there continuous reaction to stimuli coming from the system's environment. For such systems to correctly function, they have to respond to every stimulus they receive. The response to input stimuli not only depends on their nature, but also on the system state at the time of stimuli arrival. A real-time system (*cf.* Figure 2.1) is usually constituted of a *control system* that controls a certain *physical system* in its environment. The bidirectional interaction between these two systems is achieved via two peripheral subsystems: an *actuation system* that allows a certain modification of the physical system (motors, pumps, etc.), and a

sensory system that supplies the control system with information related to the physical system, through the use of sensing devices (barometers, cameras, etc.).

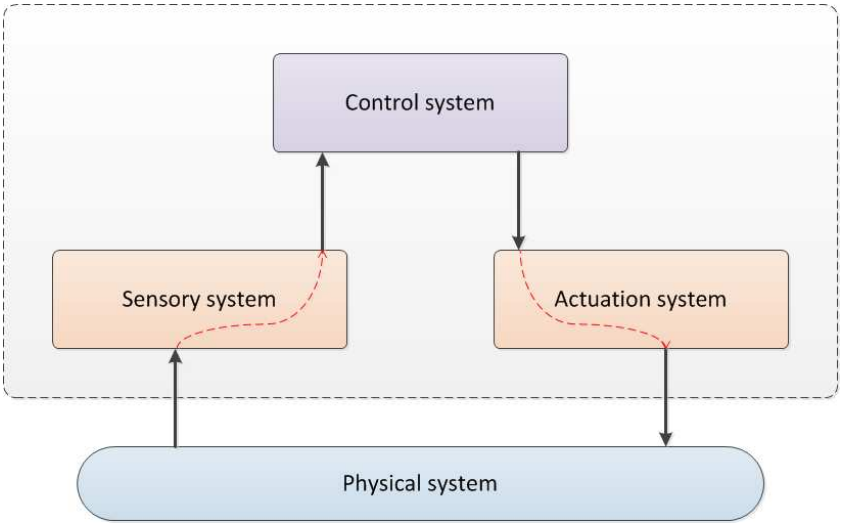


Figure 2.1: A typical real-time system components.

The correctness of a real-time system not only depends on its logical results, related to the implementation of a certain algorithm, but also on the temporal availability of the results. Thus, a real-time system must satisfy two important constraints:

- Logical accuracy: calculate correct system outputs as a function of inputs.
- Temporal accuracy: the results are affected at the appropriate moment. That is to say, a delay on result output is considered as an error that can cause serious consequences. A real-time system must hence respect real-time constraints, such as the time limit for giving a result.

Data processing is becoming more noticeable in domains where the interaction with the environment represents the essential mainspring of the system. Resorting to computers in this genre of systems is of interest due to the affordability of offered functionalities, in terms of development time, fabrication time and cost, congestion or weight, as compared to purely electronic or mechanic solutions. In addition, processors offer a great flexibility via programming, the possibility to add new functionalities, the ability to apply updates, or product customizations that are often found to be a set of less demanding tasks.

Currently, real-time computer systems can be found in many domains such as avionics, aerospace, automotive, robotics, public transportation, telecommunication, industrial processes and many others. For many of these domains, one of the important characteristics is that the computer system is given a great responsibility in terms of human lives, environmental and economical consequences. *Critical systems* are henceforth spoken of, that are subject to reliability constraints.

Depending on the criticality of temporal constraints, that may lead to different consequences in case of a failure, distinguish two essential types of real-time systems: *hard real-time systems* and *soft real-time systems*.

2.1.1 Hard real-time systems

The majority of critical real-time systems are exclusively constituted of treatments that are subject to strict temporal constraints. This means that the indispensable condition for system operation is that all system treatments have to respect all of their temporal constraints. Hard, or strict, real-time treatments are hence addressed. The concern here, and instead of ensuring fast computation, is more related to what is called the predictability [144], i.e. the ability to predict, a priori, whether the system can meet all hard (critical) timing requirements. This includes the ability to define nominal operation conditions in terms of environmental assumptions with which the system interacts.

2.1.2 Soft real-time systems

Another class of systems is less demanding when it comes to respecting the temporal constraints. Systems of this class, called soft real-time systems, can suffer an acceptable rate of temporal faults without causing catastrophic consequences. This class contains, amongst others, systems where the quality is appreciated as a service with a human aspect (susceptible to mistakes): as in the case of multimedia systems and applications (telephony, video, etc.). The measure of constraint satisfaction takes the form of probabilistic data: quality of service relative to a particular service (e.g. number of images or number of sonar samples per second), relative to system comportments as a whole (e.g. number of treatments that were able to be carried out), or the combination of the two. A problematic to this class of systems is to evaluate the quality of service that the system offers or can offer during operation, as a function of the system's environment characteristics [80].

In this thesis, the main interest is in hard real-time systems which characterizes the avionic system introduced in Chapter 1.

2.2 Generalities on real-time scheduling

The real-time scheduling problem amounts to finding a sequence for executing tasks on every processor in a given architecture. Tasks in a real-time system are subject to temporal and other types of constraints. The goal is hence to foresee, with the maximum possible exactitude, the temporal behavior of the system. In literature, different terms to describe the act of associating a set of tasks to a group of processors can be found. In this thesis the terms "mapping" and "allocation" are used to express the idea of associating tasks to processors. The term "scheduling" however, is more precise where it not only specifies the association of tasks to the processors, but also their temporal behavior on them. Whenever indicated, the term "temporal scheduling" is used to describe the temporal behavior of a set of tasks on a given processor.

2.2.1 Real-time tasks

The main part of a real-time system consists of tasks, i.e. of computer processes. A task is a computation that is executed by the processor in a sequential fashion: it is a sequential execution of code that does not suspend itself during execution. The characterization of a task i can vary from one scheduling model to another. Among the many parameters that can be found in literature on recurrent tasks (require repeated execution), consider the following (*cf.* Figure 2.2):

- The ready time or release time r_i represents the date after which task i can proceed its execution for the first time, it is also called activation date. In the case of periodic tasks, with period T_i, an instance $k \in \mathbb{N}$ of a task i is released at $r_i^k = r_i^{k-1} + T_i$, with $r_i^0 = r_i$.
- The deadline d_i represents the duration before which the task should terminate its execution. Exceeding this limit can cause a temporal violation. Two kinds of deadlines can be differentiated:
  - Hard: meeting a task deadline is critical for the system functionality. Missing a hard deadline is considered a definite failure, and leads to catastrophic consequences.
  - Soft: it is desirable to meet a task deadline, but occasionally missing it can be tolerated. A task with a soft deadline is expected to be completed either before the deadline or as early as possible after it.
- The start time t_i and end time e_i are respectively the date at which task i executes for the first time on the associated processor and the date at which it finishes its execution (for the first time). If the task requires execution with a period T_i, then t_i^k and e_i^k represent the start and end times for instance k of task i.
- The execution time b_i represents the duration of execution for task i. This parameter is considered in the majority of works on hard real-time scheduling as the worst-case execution time (WCET) [28] on the processor on which it is going to execute. The WCET represents an upper bound for the execution time of a task. In other words, although a task can finish earlier, the WCET represents an estimation on the maximum possible duration this task needs to execute. In order for this parameter to be valid, its value should not be overestimated. And after this estimation, the task's execution time must never exceed the estimated value.
- The laxity l_i corresponds to the maximum duration a task i can delay its execution without surpassing its deadline. As a function of time, $l_i(t)$ represents the same indicated duration while considering only the remaining execution $b_i(t)$ at time t, that is $l_i(t) = d_i - t - b_i(t)$.

If release times are included for a task system (first activation dates for tasks are known a priori), such a system is called a *concrete task system*. Otherwise, the task system is a *non-concrete* one. Note that given an incomplete task system (release times are not specified), an infinite number of complete task systems can be obtained by simply specifying the release time for each task. In the special case for which the release times for all tasks are equal zero, the complete task system is called a synchronous task system.

For what concerns task periodicities, a task can either execute periodically at regular intervals, or randomly with no specified period (non-periodic tasks).

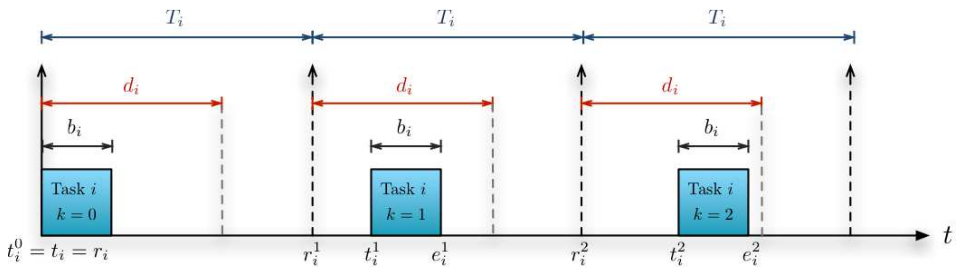
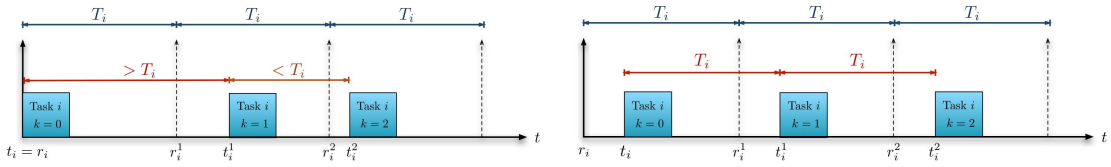


Figure 2.2: Graphical representation of a real-time model.



(a) Classic periodicity.

(b) Strict periodicity.

Figure 2.3: Periodicity types in a real-time system.

2.2.1.1 Periodic tasks

A task i is referred to as periodic of period T_i if the activation of this task is produced at regular time intervals T_i. The service provided by this periodic task is hence made indefinitely. For this reason, each of these executions is called an instance of task i. Figures 2.3a and 2.3b represent two types of periodic task periods:

- Classic period: It is the most used in real-time applications. Each instance k of task i is allowed to execute after its respective activation date r_i^k. Although no constraints are imposed on the distance between the start times of two successive instances k and $k+1$, these instances should respect deadline constraints such as the hard deadline one ($e_i^k \leq r_i^k + d_i$).
- Strict period: As compared to the classic period, execution start dates t_i^k for the instances of a periodic task i must be exactly T_i apart. Consequently, given task i offset (first execution date) t_i, instance k start date should satisfy: $t_i^k = t_i + k\,T_i$. In this case, it suffices to verify the deadline constraint for only the first instance of a task's execution. It can be also easily seen that the strict period forms a special case of the classic one.

Task periodicities can be also classified as *harmonic* or *non-harmonic*. Harmonic periods imply that every period divides every other one of greater value ($T_i | T_j$ or $T_j | T_i$ for all i, j), e.g. the set $\{2, 6, 12\}$. Non-harmonic periods are however not constrained, periods may be primary, multiples or of no relation between each other, e.g. the set $\{2, 6, 8\}$.

Due to the strict periodicity aspect discussed in Section 1.2.3.2, tasks considered in this thesis are characterized by a strict period and not a classic one. In addition, no hypothesises on the harmonic and non-harmonic classifications of periods are considered.

2.2.1.2 Non-periodic tasks

A task is specified as non-periodic if its successive activations arrive at an irregular basis, that is to say that the task does not abide by a predefined period. There exists two types of non-periodic tasks:

- **Aperiodic:** activation dates are random and cannot be anticipated. Task execution is the product of internal or external events that can trigger at any instant. In [143], different procedures for scheduling the execution of aperiodic tasks in real-time systems are exposed.
- **Sporadic:** a special case of aperiodic tasks where a lower bound on the time duration that separates two successive activations is known. These type of tasks are frequently specified as periodic tasks [104], where the aforementioned lower bound is considered as the task period, in order to apply existing results on periodic tasks.

These two types of non-periodic tasks are not considered in this thesis.

2.2.2 Latency

Latency $L(i, j)$ between two tasks i and j is equivalent to the time between the date at which i starts execution and the date at which j terminates. Imposing a latency constraint $L^{max}(i, j)$ signifies limiting this latency to a predefined maximum value, i.e. $L(i, j) \leq L^{max}(i, j)$.

The interest in limiting the time between these two tasks' executions is to guarantee that system performance remains satisfactory and stable [90]. Consider as an example an aircraft pilot hitting the brakes on the taxiway, the system must in this case guarantee, through the intervention of several internal tasks, a response whose time does not exceed certain limits, to avoid undesirable collisions.

In this thesis, imposing latency constraints between tasks is discussed in Chapter 3.

2.2.3 Classes of scheduling problems

- Uniprocessor/Multiprocessor: if the architecture has only one processor, the scheduling problem is said to be uniprocessor. If several processors are available then the problem is a multiprocessor one.
- Off-line/on-line: a scheduling is classified off-line if it is carried out on the entire task execution sequence before actual task activation. The schedule generated is subsequently stored in configuration tables and later executed by a dispatcher. The task set has to be fixed and known *a priori*, so that all task activations can be calculated off-line. The main advantage of this approach is that the run-time overhead is low and does not depend on the complexity of the scheduling algorithm used to build the schedule. However, the system is quite inflexible to environmental changes [42]. On-line scheduling however, is used if scheduling decisions are taken at run-time every time a new task enters the system or when a running task terminates. With on-line scheduling, each task is assigned a priority, according a predefined rule. These priorities can be either fixed, based on fixed parameters and assigned to tasks before their activation, or dynamic, based on dynamic parameters that may change during system evolution.

- Preemptive/non-preemptive: if task executions can be interrupted by others then the problem is preemptive. Preemption indicates that a task execution can be suspended temporally so that another one can execute due to higher priority for example. With non-preemptive scheduling, a task execution, once started, is proceeded until its completion without being interrupted.
- Idling/non-idling: in general, a processor executes a task once it is ready to execute and cannot retard this execution if it has no other tasks to serve. It is said, in this case, that the scheduling is non-idling, meaning there is no insertion of idle times (*work-conserving*). Nevertheless, in some cases a system may not be schedulable unless kept idle for a certain time. It is hence said that the scheduling is idling (idle time insertion).
- Optimal/non-optimal: a scheduling algorithm is said to be optimal [154] for a given problem if, for any set of tasks, it always produces a schedule which satisfies the constraints of the tasks whenever any other algorithm can do so. If the optimal algorithm does not find a solution then no other algorithm can do so. Otherwise, a non-optimal scheduling algorithm aims to finding approximate solutions (by ignoring part of the constraints for example).

This thesis is concerned with the off-line scheduling of non-preemptive tasks in a work-conserving system. For this matter, optimal and approximation algorithms are going to be studied.

2.2.4 Non-preemption in scheduling problems

In avionics, preemptive scheduling of tasks is prohibited. This choice was made to ease the certification and the predictability of the system. In addition, and as was indicated in Section 1.2.3, avionic tasks have no priority between each other and are assigned independent sets of resources. This guarantees a certain level of segregation between tasks. The non-preemptive aspect of the scheduling reinforces further this independence between tasks as it eliminates any influence that might arise between them.

In general, much of the prior research on the scheduling of periodic task systems has focused upon preemptive scheduling, with the non-preemptive one receiving considerably less attention [23]. Even though most of the scheduling algorithms are preemptive, there exists situations where non-preemptive scheduling is preferable [87]. For example, the overhead generated by preemption is not always negligible with respect to task execution durations and inter-processor communication. Non-preemptive scheduling on a uniprocessor guarantees exclusive access to shared resources and data, thus eliminating the need for synchronization and associated overhead [114].

This thesis will only consider the non-preemptive scheduling of tasks for the reasons indicated in the beginning of this section.

2.2.5 Schedulability analysis

As was indicated in Section 2.1.1, an important concern in a critical real-time system is the predictability, that is if the system can meet all hard timing requirements. To ensure this predictability, schedulability analysis is performed.

For this study, the two kinds of schedules (in hard real-time systems) should be distinguished:

- **Feasible schedule:** it is a schedule where all system constraints are satisfied. That is to say, task executions are terminated before their respective deadlines, no overlap in execution time occurs, and all other schedulability constraints, such as memory requirements, are verified.
- **Infeasible schedule:** Contrary to the preceding definition, a schedule is referred to as infeasible if a system constraint is violated, such as missing a deadline in a hard real-time system.

Schedulability analysis for a real-time system consists of validating the presence of feasible schedules for a given real-time system, and/or verifying if a defined scheduling algorithm succeeds in providing such feasible schedules. Principal approaches that can be employed for this analysis are: analytical approach, simulation and model-checking.

2.2.5.1 Analytical approach

This approach for analysis consists of identifying the worst-case response times for tasks in a given system, and producing an analytical expression, a schedulability condition, permitting the evaluation of system characteristics. These conditions, generally polynomial or pseudo-polynomial time, can be sufficient conditions or sufficient and necessary ones depending on task parameters [126]. MAST [75] is an example tool implementing this approach.

2.2.5.2 Simulation

Simulation is widely used for performance analysis, dimensioning or development of real-time systems and schedulability analysis. It consists of modelling the system components with the aid of an adequate formalization, and then the execution using a simulation tool on predefined scenarios. The resulting information can represent simple execution traces of the system or even evaluation of certain system components. One of the main inconveniences of simulation is that it can be optimistic [150]. Although the simulation can show that the system is not schedulable by showing a non-feasible scenario, realising an exhaustive study of all possible scenarios, to prove the schedulability of a system, can be difficult [87].

Several simulation tools exist and are used in industry for simulation purposes, such as Cheddar [139].

2.2.5.3 Model checking

Model checking is an automatic technique for verifying finite-state reactive systems [39]. In other words, it is a method that allows an exhaustive exploration of the system state space. These methods allow verifying several system properties other than scheduling [74]. The system is modelled as a state transition graph or a Petri-Net graph. An efficient search algorithm is then used to determine whether or not the graphs satisfy system specifications. The major drawback, is that the state space that model-checking methods must take into account may be of exponential size. Consequently, this limits the size of analysable models.

In this thesis, instead of performing schedulability analysis, optimization algorithms are implemented to supply schedules that respect design constraints by construction. The optimization problem will be clearly defined in Chapter 3.

2.3 Embedded systems

An Embedded System is a processor based system that is embedded as a subsystem in a larger system (which may or may not be a computer system). It is designed to do some dedicated functions, often with real-time computing constraints.

Embedded systems necessitate more and more processors, which may be of different types, for their proper functioning. Some classifications for embedded architectures according to the nature of processors are:

- Homogenous: the processors are interchangeable and have the same processing capacity.
- Heterogenous: the processors are either not destined to execute the same tasks, or capable of doing so but with different processing capacities.

Constraints in multiprocessor systems, and more specifically embedded ones, are a direct consequence of resource restriction. These systems have a processing power and memory capacities, which are limited due to reasons related to weight, volume, energy consumption (autonomous vehicles) or even price (applications aimed for the public). Between these different constraints, memory management and energy consumption are the most studied in literature.

The IMA architecture introduced in Chapter 1 represents an embedded system where a set of homogeneous processing units are interconnected.

2.3.1 Memory management

With processors becoming more and more fast, memory performance advances are struggling to keep up the pace. Memory available for executing applications in embedded systems was hence always a precious resource. In fact, although lower prices and increased memory capacities are becoming common, memory requirements of applications continue to augment largely [137] as they become more and more complex. Wise management of memory becomes a delicate issue where rational allocation of memory to tasks should be always thought of.

Based on the memory distribution, some classifications for embedded multiprocessor systems are:

- Parallel: the set of processors communicate through a shared memory area.
- Distributed: the set of processors have memory distributed among them, and communicate only through message transmission.

In this thesis, the IMA's processing units are assigned independent memory resources. Hence, IMA architectures are clearly classified as distributed systems where the memory capacities of processors have to be respected.

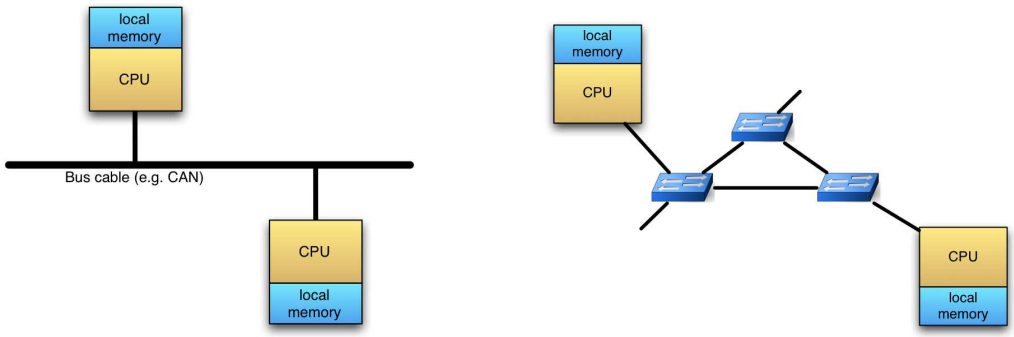
2.3.2 Distributed systems

A distributed system is a collection of processors interconnected by a communication network in which they coordinate only by passing messages [43]. In addition, loosely coupled distributed systems, are those in which the processors do not share memory. Each processor has its own local memory.

One advantage of distributed systems is the *independent failures* concept [53]. If a processor in a distributed system fails, it is not immediately made known to the other components in the system. Faults in this processor isolate it from the system, but the system does not stop running. Furthermore, each component in the network can fail independently, still leaving others running successfully.

The architecture of the interconnection network can follow one of the two commonly used:

- Bus based: In this type of architecture there is a single network and a bus cable that connects all components (processors). Figure 2.4a is an example of such architectures for a distributed system of processors, where each processor has its own local memory.
- Switch based: In the switch based architecture, components such as processors are interconnected through a set of switches. Communication between any two processors can be achieved following one or several possible routes. Figure 2.4b represents an example of such architectures. Another example is that of IMA architectures (*cf.* Figure 1.1b).



(a) Bus based. (b) Switch based.

Figure 2.4: Architectures for distributed systems with local memories.

In addition to the above mentioned architectures, the *complete topology*, where each processor pair are interconnected through a separate medium (bus cable), is also known. An example of such a topology is the federated architecture in avionics (*cf.* Figure 1.1a).

2.3.3 Energy consumption

Controlling energy consumption is a largely tackled problem in the embedded systems domain. With time it has become an essential factor for their design due to the complexity that gives rise to more

elevated energy consumption. Hence, energy consumption is an important parameter to take into consideration for developing embedded applications. There exists several approaches to relieve this problem which is not cited here for the sole reason that *this aspect of embedded systems is not taken into consideration in this thesis*.

2.3.4 Fault-tolerance

Fault-tolerance is the ability of a system to maintain its functionality, even in the presence of faults. It has been extensively studied in the literature [17, 99, 128, 67, 131, 45, 83]. Not all faults lead to immediate failure of the system, faults may be latent (activated but not apparent at the service level), and later becomes effective. Fault-tolerant systems attempt to detect and correct latent errors before they become effective. The means for fault-tolerance are either:

- error processing: to remove errors from the system's state, which can be carried out either with recovery (rolling back to a previous correct state) or with compensation (masking errors using the internal redundancy of the system);
- fault treatment: to prevent faults from being activated again, which is carried out in two steps, diagnostic (determining the cause, location and nature of the error) and then passivation (preventing the fault from being activated again).

These means use *redundancy* in order to treat errors, of which three forms exist: hardware redundancy (e.g. using a spare processor), software redundancy (e.g. using two implementations of the same application), and time redundancy (e.g. re-executing an application later).

Task and processor redundancies are considered to be predefined and included in the input for this thesis.

2.3.5 Other considerations

In addition to the above mentioned constraints, and temporal ones, there exists other types of constraints such as those related to resource accessing or even locality [51]. If the resource (processor, memory, etc.) is shared, then it is necessary to add constraints on tasks to avoid the use of a critical resource by several tasks at the same time. Elsewhere, it is possible to impose a *locality* constraint, signifying that a task must exclusively execute on a given processor, e.g. to execute on a processor that offers a resource not present on others. There also exists *anti-locality* constraints that precise that two given tasks must not execute on the same processor, as in the case of software redundancy, where the two redundant tasks must execute on different processors.

In this section, the most important constraints considered in later chapters are the locality and anti-locality of tasks on processors.

2.4 Complexity of scheduling problems

The purpose of complexity theory is the classification of computational problems according to their inherent difficulty. In an optimization theory context, each problem can be formulated as a decision

problem. A decision problem is a special type of computational problems whose answer is either yes or no, i.e. it is a selection process where one of two or more possible solutions is chosen to reach a desired goal in a given problem.

The O (big-O) notion is used to bound the complexity of an algorithm. For example, the algorithm to search an element in a list of size n has a complexity $O(n)$, which signifies that for finding this element, the number of elementary instructions (comparing two elements) to execute never exceeds n instructions.

The most known complexity classes for decision making problems are:

- The P class: it represents the class of problems that can be solved in polynomial time. In other words, there exists an algorithm of computational complexity $O(n^k)$ for a certain k. This class is called that of easy problems.
- The NP class: it is the abbreviation for "Non Deterministic Polynomial Time". This class contains all the problems for which a potential solution set (of exponential cardinality, at worst) can be associated, such that it can verified in polynomial time if a potential solution satisfies the posed question.

In addition, a problem is said to be NP-complete if it belongs to the NP class, and every other problem in the NP class can be converted (reduced) to it in polynomial time. Problems qualified as NP-hard are at least as hard as the hardest problems in NP but do not have to be in it.

From a computational complexity point of view, the scheduling problem can be addressed in different manners: the complexity of evaluating the validity (feasibility) of a given schedule towards system constraints; the complexity associated with the schedulability analysis of a system, i.e. if there exists feasible solutions to the problem; or even the complexity of evaluating the optimality of a scheduling policy can be differentiated.

These three problems are closely related. Consider the following example: Given an algorithm that allows deciding in polynomial time if a schedule is feasible, and in addition, given a scheduling policy A that is proved to be optimal for a class of systems B. A system of B can be hence proved schedulable in polynomial time by interpreting the validity of this system's scheduling by A, since the schedule provided by A is optimal and hence no better solution exists.

The complexity is not only related to the type of problems involved, but also to the considered systems and scheduling policies. Each of these classes implies the specificity of complexity estimation, which is a delicate process. Table 2.1 shows the complexities for some scheduling problems whose objective is to minimize the total completion time of tasks ($\sum_i e_i$).

2.5 Real-time scheduling algorithms

Among the various scheduling algorithms proposed for the scheduling of real-time tasks, distinguish the following approaches [71]:

Table 2.1: Complexity of some scheduling problems.

Environment	Characteristics	Complexity
Single machine	non-synchronous/preemptive	polynomial [19]
Single machine	precedence	NP-hard [102]
2 parallel machines	non-synchronous/unit-time/precedence	polynomial [21]
Parallel machines	non-synchronous/preemptive	NP-hard [32]
Unrelated parallel machines	preemptive	NP-hard [140]

- **Cyclic executive:** The scheduling algorithm prepares beforehand a table (this table is known as the cyclic schedule [20]) which determines when tasks are scheduled. The cyclic executive approach needs to store a table before the execution of the system; the size of this table is generally proportional to the least common multiple of all periods (of periodic tasks).
- **Static**: The scheduling algorithm computes the priorities of the tasks beforehand, based on the task characteristics (e.g., the worst case execution time, the deadline, the period, etc.). The priorities are then assigned to each task before the activation of all tasks. During the execution of the system, the system selects the highest priority task that is ready for execution.
- **Dynamic**: The scheduling algorithm computes the priorities during the execution of the system. The priority of each active execution is based on the system state such as the current time . In addition to the request characteristics, such as the remaining execution time of each request, or the time available before reaching the deadline.

The following sections shed light on the two major classes of scheduling algorithms, concerned with the uniprocessor and multiprocessor scheduling problems respectively.

2.5.1 Uniprocessor scheduling

One of the most important works on uniprocessor scheduling was that of Liu and Layland [110] in 1973, after which several solid results have been established. The literature in this domain is enriched and consists of optimal scheduling algorithms as well as performance analysis of schedules. The following sections discuss some well-known uniprocessor scheduling algorithms.

2.5.1.1 Rate Monotonic (RM)

Introduced by Liu and Layland in 1973 [110], the rate monotonic algorithm represents one of the most popular static priority rules. The rate monotonic scheduling is a preemptive scheduling with static priority that is applied to independent periodic task sets. It was defined for a sub-class of periodic task sets:

- Periodic tasks in a *synchronous* system, that is, all tasks are started at the same time ($r_1 = r_2 = \ldots = r_n$), and
- systems with *late deadline*, where the deadline coincides with the period ($d_i = T_i$).

The priorities are allocated to the different tasks in the following manner: the shorter the task period, the more priority it is allocated. The principal contribution of this algorithm is that it is optimal for the model proposed by Liu and Layland [110] (the tasks are preemptive, periodic and independent) from a scheduling point of view (if a feasible solution exists then the algorithm can find it).

The sufficient condition for schedulibility using the RM algorithm when applied to a system of n tasks is [110]:

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

where the left-hand side of the inequality represents the processor utilisation, and the right-hand side represents the utilisation bound.

Later on, a necessary and sufficient condition was introduced in [85]. The authors show that if a set of periodic tasks (classic periods), sorted by decreasing priority, can be scheduled by RM and the response time $R_i = (e_i - t_i)$ of task i is upper bounded by the solution of:

$$R_i = \left(\sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil \; b_j \right) + b_i,$$

where $\lceil x \rceil$ is the superior integer part of x, then the scheduling of this task set is feasible if and only if: $R_i \leq T_i$, $\forall i = 1 \ldots n$.

In the non-preemptive context, RM becomes non-optimal.

2.5.1.2 Deadline Monotonic (DM)

The deadline monotonic scheduler is an extension of the rate monotonic scheduler, and hence follows a static priority rule. It was introduced by Leung and Whitehead in 1982 [105] where the algorithm relaxes late deadline condition into the *general deadline*; namely it allows the deadline of a task to be less than the period ($d_i \leq Ti$). Hence, the deadline monotonic rule is defined for a larger sub-class of periodic task sets.

The smaller the deadline d_i, the more task i has priority. DM is optimal for Liu's and Layland's model [110] from a scheduling point of view.

The sufficient condition for scheduling n periodic tasks (sorted by decreasing priority) $\{1, \ldots, i, \ldots, n\}$ using DM is:

$$b_i + \sum_{j=1}^{i-1} \left\lceil \frac{d_i}{T_j} \right\rceil \; b_j \leq d_i, \quad \forall i = 1 \ldots n.$$

As is the case for RM, there exists a necessary and sufficient condition for the DM algorithm. If a set of periodic tasks, sorted by decreasing priority, is scheduled by DM such that the response time R_i

of task i is upper bounded by the solution of the equation:

$$R_i = \left(\sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_i} \right\rceil b_j \right) + b_i,$$

then the scheduling of this set is feasible if and only if: $R_i \leq d_i, \forall i = 1 \ldots n$.

DM becomes non-optimal in the non-preemptive case, except for the following case: $b_i \leq b_j$, $d_i \leq d_j$, $\forall (i, j)$.

2.5.1.3 Earliest Deadline First (EDF)

This popular algorithm was also introduced by Liu and Layland in 1973 [110]. It is a scheduling that can be preemptive or non-preemptive with dynamic priority, applied to periodic independent tasks with $d_i = T_i$. The idea for allocating priorities in this algorithm is that the task whose absolute deadline is the earliest to arrive has the highest priority. The priorities are re-evaluated, if necessary, over time. This re-evaluation is effected when, for example, a new task arrives and its deadline is the nearest as compared to other ready tasks. EDF is optimal for preemptive uniprocessor scheduling with dynamic priority for independent periodic tasks whose deadlines are equivalent to their periods. It was later shown that this algorithm remains optimal if the tasks are non-periodic.

The necessary and sufficient condition for schedulability in the preemptive case (if $d_i = T_i$, $\forall i$) is:

$$\sum_{i=0}^{n} \frac{b_i}{T_i} \leq 1.$$

2.5.1.4 Least-Laxity First (LLF)

This algorithm is based on the laxity of tasks. LLF was first introduced by Mok and Dertouzos [119, 118]. At each invocation, LLF elects the task whose laxity is the smallest. [41] shows that the conditions for schedulability for the LLF algorithm are the same as those for EDF, this means that the necessary and sufficient condition in the preemptive case (with $d_i = T_i$, $\forall i$) is:

$$\sum_{i=0}^{n} \frac{b_i}{T_i} \leq 1.$$

The LLF algorithm becomes inconvenient when several tasks have the same laxity, in which case a large number of changes is generated. This explains why it is rarely used in the uniprocessor case.

2.5.2 Multiprocessor scheduling

The multiprocessor scheduling problem was formulated for the first time by Liu in 1969 [109]. Most problems related to hard real-time scheduling on multiprocessor systems under non-trivial assumptions have been proven to be NP-complete [65, 66, 145, 151].

When dealing with multiprocessor scheduling, first results that can be obtained are the absence of optimal scheduling algorithms having a polynomial complexity, and that solutions to multiprocessor scheduling problems are certainly not trivial extensions to mono-processor ones [87].

Traditionally, there have been two major approaches for the multiprocessor task scheduling [35]: partitioning and global scheduling.

2.5.2.1 Partitioning

In this approach, each task is assigned to a single processor where all of its instances are executed, without migration possibilities. In other words, the n task set is partitioned into m subsets, with m being the number of processors. Each of these subsets is then scheduled on one processor. The advantage of this approach is that it transforms the multiprocessor scheduling problem into a set of uniprocessor ones [35]. The main drawback is that finding an optimal assignment to processors is similar to a bin-packing problem, which is NP-hard in the strong sense [66].

In spite of this, partitioning is widely used and appears in two variations [18]: Minimizing the number of processors needed to guarantee the feasibility of the task set, or alternatively, given a fixed multiprocessor platform, finding sufficient schedulability bounds, such as utilization. Due to intractability, many heuristics for partitioning have been proposed, most of which are bin-packing-based algorithms and can be applied in polynomial times using sufficient schedulability tests [15].

2.5.2.2 Global scheduling

In global scheduling, a task is allowed to execute on any processor, even when resuming after having been preempted, and thus unlike partitioning, migration is allowed.

Producing a conventional global scheduler suitable for real-time systems has proved to be a daunting task [77], mainly because of the following limitations:

- No efficient schedulability tests currently exist for this approach [15].
- The use of optimal uniprocessor scheduling algorithms are not optimal in this the multiprocessor context. The RM algorithm for example leads to low processor utilization [35].

The primary advantage of global scheduling is that optimal scheduling is possible since migration is permitted [77].

Nevertheless, *proportionate fair* (or Pfair) scheduling [25] is a global-scheduling approach proposed as a means for optimally scheduling periodic tasks on a multiprocessor when each task's deadline equals its period. It is capable of optimally scheduling recurrent and rate-based tasks on a multiprocessor [77]. It is also the only known optimal multiprocessor scheduling algorithm with polynomial complexity.

2.5.2.3 Some advances in periodic multiprocessor scheduling

Apart from the methods indicated above, interest in periodic multiprocessor scheduling has become more profound in past years. Several uniprocessor algorithms were extended to their multiprocessor counterparts [47, 111, 112]. Other, superior ones, were also proposed, such as in the work of Goossens *et al*. [72] who have proposed a new priority-driven scheduling algorithm for periodic tasks in multiprocessor systems, and that appeared to be superior to EDF in the sense that it schedules all tasks that can be scheduled by EDF, in addition to some of those that cannot. Feasibility analysis and schedulability conditions where proposed for the non-preemptive EDF (multiprocessor) by Baruah [23], and later on for EDF with migrating general task models by Ficher and Baruah [60].

In this thesis, the multiprocessor partitioning problem is considered.

2.5.3 Non-preemptive and strictly periodic multiprocessor scheduling

A more specific multiprocessor scheduling problem is the one with non-preemptive task sets of strict periodicity. Such a problem was not much tackled in literature until the last couple of years, where contributions began to arise. Not only does the non-preemption aspect add computational complexity [24] on the problem, but also the strict periodicity makes its even harder. In most of the studies, loose periodicity was considered, where some slack time is allowed between successive executions of a periodic task [93]. One of the early works on strictly periodic scheduling is that of Korst [91] who was motivated by real-time video signal processing. He considers the problem of non-preemptively scheduling periodic tasks on a minimum number of processors, assuming that the tasks have to be executed strictly periodically, and proposes a greedy heuristic to solve this problem. Korst also establishes a necessary and sufficient condition for the schedulability of two periodic tasks. Korst *et al*. show that the problem is NP-complete in the strong sense, even in the case of a single processor, but that it is solvable in polynomial time if the periods and execution times are divisible [92]. They also propose an approximation algorithm, which is based on successively assigning tasks to processors according to some priority rule.

Kermia and Sorel [88] propose a heuristic for scheduling non-preemptive and independent task sets with strict periodicity upon multiprocessors. Their objective is to minimize the schedules' cycle times while respecting latency and precedence conditions. Their heuristic is composed of several algorithms, the first being the assignment of tasks to processors, and after which the scheduling is carried out. Their assignment favours placing tasks whose periods are equal or multiples on the same processor. Comparison with an exact branch-and-cut algorithm shows the effectiveness of the proposed heuristic. Meumeu and Sorel [116] and later Marouf and Sorel [114] present schedulablity conditions in a similar context.

In [55], Eisenbrand *et al*. consider the problem of scheduling strictly periodic tasks on a minimum number of processors. They show that if the periods are harmonic, then there exists a 2-approximation for this minimization problem and that this result is tight. In [56], the authors handle the same problem with additional constraints in an avionic context. Assuming harmonic periods, they propose an Integer Linear Programming (ILP) formulation and primal heuristics that together solve real-world instances to optimality.

Task periodicity being set aside, it should be noted that commonly found scheduling problems have the objective of minimizing the makespan [108]. On one processor, this makespan represents the time difference between the start time of the first executing task and the end time of the last executing task. This problem was largely treated in literature [97].

As can be seen above, not many works exist for the mentioned scheduling problem. Several useful methods were lately proposed and were shown to be highly efficient. However, they consider only harmonic periods or even favour grouping harmonic ones on the same processor. This thesis aims at raising this difficulty by proposing algorithms adapted to all kinds of periods, harmonic and non-harmonic alike.

2.6 Theoretic concepts for the thesis

After a brief introduction on various real-time scheduling problems, the discussion in this section is limited to the non-preemptive multiprocessor scheduling problem with strictly periodic task sets. This particular problem constitutes the main subject of this thesis. The goal is to find an algorithm able to solve this problem in an avionics context where several system constraints are imposed (*cf.* Chapter 1). Indeed, task periodicities in major industrial applications have harmonic periods, or even a limited set of non-harmonic ones. In spite of this fact, this thesis seeks surpassing this limitation for which several algorithms where proposed (*cf.* Section 2.5.2.3). The goal is hence the conception of an effective algorithm for general period task systems.

2.6.1 Particularities of the study

The scheduling problem to be considered has to not only satisfy the runtime support constraints of the system, e.g. respecting temporal requirements, but also architectural constraints derived from several functional requirements such as the locality of tasks on processors, where one task may be prohibited from executing on certain processors.

A particularity of this study, and as will be discussed in Chapter 3, is that the optimization objective to be considered is not classic, e.g. minimize number of processors or makespan, but an ambitious one aimed at providing an augmented evolution margin for task execution durations.

As was mentioned in Section 2.5.2.3, this type of scheduling problems are not much studied in literature [88] where few contributions actually exist. Main reasons for so are the following:

- The multitude of the associated constraints in the system poses a significant complexity on the scheduling problem.
- The strict periodicity is a particular case of the classic periodicity constraint that can be found in literature. Recall that a period is classified strict if for a task i of period T_i, $t_i^{k+1} - t_i^k = T_i$, $\forall k \in \mathbb{N}$, where t_i^{k+1} and t_i^k are the execution times of two consecutive instances (invocations) of task i. The presence of this constraint in the system restrains its schedulability chances [116] and renders it NP-hard in the strong sense [46].

It should be noted that the problem, even in a uniprocessor setting, remains a complicated one to solve [68].

Theoretic methods, implemented in following chapters for the development of adapted solution strategies, are hereafter discussed for the aforementioned scheduling problem.

2.6.2 Some known solution strategies

Existing algorithms are categorized into:

- Exact/Optimal: In optimization problems, these algorithms supply the best possible solutions. If no solution can be supplied by these algorithms, then none exist. Techniques that allow avoiding zones that do not contain adequate solutions exist and help reducing computation times.
- Approximation/Sub-optimal: These algorithms are used to give approximate solutions in polynomial times, especially in NP-hard problems where exact methods can be restrictive and have exponential complexities [153].

The problem addressed is, as many of scheduling problems with imposed constraints, NP-hard in the strong sense [87]. This is why these algorithms are constantly ameliorated in order to solve problems larger than preceding ones. Each algorithm is distinguished by the manner it explores the solution space.

2.6.2.1 Linear programming

A linear programming problem [59] can be defined as the problem of *maximizing* or *minimizing* a *linear objective function subject to linear constraints*. The constraints may be equalities or inequalities. It was introduced by Kantorovich [86] who started working on the subject in 1939, important resolution methods are the well known Simplex method for solving linear programs which was introduced by Dantzig [48], and the duality theorem [117]. Given a polytope and a real-valued affine function defined on this polytope, a linear programming method will find a point on the polytope where this function has the smallest or largest value if such point exists, by searching through the polytope vertices. Furthermore, linear programming algorithms are known to be exact.

Linear programs are problems that can be expressed in canonical form:

$$\text{Maximize:} \quad \mathbf{c}^T \mathbf{x} \tag{2.1}$$

$$\text{subject to:} \quad A\,\mathbf{x} \leq \mathbf{b}, \tag{2.2}$$

$$\mathbf{x} \in \mathcal{X}, \tag{2.3}$$

where $\mathbf{x}$ represents a vector of m unknown variables specified by the domain $\mathcal{X}$, $\mathbf{c}$ and $\mathbf{b}$ are vectors of known coefficients and A is a known matrix of coefficients. The expression (2.1) represents the objective function to be optimized, and constraints (2.2) and (2.3) specify the convex polytope over which the objective function is to be optimized.

If the unknown variables are all real, i.e. $\mathcal{X} = \mathbb{R}^m$, then the problem can be solved using the Simplex method.

If the unknown variables are all required to be integers, i.e. $\mathcal{X} = \mathbb{Z}^m$, the linear programming problem is called an integer linear programming one (ILP), if they are required to be binaries, the terminology becomes binary integer programming (BIP) (which is a special case of the integer one). In addition, if the unknown variables are a mixture of real, integer and even binary variables, then the problem is called a mixed integer programming (MIP) or a mixed integer linear programming (MILP) one. Algorithms for solving these problems include methods such as branch-and-bound [98], branch-and-cut [125, 76], branch-and-price [135, 22] and cutting-plane [70, 113].

Davidovic et al. [49] present a MILP formulation for non-periodic task scheduling (with dependencies) in a homogeneous multiprocessor environment with latency conditions. For reducing the number of constraints, they use a problem reformulation with the aid of a constraint reduction procedure. They solved several small-sized problems using the linear program solver CPLEX [81]. Thanikesavan et al. [147, 148, 149] present a MILP formulation to solve a multiprocessor periodic task scheduling problem, but with no emphasis on strict periodicity and considering task instances (on a time horizon equivalent to the LCM of all partition periods). Eisenbrand *et al*. [56] propose an efficient ILP formulation for scheduling large problems with harmonic periodic task sets.

2.6.2.2 Greedy algorithms

Greedy algorithms are heuristics that are simple and straightforward. They are short-sighted in their approach in the sense that they take decisions on the basis of information at hand without worrying about the effect these decisions may have in the future. Their main idea is incremental solution construction where a new element is added at each step. In a multiprocessor task assignment for example, tasks may be ordered in a certain manner, then assigned to processors one by one such that each task performs a *greedy choice* in which it chooses the assignment that best suites it.

If this short-sighted approach, where no *back-tracking* is allowed, always gives optimal solutions, then it is called an exact greedy algorithm, else it is called a greedy heuristic. Kruskal's algorithm [96] for finding minimum spanning trees in graphs is an example of exact greedy algorithms.

To ensure that a greedy algorithm gives an optimal solution for a problem, the following properties have to be shown:

- **The greedy choice property:** a globally optimal solution can be found by performing locally optimal choices (greedy choices).
- **The optimal sub-structure property:** after performing a greedy step (choice), the resulting subproblem (remaining steps) represents a reduced form of the initial problem. In other words, if the first choice is eliminated from an optimal solution of the initial problem, an optimal solution is obtained for the reduced problem.

The *First-Fit* (based on bin-packing) approximation algorithm presented in [55], represents a greedy algorithm for scheduling strictly periodic harmonic tasks on a minimum number of processors. Tasks are ordered based on their periods and then placed, following their order, on processors with appropriate bins. Bins on a processor are defined from the smallest task period resident on this processor.

This algorithm is classified as a heuristic as it gives in most cases an upper bound on the number of processors.

2.6.2.3 Local Search heuristics

Local Search algorithms represent a very old heuristic class. They are approximation techniques mostly used for solving hard combinatorial optimization problems. A local search heuristic is an iterative process based on two essential elements: a neighbourhood structure $\mathcal{N}$ and a neighbourhood exploration procedure. The main idea is to iteratively ameliorate the current solution x^i at iteration i by exploring its neighbourhood $\mathcal{N}(x^i)$ for a less costly one. Neighbourhood solutions are generally obtained by applying an elementary transformation on the current solution.

Local searches are hence achieved by iteratively replacing present solutions with better ones, until no more better solutions can be found. There are several manners for picking a neighbouring solution. The *first descent* technique chooses the first ameliorating solution it encounters. Whereas the *steepest descent* explores all neighbouring solutions and picks the best one. This latter technique appears to be more costly in the sense of neighbourhood exploration, but nevertheless provides a better quality for the algorithm. Algorithm 4 describes the steepest-descent local search for a cost minimization problem. $F(x)$ represents the utility function for solution (configuration) x, that is it represents the cost associated to this solution and allows defining a criterion for choosing neighbours. As long as the best found neighbour z of x is ameliorating, the local search continues with additional iterations. If not, it stops and a local minimum is attained (local maximum in maximization problems).

Algorithm 4 Steepest-descent local search algorithm

```
1: procedure Local_Search
2: choose initial point x ← x^0
3: repeat
4:    z ← x
5:    for y ∈ N(x) do
6:       if F(y) < F(z) then
7:          z ← y
8:       end if
9:    end for
10: until z = x
```

The principal advantage of this method is that, based on the size of the neighbourhood and the neighbour evaluation cost, it is rather simple and rapid in most cases. However, produced solutions are not guaranteed to be near optimal. This can be remedied by performing *multi-start* where several local search runs are performed with different initial points x^0, that are picked randomly. In this manner several local points can be detected. Another possibility is to accept neighbours with the same cost if no better can be found. Other techniques such as the variable neighbourhood search [107] seek escaping from local minima (or maxima).

2.6.3 Game theory

Game theory [120, 63, 123] can be defined as the study of mathematical models of conflict and cooperation between intelligent rational decision-makers. It provides mathematical techniques for analysing situations in which two or more individuals make decisions that will influence one another's welfare. In other words, it uses mathematics to help understanding observed phenomena when decision-makers interact. A decision-maker is called *rational* if he makes decisions consistently in pursuit of his own objectives, that is to say he is aware of his alternatives, forms expectations about any unknowns, has clear preferences, and chooses his *action* deliberately after some process of optimization. Game theory is vast, and hence this section is limited to aspects and particularities interesting for this thesis.

2.6.3.1 Non-cooperative games

In the language of game theory, a *game* is a description of strategic interaction between two or more individuals. Individuals are often referred to as players. This game includes the constraints on the actions players can take in addition to the players' interest.

One main branch of game theory is the non-cooperative game in which players make decisions independently according to their own objectives. In such type of games, consider a finite set $N = \{1, \ldots, n\}$ of players, and a set of possible actions A_i for each player $i \in N$. A *pure strategy* for player i is an action from A_i. A *mixed strategy* corresponds to a probability function which prescribes a randomized rule for selecting an action from A_i. S_i denotes the set of strategies available for player i.

A strategy profile $s = (s_1, \ldots, s_n)$ assigns a strategy $s_i \in S_i$ to each player i. A *payoff* function $F_i(s)$ is also associated to each player. It specifies the payoff received by player i if the strategy profile s is adopted by the players. Denote by s_{-i} the profile for the set of players $N \setminus \{i\}$.

A strategy s_i^* is said to be a best response for player i against the profile s_{-i} if

$$s_i^* \in \underset{s_i \in S_i}{\operatorname{argmax}} F_i(s_i, s_{-i}).$$

2.6.3.2 Nash equilibrium

The most commonly used solution concept in game theory is that of *Nash equilibrium* [121]. A Nash equilibrium is a state in which no player has an incentive to unilaterally change his strategy. Briefly, no player can profitably deviate, given the actions of the other players [123].

Formally speaking, a strategy profile s^e is an equilibrium profile if for every $i \in N$, s_i^e is a best-response for player i against s_{-i}^e, that is to say,

$$s_i^e \in \underset{s_i \in S_i}{\operatorname{argmax}} F_i(s_i, s_{-i}^e), \quad i \in N.$$

The application of game theoretic approaches with the aforementioned principles can be found in machine scheduling contexts such as [38] and [54]. In [1], the authors use similar game theoretic approaches for a multi-objective task mapping/scheduling problem, where parallel tasks are distributed upon a multi-core architecture for makespan and power consumption minimization.

2.7 Conclusion

In this chapter several definitions and classifications of scheduling problems were discussed. As was already indicated, the work in this thesis is closely related to the non-preemptive multiprocessor scheduling for strictly periodic tasks. Among all of the contributions on scheduling, relatively few works have addressed this type of problems. Despite of the various efforts, it remains not well resolved. In this thesis, methods for solving such problems are investigated, in a general setting where no constraints are imposed on the type of strict periods (harmonic or non-harmonic). In Chapter 3, an exact MILP based formulation is proposed and tested. A game theoretic approach is then investigated in Chapter 4. In addition, although not much was discussed concerning flow routing in networks, Chapter 5 focuses on the Virtual Link routing problem in AFDX networks.

CHAPTER 3

MILP formulation of the scheduling problem

In this chapter, an exact linear programming algorithm is investigated for the scheduling problem introduced in Chapter 1. Several system constraints are taken into account such as strict periodicity and resource capacities, in addition to domain specific conditions (such as locality constraints). In the avionic context, few works have tackled automation in decision making for what concerns partition scheduling on the avionic platform [62]. Models for allocating partitions to modules were studied in [133, 29] based on directives derived from safety and operational reliability requirements, however, temporal scheduling of partitions on each module was not addressed. Recently, the authors in [56] tackled the scheduling problem and effectively solved the processor minimization to optimality, using similar methods (ILP), but for harmonic period cases. In this Chapter, a mixed integer linear programming formalization (*cf.* Section 2.6.2.1) is presented for the multiprocessor scheduling with strict periodicities. Additionally, a new optimization objective, different from those found in literature (such as minimizing the number of processors), is proposed.

Definitions and models to the problem are given in Section 3.1. A mathematical formulation based on mixed integer linear programming is then proposed in Section 3.2, where the optimization objective is clearly indicated. In Section 3.3, a pretreatment method based on graph theory, to reduce the solution space, is introduced. Section 3.4 demonstrates some results on the proposed model. Section 3.5 finally concludes on this chapter's methodology.

3.1 Problem definitions and modeling

It was seen in Chapter 2 the notion of tasks and their respective scheduling in a multiprocessor system. From this point on, in the context discussed in Chapter 1, the task and processor notions are replaced by partition and module respectively. The functional attributes between tasks and partitions remain equivalent, as a partition represents a set of instructions to achieve a certain functionality. The equivalence also applies between processors and modules, as a module is a processing unit, with attributed

resources, destined to executing the system partitions.

In airborne systems, communication occurs in the framework of processing chains through which some kind of data is treated sequentially by a certain number of partitions. Data usually originates from a sensor or user input, and after processing by one or more partitions, a command is send to an actuator or screen (e.g. displaying altitude after analysing altimeter readings). For ease of presentation, it shall be assumed that a processing chain starts at the first partition of the chain and ends at the last one. Figure 3.1 shows a simple example of a chain where partition 1 sends some kind of data to partition 2 which itself manipulates and transmits it back to partition 1. This partition finally sends a command or result to partition 3, the final consumer in the chain.

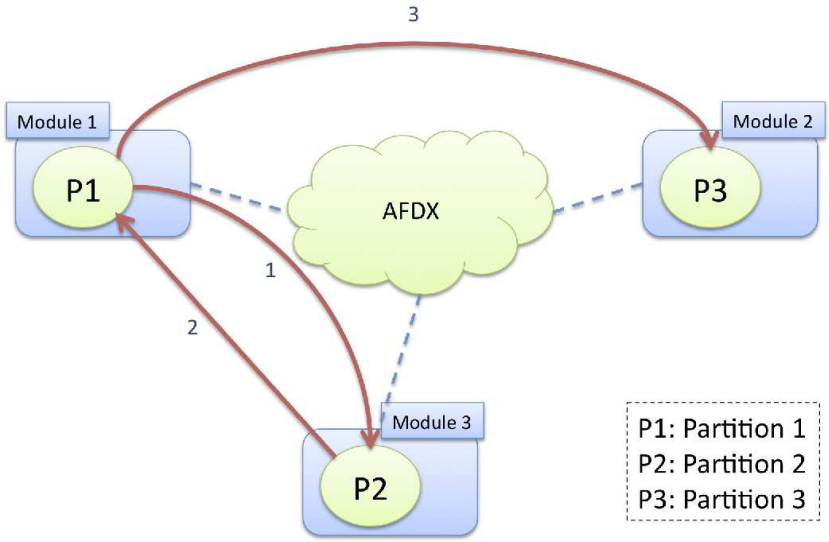


Figure 3.1: Processing chain consisting of 3 partitions.

The partition scheduling problem consists of a set $\Pi = \{1, \ldots, N\}$ of N partitions that have to be mapped and scheduled (temporally) on a set $\mathcal{P} = \{1, \ldots, P\}$ of P processing modules (distributed processor architecture). In sections 3.1.1, 3.1.2 and 3.1.3, mathematical models for the partitions, modules and communication chains are supplied.

3.1.1 Module model

Each module $k \in \mathcal{P}$ is characterized by the available memory capacity M_k, and the maximum number of partitions H_k the module can host. Inter-module communication is characterized by the delay matrix $\Delta = \delta_{k,l}, \forall (k,l) \in \mathcal{P}^2$, where $\delta_{k,l}$ represents the maximum communication delay between modules k and l for $k \neq l$. It is assumed that $\delta_{k,k} = 0$. This inter-module transmission delay is considered to be precomputed (estimated) from network analysis phases.

For safety requirements, the modules are arranged into cabinets, $\mathcal{C} = \{1, \ldots, C\}$, representing groups of modules sharing communication means.

3.1.2 Partition model

A partition $i \in \Pi$ has the following attributes:

- $T_i \in \mathbb{N}$, the partition period (strict),
- b_i, the time budget of the partition, i.e. the duration of partition execution (WCET in most cases),
- m_i, the memory budget of the partition, i.e. required memory capacity.

Let $\mathcal{T}_i = \{0, 1, \ldots, T_i - 1\}$ denote the possible offsets for t_i. Due to strict periodicity, the k^{th} invocation (instance) of the partition is executed at time $t_i^k = t_i + kT_i$. It should be noted that in this thesis partition deadlines are equivalent to their periods. Hence, ensuring a strictly periodic schedule for partitions, and given the preceding limitations of the partition offsets ($t_i \in \mathcal{T}_i$), will suffice for guaranteeing the deadline constraints.

Two given partitions may be in exclusion for security reasons, i.e. they cannot be hosted by the same module. Let $\mathcal{E}$ denote the set of couples $(i, j) \in \Pi^2$ such that partitions i and j must be executed on two different modules. Similarly, two partitions can be in exclusion on cabinet level, meaning that they cannot co-exist in the same cabinet, and let $\mathcal{E}_c$ denote the set of couples $(i, j) \in \Pi^2$ such that partitions i and j must be executed on two modules from different cabinets.

3.1.3 Communication model

To achieve a certain functionality, certain data may be exchanged between several partitions. This exchange is carried out linearly, in the form of a processing chain, from one partition to the other. Based on the criticality of this data manipulation, a delay constraint may be imposed, and thus limiting the latency to a predefined limit. Figure 3.2 represents an example where the pilot issues a zoom request on one of the displays. In this example, the response time for the operation should not exceed 100 milliseconds.

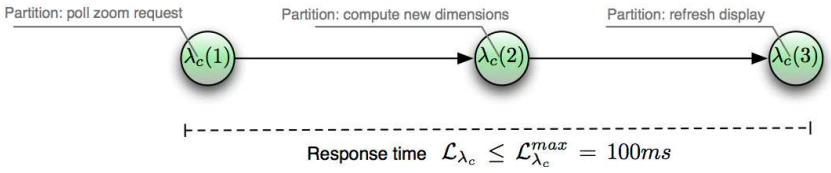


Figure 3.2: A processing chain λ_c aimed at responding to a screen zoom request.

Let $\Lambda = \{\lambda_1, \ldots, \lambda_o\}$ be the set of all processing chains in the system. These chains are independent, meaning that there is no communication between any two chains. Each processing chain λ_c is a sequence of partitions through which data is transferred, and $\lambda_c(k)$ represents the k^{th} partition to

handle the data in the chain. A processing chain is characterized by the maximum tolerated end-to-end delay $\mathcal{L}_{\lambda_c}^{max}$ after which critical (or even hazardous) situations may be encountered.

3.2 Problem formulation

This section gives a mathematical formulation for the scheduling problem defined above.

The allocation problem amounts to finding a function that associates a module to each partition, such that all imposed constraints are verified for all modules. Based on a given allocation, all instances of a partition must execute on the same module. An allocation can be represented as a vector of binary variables $\mathbf{a} = (a_{i,k})$ such that:

$$a_{i,k} = \begin{cases} 1 & \text{if partition } i \text{ is assigned to module } k, \\ 0 & \text{otherwise.} \end{cases}$$

Since partitions execute strictly periodically, a schedule is entirely defined by first partition execution dates which is represented by the offset vector $\mathbf{t} = (t_1, \ldots, t_N)$.

As indicated above, a first allocation constraint concerns mapping each partition $i \in \Pi$ onto one and only one module, hence:

$$\sum_{k \in \mathcal{P}} a_{i,k} = 1 \quad , \forall i \in \Pi. \tag{3.1}$$

3.2.1 Temporal scheduling constraints

Note that the mth instance of partition i is executed in the time interval

$$I_m(t_i) = [t_i + mT_i,\, t_i + mT_i + b_i)\,.$$

To avoid overlap in execution times between two partitions i and j allocated to the same module, any two invocations m and n of these partitions must not overlap in time. This can be expressed as follows:

$$\forall (i,j) \in \Pi^2, \forall m,n \in \mathbb{N}^*, \forall k \in \mathcal{P}$$
$$a_{ik} = a_{jk} = 1 \Rightarrow I_m(t_i) \cap I_n(t_j) = \emptyset.$$

Korst proposed in [91] (p.65) a necessary and sufficient condition to ensure that two partitions do not overlap in time. This condition is represented in Section 3.2.1.1.

3.2.1.1 Schedulability condition for strictly periodic partitions

The theory behind the sufficient and necessary schedulability condition proposed in [91] is hereafter presented.

Theorem 1 (Bachet-Bézout identity) *The Bachet-Bézout identity [84] proves that if a and b are two non-zero integers ($a, b \in \mathbb{Z}$), then there exists integers x and y such that*

$$ax + by = gcd(a, b)$$

where $gcd(a, b)$ is the Greatest Common Divisor of a and b.

Definition 3.1 *The floor function of a decimal value $x \in \mathbb{R}$ is defined as $\lfloor x \rfloor \in \mathbb{Z}$ such that,*

$$x - 1 < \lfloor x \rfloor \leq x.$$

Definition 3.2 *Given two numbers, a (the dividend) and b (the divisor), a modulo b (abbreviated as a mod b) is the remainder r, on division of a by b. As will be seen later, given that the divisor b will always be positive, the several definitions on the modulo operation [31] (depending on type of division used, such as Euclidean) yield the same results on the remainder, in other words:*

$$a \mod b = a - b \left\lfloor \frac{a}{b} \right\rfloor = r \tag{3.2}$$

with r following the positive sign of b and satisfying $0 \leq r < |b|$.

In the rest of this thesis, the symbol $\%$ is used as a shorthand notation for the modulo operator, i.e. $a\%b$ is to be read $a \mod b$.

Denote by $g_{i,j}$ the greatest common divisor of T_i and T_j, the periods of partitions i and j respectively (Remark that $g_{i,j} = g_{j,i}$). Consequently, $T_i = n_i\, g_{i,j}$ and $T_j = n_j\, g_{i,j}$.

Using the definition of modulo (3.2):

$$(t_j - t_i)\%g_{i,j} = (t_j - t_i) - q_{j,i}g_{i,j},$$

where t_i and t_j are the start times for partitions i and j respectively. The quotient in the division between $(t_j - t_i)$ and $g_{i,j}$ is represented by $q_{j,i}$, that is,

$$q_{j,i} = \left\lfloor \frac{t_j - t_i}{g_{i,j}} \right\rfloor.$$

Lemma 2 *For any two partitions i and j executing on the same module,*

$$\min_{k,l \in \mathbb{Z}} |(t_j + lT_j) - (t_i + kT_i)| = \min [\ (t_j - t_i)\%g_{i,j}\ ,\ (t_i - t_j)\%g_{i,j}] \tag{3.3}$$

where $k, l \in \mathbb{Z}$ represent execution instances for the two partitions.

Proof. Assume that $t_j + lT_j \geq t_i + kT_i$. The distance between the two executions is then

$$\begin{aligned}(t_j + lT_j) - (t_i + kT_i) &= (t_j - t_i)\%g_{i,j} + q_{j,i}g_{i,j} + lT_j - kT_i \\ &= (t_j - t_i)\%g_{i,j} + [q_{j,i} + ln_j - kn_i]\ g_{i,j}.\end{aligned}$$

Since $0 \leq (t_j - t_i)\%g_{i,j} < g_{i,j}$, the condition $t_j + lT_j \geq t_i + kT_i$ implies that $q_{j,i} + ln_j - kn_i \geq 0$ and thus that

$$(t_j + lT_j) - (t_i + kT_i) \geq (t_j - t_i)\%g_{i,j}. \tag{3.4}$$

According to Theorem 1, there exist $k, l \in \mathbb{Z}$ such that $q_{j,i}g_{i,j} = kT_i - lT_j$ and thus the above inequality can be satisfied as an equality. The proof in the case $t_j + lT_j < t_i + kT_i$ is symmetric. ■

In fact, Lemma 2 indicates that the minimal distance between the executions of partitions i and j is $\min[(t_j - t_i)\%g_{i,j}, (t_i - t_j)\%g_{i,j}]$. The term $(t_j - t_i)\%g_{i,j}$ represents the time duration partition i is allowed to execute without interfering with partition j's execution. Similarly, $(t_i - t_j)\%g_{i,j}$ represents the execution time partition j is allowed to occupy without interfering with partition i's execution. This is showed in Figure 3.3 for a two partition example (i and j) with $T_j = 2\,T_i$.

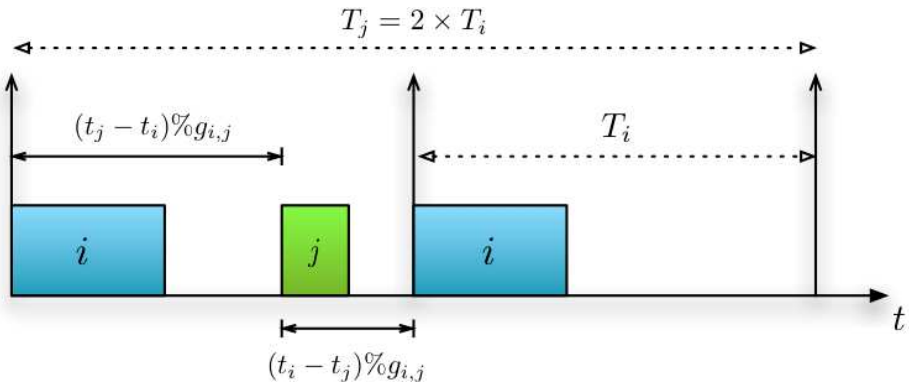


Figure 3.3: Representation of the distance between two partitions.

Theorem 3 *For two partitions i and j, the intersection between intervals $[t_i + kT_i, t_i + kT_i + b_i)$ and $[t_j + lT_j, t_j + lT_j + b_j)$ is empty, $\forall (k, l) \in \mathbb{Z}^2$, or in other words the two partition executions never overlap in time on the same module, if and only if*

$$b_i \leq (t_j - t_i)\%g_{i,j} \leq g_{i,j} - b_j. \tag{3.5}$$

That is, $b_i \leq (t_j - t_i)\%g_{i,j}$ and $b_j \leq (t_i - t_j)\%g_{i,j}$.

Proof. Let $k, l \in \mathbb{Z}$ such that $t_j + lT_j \geq t_i + kT_i$. From (3.4), it can be concluded that a necessary and sufficient condition for $I_k(t_i) \cap I_l(t_j) = \emptyset, \forall k, l$ is $(t_j - t_i)\%g_{i,j} \geq b_i$.

Similarly, if $t_j + lT_j \leq t_i + kT_i$, a necessary and sufficient condition for $I_k^i(t_i) \cap I_l^j(t_j) = \emptyset, \forall k, l$ is $(t_i - t_j)\%g_{i,j} \geq b_j > 0$. This condition can be transformed to $(t_j - t_i)\%g_{i,j} \leq g_{i,j} - b_j$ using the following,

$$(-a)\%b = b - a\%b \quad \text{for } a\%b > 0. \tag{3.6}$$

Condition (3.5) is obtained and the proof is concluded. ■

Corollary 4 *Ordering between i and j has no effect in equation* (3.5)*. Hence,*

$$b_i \leq (t_j - t_i)\%g_{i,j} \leq g_{i,j} - b_j \Leftrightarrow b_j \leq (t_i - t_j)\%g_{i,j} \leq g_{i,j} - b_i$$

Proof. Again following (3.6),

$$\begin{aligned} b_i \leq (t_j - t_i)\%g_{i,j} \leq g_{i,j} - b_j &\Leftrightarrow b_i \leq (-(t_i - t_j))\%g_{i,j} \leq g_{i,j} - b_j \\ &\Leftrightarrow b_i \leq g_{i,j} - (t_i - t_j)\%g_{i,j} \leq g_{i,j} - b_j \\ &\Leftrightarrow b_j \leq (t_i - t_j)\%g_{i,j} \leq g_{i,j} - b_i. \end{aligned}$$

■

Theorem 3 assists in validating a schedule, i.e. there is no overlapping in execution times between partitions on modules, if and only if each couple of partitions (on a module) verify condition (3.5). This condition between partitions i and j can be equivalently written as the following linear constraints:

$$\begin{aligned} &\forall (i,j) \in \Pi^2, \forall k \in \mathcal{P}, \\ &b_i - (2 - a_{i,k} - a_{j,k})\, Z \leq (t_j - t_i) - q_{j,i}\, g_{i,j} \leq g_{i,j} - b_j + (2 - a_{i,k} - a_{j,k})\, Z, \end{aligned} \tag{3.7}$$

where $q_{j,i}$ is as indicated earlier the integer variable representing the quotient from the modulo operation in (3.5) (i.e. $q_{j,i} = \left\lfloor \frac{t_j - t_i}{g_{i,j}} \right\rfloor$) and Z is a large constant that ensures that the constraint is not active unless $a_{i,k} = a_{j,k} = 1$. That is to say that the schedulability constraint is not important unless the two partitions execute on the same module. The introduction of Z to obtain conditional constraints was inspired from [130].

To properly linearise the problem, the quotient $q_{j,i}$ must verify the following constraint for every partition couple allocated the same module,

$$0 < (t_j - t_i) - q_{j,i}\, g_{i,j} < g_{i,j}. \tag{3.8}$$

Clearly, given that b_i and b_j are positive, verifying constraints (3.7) imply the validity of (3.8). Hence, (3.8) represent redundant constraints.

From Definition 3.1

$$\frac{t_j - t_i}{g_{i,j}} - 1 < \left\lfloor \frac{t_j - t_i}{g_{i,j}} \right\rfloor \leq \frac{t_j - t_i}{g_{i,j}},$$

and hence a bound for $q_{j,i}$ can be supplied, which is:

$$\frac{-T_i + b_i}{g_{i,j}} - 1 < q_{j,i} \leq \frac{T_j - b_j}{g_{i,j}}. \tag{3.9}$$

Additionally, as indicated in Section 3.1.2 ($t_i \in \mathcal{T}_i$), a schedule $\mathbf{t}$ must satisfy the following:

$$0 \leq t_i < T_i\,, \forall i \in \Pi. \tag{3.10}$$

In the following, let $\zeta(\mathbf{a}, \mathbf{b})$ denote the set of all feasible schedules $\mathbf{t}$ satisfying (3.7) and (3.10), where $\mathbf{b} = (b_i)$ is the vector representing partition time budgets, and $\mathbf{a}$ is the vector representing partition allocation to modules.

3.2.2 Resource constraints

Memory consumed by hosted partitions must respect the modules' memory capacities, that is,

$$\sum_{i \in \Pi} a_{i,k}\, m_i \leq M_k, \forall\, k \in \mathcal{P}. \tag{3.11}$$

Modules can host a maximum number of partitions, and hence,

$$\sum_{i \in \Pi} a_{i,k} \leq H_k,\ \forall k \in \mathcal{P}. \tag{3.12}$$

If two partitions i and j are in exclusion at module level, i.e. $(i,j) \in \mathcal{E}$, they must be allocated to two different modules, or in other words,

$$a_{i,k} \leq 1 - a_{j,k},\ \forall k \in \mathcal{P}, \forall (i,j) \in \mathcal{E}. \tag{3.13}$$

Whereas if these two partitions are in exclusion at cabinet level, i.e. $(i,j) \in \mathcal{E}_c$, then they must be allocated to modules from two different cabinets, that is,

$$a_{i,k} \leq 1 - \sum_{p \in c} a_{j,p},\ \forall k \in c, \forall c \in \mathcal{C}, \forall (i,j) \in \mathcal{E}_c. \tag{3.14}$$

Furthermore, it is possible to enrich the model by a new type of constraints to facilitate problem solving. These constraints are a result of a simple corollary of Theorem 3.

Corollary 5 *Two partitions i and j can be allocated the same module if and only if*

$$g_{i,j} \geq b_i + b_j \tag{3.15}$$

is verified.

Proof. The necessary and sufficient condition (3.5) implies (3.15). If this latter condition is not verified then (3.5) will also be violated, and hence, the implicated partitions can never be executed on the same module as they will always overlap in time. Condition (3.15) becomes only a necessary one in the presence of several partitions [91]. ■

It is consequently easy to add exclusion constraints similar to those of (3.13) between partitions that do not verify Corollary 5. Let $\mathcal{E}_a$ be the set of partition couples $(i,j) \in \Pi^2$ such that $b_i + b_j > g_{i,j}$, and add the following constraints:

$$a_{i,k} \leq 1 - a_{j,k},\ \forall k \in \mathcal{P}, \forall (i,j) \in \mathcal{E}_a. \tag{3.16}$$

Denote by $\mathcal{A}$ the set of vectors $\mathbf{a}$ satisfying (3.1), (3.11)-(3.16). It represents the set of all possible allocations satisfying the resource constraints.

3.2.3 Communication delay or latency constraints

The partitions of each processing chain $\lambda_c \in \Lambda$ must be allocated in such a manner that the resulting latency (end-to-end delay) $\mathcal{L}_{\lambda_c}$ does not exceed the predefined limit $\mathcal{L}_{\lambda_c}^{max}$. This can be expressed as:

$$\mathcal{L}_{\lambda_c} = \sum_{i=1}^{|\lambda_c|-1} L_{\lambda_c(i),\lambda_c(i+1)} + b_{\lambda_c(|\lambda_c|)} \leq \mathcal{L}_{\lambda_c}^{max}, \forall \lambda_c \in \Lambda, \tag{3.17}$$

where $L_{\lambda_c(i),\lambda_c(i+1)}$ represents the partition-to-partition communication delay, that is the delay to process data and send it between the consecutive partitions $\lambda_c(i)$ and $\lambda_c(i+1)$, and $b_{\lambda_c(|\lambda_c|)}$ is the execution time for the last partition in the chain. (3.17) indicates that the end-to-end communication delay is the sum of consecutive partition-to-partition communication delays in the chain, i.e. each partition in the sequence sends data to the following one.

Partition-to-partition communication delay, between partitions i and j for instance, represents the time needed to process data by i and send the result to j, either through the AFDX network, or locally through API ports. $L_{i,j}$ can hence be written under the following form:

$$L_{i,j} = b_i + T_j + \sum_{k\in\mathcal{P}} \sum_{l\in\mathcal{P}} a_{i,k} a_{j,l} \delta_{k,l}, \quad (i,j) \in \Pi^2 \tag{3.18}$$

The component $\sum_{k\in\mathcal{P}} \sum_{l\in\mathcal{P}} a_{i,k} a_{j,l} \delta_{k,l}$ adds the inter-module transmission delay between module k and l, where partitions i and j are respectively located, due to using the AFDX network. It should be noted that, if the two partitions are located on the same module, then this component will be zero; given that $\delta_{k,k} = 0$. The period T_j is also added to equation (3.18) indicating, with messages being read at the beginning of partition execution, either

1. the worst-case for data acquisition after reception by the destination module when the two partitions are on different modules, or
2. an upper bound on data reception when the two partitions are on the same module.

Figure 3.4 represents the three delay components appearing in equation (3.18).

The product $a_{i,k} a_{j,l}$ poses a problem on the linearity of the model (equation (3.18) is not linear because of the aforementioned product). In order to utilize the MILP formulation, a reformulation has to be done [49]. Equation (3.18) becomes:

$$L_{i,j} = b_i + T_j + \sum_{k\in\mathcal{P}} \sum_{l\in\mathcal{P}} z_{i,j,k,l} \delta_{k,l} \quad , (i,j) \in \Pi^2 \tag{3.19}$$

where the continuous variable $z_{i,j,k,l} \in [0,1]$ replaces the bilinear term $a_{i,k} a_{j,l}$ and has to satisfy the following linearization constraints,

$$z_{i,j,k,l} \leq a_{i,k}, \tag{3.20}$$

$$z_{i,j,k,l} \leq a_{j,l}, \tag{3.21}$$

$$z_{i,j,k,l} \geq -1 + a_{i,k} + a_{j,l}, \tag{3.22}$$

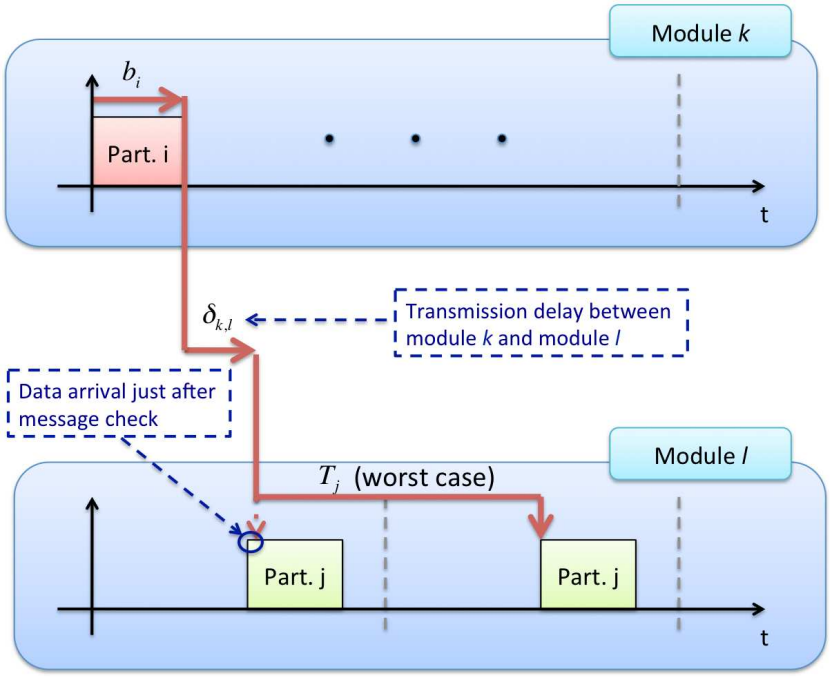


Figure 3.4: Delay components for partition couple communication.

$\forall(i,j) \in \Pi^2, \forall(k,l) \in \mathcal{P}^2$, which guarantee that $z_{i,j,k,l} = a_{i,k}a_{j,l}$. The following constraints (3.23)-(3.24) represent some observations,

$$z_{i,j,k,l} = z_{j,i,l,k}, \tag{3.23}$$

$$z_{i,i,k,k} = a_{i,k}. \tag{3.24}$$

The rather large number of linearization constraints (3.20)-(3.22) ($3n^2m^2$ constraints) can slow down the solution process considerably. For this reason, reduction constraints [106] are generated by multiplying the allocation constraints (3.1) by $a_{j,l}$ to obtain the following,

$$\sum_{k\in\mathcal{P}} z_{i,j,k,l} = a_{j,l} \quad ,\forall(i,j) \in \Pi^2, \forall l \in \mathcal{P}. \tag{3.25}$$

Proposition 6 *If the constraints* (3.1) *and* (3.25) *hold, provided* (3.23)*, then* $z_{i,j,k,l} = a_{i,k}a_{k,l}$ *and in particular* $z_{i,j,k,l} \in \{0,1\}$*. Therefore constraints* (3.1), (3.23), (3.25) *imply the linearization constraints* (3.20)-(3.22)*, as was shown in [106].*

Denote by Θ the set of vectors $\mathbf{a}$ (and consequently $\mathbf{z} = (z_{i,j,k,l})$) satisfying (3.17), (3.23), (3.25) and $z_{i,j,k,l} \in \{0,1\}$. This set represents all possible allocations where the communication delay constraints are respected. For sake of simplicity, the representation $(\mathbf{a},\mathbf{z}) \in \Theta$ is replaced by $\mathbf{a} \in \Theta$.

3.2.4 Formulation as a mixed integer linear program

A feasible solution to the scheduling problem is a couple $(\mathbf{a}, \mathbf{t})$, where $\mathbf{a}$ is an allocation and $\mathbf{t}$ is a schedule, satisfying constraints (3.1), (3.7), (3.10), (3.11)-(3.14), (3.16), (3.17), (3.23) and (3.25). In other words,

$$\mathbf{a} \in \mathcal{A} \cap \Theta,$$
$$\mathbf{t} \in \zeta(\mathbf{a}, \mathbf{b}).$$

For what concerns the optimization criterion considered in this thesis, it is desirable to choose a solution which ensures better evolution capacity for partitions, e.g. to permit adding new functionalities, without the need to reconsider all decisions (allocation and scheduling) already taken. Figure 3.5, for example, represents two possible solutions (S1) and (S2) for an allocation/scheduling problem with two modules and four partitions. It is obvious that the second solution (S2) offers more idle time in front of every partition execution, thus enabling to augment partition execution times if necessary. In the first solution, however, it is impossible to evolve available partitions on the first module, for what concerns time budgets.

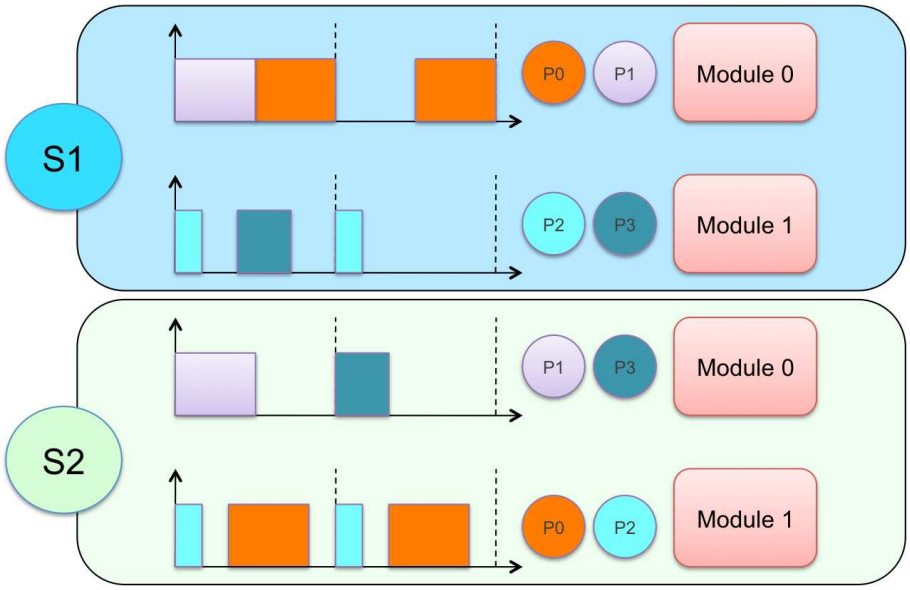


Figure 3.5: Scheduling can be carried out in different manners. A solution which provides better evolution potential for partition temporal executions is chosen.

For this reason, the problem is expressed as an optimization problem. The objective is to find a solution that maximizes the idle times between the task executions while ensuring that they do not overlap in time. In other words, maximizing a coefficient α by which all initial partition time budgets can be multiplied is sought, whilst respecting all system constraints and without interfering with other

partitions' executions.

Figure 3.6 shows the impact of α on a schedule constituted of two partitions on a given module. Hashed rectangles represent initial time budgets, whereas the larger filled ones represent the maximum allocable time budgets. Evidently, values of $\alpha < 1$ imply schedules with diminished time budgets and hence can be considered infeasible. In other words, values of $\alpha \geq 1$ indicate the feasibility of a given problem. In addition, increased values of α ensure overcoming errors arising from the underestimation of partition time budgets (*cf.* WCET in Section 2.2.1), if any.

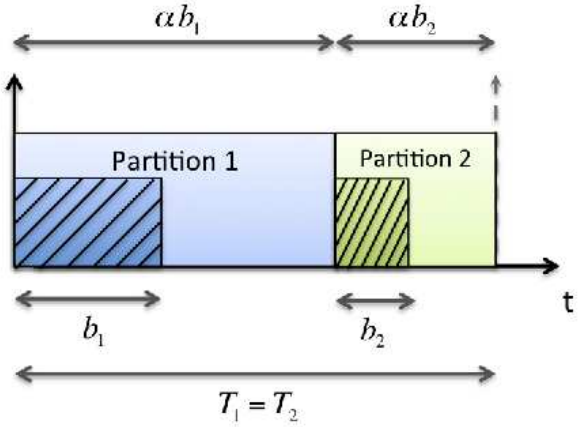


Figure 3.6: Impact of the evolution coefficient α on the scheduling.

The allocation and scheduling problem can now be formulated as follows,

$$\text{Max}_{\mathbf{a},\mathbf{t}}\alpha$$

s.t.

$$\mathbf{a} \in \mathcal{A} \cap \Theta,$$
$$\mathbf{t} \in \zeta(\mathbf{a}, \alpha\,\mathbf{b}).$$

In more detail, the complete formulation is:

$$\text{Maximize } \alpha \tag{3.26}$$

$s.t.$

$$\sum_{k\in\mathcal{P}} a_{i,k} = 1, \quad \forall i \in \Pi, \tag{3.27}$$

$$\sum_{i\in\Pi} a_{i,k}\, m_i \leq M_k, \quad \forall k \in \mathcal{P}, \tag{3.28}$$

$$\sum_{i\in\Pi} a_{i,k} \leq H_k, \quad \forall k \in \mathcal{P}, \tag{3.29}$$

$$a_{i,k} \leq 1 - a_{j,k}, \quad \forall k \in \mathcal{P}, \forall (i,j) \in \mathcal{E}, \tag{3.30}$$

$$a_{i,k} \leq 1 - \sum_{p\in c} a_{j,p}, \quad \forall k \in c, \forall c \in \mathcal{C}, \forall (i,j) \in \mathcal{E}_c, \tag{3.31}$$

$$a_{i,k} \leq 1 - a_{j,k}, \quad \forall k \in \mathcal{P}, \forall (i,j) \in \mathcal{E}_a, \tag{3.32}$$

$$(t_j - t_i) - q_{j,i}\, g_{i,j} \leq g_{i,j} - \alpha\, b_j + (2 - a_{i,k} - a_{j,k})\, Z, \quad \forall k \in \mathcal{P}, \forall (i,j) \in \Pi^2, \tag{3.33}$$

$$(t_j - t_i) - q_{j,i}\, g_{i,j} \geq \alpha\, b_i - (2 - a_{i,k} - a_{j,k})\, Z, \quad \forall k \in \mathcal{P}, \forall (i,j) \in \Pi^2, \tag{3.34}$$

$$\sum_{i=1}^{|\lambda_c|-1} \left(b_{\lambda_c(i)} + T_{\lambda_c(i+1)} + \sum_{k\in\mathcal{P}} \sum_{l\in\mathcal{P}} z_{i,i+1,k,l} \delta_{k,l} \right) + b_{\lambda_c(|\lambda_c|)} \leq \mathcal{L}_{\lambda_c}^{max}, \quad \forall \lambda_c \in \Lambda, \tag{3.35}$$

$$\sum_{k\in\mathcal{P}} z_{i,j,k,l} = a_{j,l}, \quad \forall (i,j) \in \Pi^2, \forall l \in \mathcal{P}, \tag{3.36}$$

$$z_{i,j,k,l} = z_{j,i,l,k}, \quad \forall (i,j) \in \Pi^2, \forall (k,l) \in \mathcal{P}, \tag{3.37}$$

$$\frac{-T_i + b_i}{g_{i,j}} - 1 < q_{j,i} \leq \frac{T_j - b_j}{g_{i,j}}, \quad \forall (i,j) \in \Pi^2, \tag{3.38}$$

$$a_{i,k} \in \{0,1\}, \quad \forall k \in \mathcal{P}, \forall i \in \Pi, \tag{3.39}$$

$$t_i \in \mathcal{T}_i, \quad \forall i \in \Pi, \tag{3.40}$$

$$z_{i,j,k,l} \in [0,1], \quad \forall (i,j) \in \Pi^2, \forall (k,l) \in \mathcal{P}^2. \tag{3.41}$$

The above formulation seeks maximizing the minimum evolution potential of the partitions in the system (temporal execution-wise). It will be possible to find partitions capable of evolving with a potential corresponding to this minimum value, and others capable with a more greater one.

The originality of this objective is that most of the work presented in literature focused on minimizing the number of used processors, or even the makespan. This thesis, on the contrary, aims at provisioning an augmented temporal flexibility for the partitions' executions, so that spare processing power could be allocated to meet the resource demand growth with minimal configuration modification.

3.3 Pre-treatment using graph theory

In this section, reducing the number of problem variables is searched, which may reduce the time required for solving the MILP presented in Section 3.2.4. For this, a pretreatment phase based on graph

theory is proposed. This pretreatment must guarantee that the optimal solution remains accessible. The method illustrated hereafter was inspired from the work carried out by Korst in [91] and is applied when all modules are identical, which is the case in reality. In the case where the modules are heterogeneous, the method no longer remains applicable.

A graph G, in which each node is associated to a partition, is constructed. An arc connecting two nodes is established if the two corresponding partitions are not involved in an exclusion, e.g. condition (3.15) is not verified. If two nodes are not connected, it is impossible to map the two corresponding partitions on the same module.

The connected components of G represent partition sets that must be mapped onto different modules. The search for a Maximal Independent Set (MIS) in G [138], allows obtaining a set of totally independent partitions, that must consequently be allocated different modules. Algorithms for finding connected components and maximal independent sets can be found in Appendix A and are based on graph theory basics and some of the algorithms developed in [2] and [138].

For every connected component of G, partitions corresponding to nodes in the MIS are placed on distinct modules–since partitions of this set cannot be placed on the same module–, this allows to minimize the number of problem allocation variables and accelerate the solution process.

Figure 3.7 represents the graph representation for a set of six partitions. Each node couple is connected by an arc if no exclusion exists between the corresponding partitions, according to previously defined exclusion constraints (*cf.* Section 3.2.2). For example, partitions 1 and 2 are likely to be placed on the same module, while partitions 1 and 3 surely do not co-exist on the same one. Nodes 1, 3 and 6 are found to represent an MIS, hence, by fixing the allocations for partitions 1, 3 and 6 on three different modules, the problem's allocation variables can be in effect minimized from six to three.

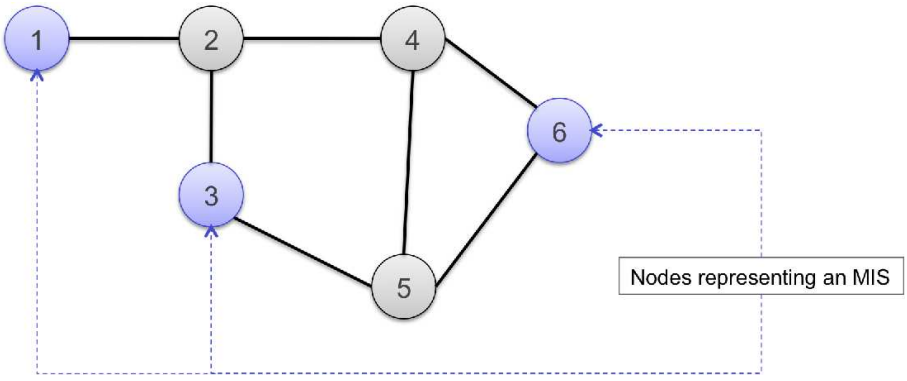


Figure 3.7: Graph representation for a set of partitions.

3.4 Results

In this section, some experimentation on the optimization problem (*cf.* Section 3.2.4) alongside the pretreatment proposed in Section 3.3 are presented. The MILPs where solved using the linear program solver CPLEX [81] from IBM ILOG. The machine used is based on a quad-core Intel® Xeon™rated at 3.2GHz with 8MB of cache and 32GB system memory.

A chosen set of twenty-three problems, in which the number of modules and partitions varied from 1 to 4 and 6 to 80 respectively, are demonstrated in Table 3.1. Partition periods were taken in time intervals as shown in the same table. Partition time budgets were considered of small order as compared to the periods so as to obtain schedulable systems. Partition periods were generated in a manner limiting the LCM between them based on a method similar to that indicated in [73], and hence, obtaining a not so large MAF on each module. In addition, these periods were considered general (non-harmonic) as the MILP formulation should be suitable for all period types, harmonic and non-harmonic alike, nevertheless this period scheme was considered so as not to limit the study to harmonic instances, which may be the case in most applications. Communication was considered so that 40 to 60 percent of partitions were involved in some kind of processing chain (acquisition and transmission of data). Partition exclusions were generated such that 40 percent of partitions at maximum were involved. Memory capacity and partition count on modules were generated to impose some constraints on the problems. Memory requirements for partitions, for instance, were generated such that their sum equals the total memory capacities of modules multiplied by a certain utilisation factor (taken as 50%).

The problems were solved by considering offsets ($t_i \, , \forall i \in \Pi$) as integer and continuous variables respectively, though mainly interest is in the former. Theoretically, solving for t_i continuous should give some relaxation to the problem, and consequently, one should expect faster computation times (CpuTime).

Table 3.1 is divided into two parts, experiments 1 to 11 where periods ranged up to couple of hundreds, and experiments 12 to 23 where periods ranged up to couple of thousands. Values of α demonstrate the optimization carried out, where for example in experiments 4 and 16, partition time budgets can be increased by half ($\alpha = 151\%$) without affecting the proposed allocation and schedule.

Computation times for experiments 1 to 7 and 13 to 17 were insignificant, whereas for the rest, and depending on the complexity of the problem, became more important. Experiments 10, 11, 22 and 23 were stopped after 24 hours of execution. Experiments 10 and 22 found feasible solutions but could not prove optimality in the indicated time. Experiments 11 and 23 were so complex that no solution was even found for the same amount of time. This brings us to the point that real complex systems and even future ones, where number of modules and partitions may be of importance, may pose a problem for the MILP formulation.

The difference in computation times between experiments 12 and 17, with the latter being presumably harder, is a clear indication that the complexity of a problem arises not only from the number of components (modules and partitions) but also from any of the component attributes (such as partition periods). To further emphasize on this, Figure 3.8 demonstrates 9 examples with the same number of

Table 3.1: Time required for solving the mixed integer linear program

Experiment	Module number	Partition number	Partition period range	CpuTime t_i : int	CpuTime t_i : real	α t_i : int	α t_i : real
1	3	6	[50,150]	0.04s.	0.01s.	100%	111,1%
2	3	6	[60,300]	0.05s.	0.01s.	135.93%	136.36%
3	3	6	[40,350]	0.03s.	0.01s.	134.61%	137.93%
4	3	6	[120,400]	0.03s.	0.01s.	151.61%	151.89%
5	4	10	[30,360]	0.1s.	0.07s.	145%	145.16%
6	4	10	[90,400]	0.3s.	0.08s.	114.28%	115.38%
7	4	10	[80,400]	0.22s.	0.05s.	108%	108.10%
8	4	20	[20,360]	2hr.	1s.	115.38%	115.38%
9	4	20	[10,270]	5.57hr.	6.5s.	110%	125%
10	4	30	[10,720]	24hr.+	24hr.+	(200%)	(200%)
11	4	80	[20,450]	24hr.+	24hr.+	-	-
12	1	10	[600,2400]	14.53mn.	3.14mn.	136%	136.36%
13	3	6	[500,1500]	0.02s.	0.01s.	126.12%	126.12%
14	3	6	[600,3000]	0.2s.	0.01s.	136.25%	136.36%
15	3	6	[400,3500]	0.21s.	0.01s.	137.69%	137.93%
16	3	6	[1200,4000]	0.62s.	0.01s.	151.87%	151.89%
17	4	10	[300,3600]	0.09s.	0.07s.	145.11%	145.16%
18	4	10	[900,4000]	1.22s.	0.08s.	115.35%	115.38%
19	4	10	[800,4000]	0.38s.	0.05s.	108%	108.10%
20	4	20	[200,3600]	12.07s.	2.36s.	115.38%	115.38%
21	4	20	[100,2700]	6hr.	15.7mn.	123.33%	125%
22	4	30	[100,7200]	24hr.+	24hr.+	(200%)	(200%)
23	4	80	[200,4500]	24hr.+	24hr.+	-	-

components, 4 modules and 40 partitions, and where the computation time varied from about 5 to 50 minutes.

It should be also noted that, though not presented in Table 3.1, the proposed pretreatment phase (*cf.* Section 3.3) had a great impact on reducing the computation times as can be clearly seen in Table 3.2, representing a set of five separate problems.

What seems to be interesting from Table 3.1, is that solving for continuous values of time budgets improves computation times (e.g. from about 5 and a half hours to 6.5 seconds in experiment 9) while obtaining α values that are close to those from solving the original problem (without relaxation of time budgets), especially for greater order of periods. This leads to a first heuristic that is based on solving the problem for $t_i \in \mathbb{R}, \forall i \in \Pi$, then by guarding the computed allocations, sub-MILP problems based on finding an optimal scheduling on each module are solved, but this time for $t_i \in \mathbb{N}$. It is admitted that this two phase heuristic may be inefficient in some cases where solving with relaxation, as was shown in Table 3.1, may require a colossal amount of time for computation. Nevertheless it seemed interesting

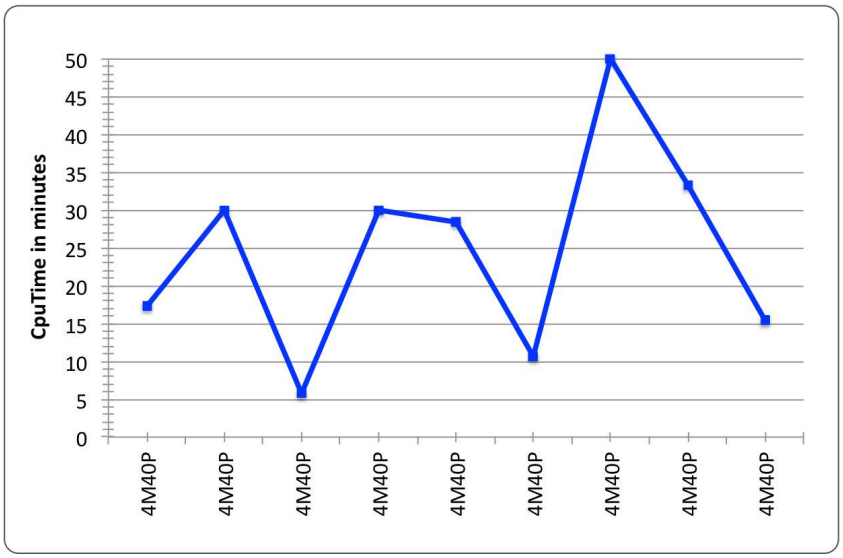


Figure 3.8: Computation time depends on system characteristics.

Table 3.2: The impact of pretreatment phase

Experiment	Module count	Partition count	CpuTime (no pretreatment)	CpuTime (pretreatment)
a	3	6	0.31s.	0.01s.
b	4	8	0.78s.	0.25s.
c	4	8	3.87s.	0.77s.
d	8	16	1.9hr.	29s.
e	8	16	5days	11mn.

to investigate this method and test it on a simple set of experiments as shown in Table 3.3.

In this latter table, computation times were reduced significantly without degrading much the quality of the solution. For instance, Experiment 8 computation time was reduced from 2 hours to approximately 10 seconds, with about 1% relative error on the optimization coefficient α. However, as indicated earlier this reduction in computation time is not always guaranteed.

3.5 Conclusion

In this chapter, an exact MILP formulation model for the scheduling problem has been proposed. The particularity resides in the strict periodicity of partition executions and the diversity of system constraints, in addition to the proposed optimization objective. The experimentations have shown the inefficiency of the proposed approach for fairly large examples, although a pretreatment phase can facilitate

Table 3.3: Two phase heuristic

Experiment	CpuTime original	CpuTime two phase	α original	α two phase
7	0.22s.	0.05s.	108 %	108 %
8	2hr.	10.37s.	115.38 %	114.28 %
9	5.57hr.	26.68mn.	110 %	100 %
15	0.21s.	0.01s.	137.69 %	136.66 %
16	0.62s.	0.01s.	151.87 %	151.87 %
18	1.22s.	0.8s.	115.35 %	115.35 %
21	6hr.	16mn.	123.33 %	123.33 %

the problem. The major inconvenience is the computation time where, for quite complex architectures with a significant number of components, it can become somewhat colossal.

The following chapter focuses on developing a less complex adapted heuristic and whose performance can be assessed with the aid of the aforementioned exact method. Therefore, this exact formulation has the utmost utility in assessing the quality of proposed approximations or heuristics.

CHAPTER 4

A best-response scheduling algorithm

In Chapter 3, an exact algorithm was suggested for the non-preemptive and strictly periodic scheduling problem. In this chapter, a heuristic inspired from Game Theory (*cf.* Section 2.6.3) is introduced. It is first implemented in a uniprocessor setting, to then be extended to the multiprocessor one, the complete architecture. Partitions are allowed to independently select their strategies, or in other words their offsets (and module allocations in the multiprocessor setting), to maximize their own utility function. A utility function is simply another notion to represent the payoff function from Section 2.6.3. A detailed analysis of this algorithm and an efficient scheme to compute the best-response strategy of a partition are provided. The convergence of the algorithm to an equilibrium point where no partition has any incentive to unilaterally deviate is also shown.

As will be shown numerically, the main merit of this algorithm is that it is several orders of magnitude quicker than the exact MILP formulation proposed in Chapter 3, while at the same time it generates periodic schedules with modest relative errors with respect to optimal solutions. In particular, it is remarkable that this best-response algorithm was able to find feasible solutions to aircraft sized problems provided by one of the industrial partners in a few minutes. Furthermore, in order to increase the chances of obtaining an optimal or near optimal solutions, multi-start methods can be implemented. These methods explore randomly the solution space and give some estimations on the optimality of the generated equilibria.

In Section 4.1, the best-response algorithm is introduced for uniprocessor scheduling, to then extend it to the multiprocessor one in Section 4.2. In Section 4.3, multi-start methods and the application of Bayesian stopping rules are discussed. Some results and performances of the algorithm are demonstrated in Section 4.4 to finally conclude in Section 4.5.

4.1 Uniprocessor or single module scheduling

Before addressing the multiprocessor scheduling problem, light is shed on the uniprocessor one. A set Π of N partitions are considered resident on a given processing module. A schedule, or an offset vector $\mathbf{t}$ is sought so that no overlapping in time occurs between partition executions. In this setting, each partition $i \in \Pi$ is characterized by,

- its strict period T_i, and
- its time budget b_i.

As was indicated in the previous chapter, let $\mathcal{T}_i = \{0, \ldots, T - i - 1\}$ be the set of offset possibilities for partition i, and let $\mathcal{T} = \times_{i=1}^{N} \mathcal{T}_i$ be the set of all possible offset vectors.

As was already shown in Theorem 3 from Chapter 3, a sufficient and necessary condition for the scheduling of two partitions i and j on the same module is

$$b_i \leq (t_j - t_i)\%g_{i,j} \leq g_{i,j} - b_j.$$

The term $\frac{(t_j - t_i)\%g_{i,j}}{b_i}$ can be interpreted as the maximum factor by which b_i can be multiplied without interfering with the executions of partition j. It can be thought of as the evolution margin for partition i with respect to partition j. This interpretation easily follows from Lemma 2 in Section 3.2.1.1.

As was indicated in Section 3.2.4, the interest is in a maximized factor, denoted α, by which all partition durations can be multiplied while still guaranteeing the existence of a feasible schedule. This will allow an evolution margin for partition budget times, should it be required for future expansion of partitions. The MILP formulation is, in the uniprocessor case, and assuming that all resource constraints are satisfied, written as,

$$\begin{array}{ll} \text{maximize} & \alpha \\ \text{subject to} & \\ & (t_j - t_i) - q_{j,i}\, g_{i,j} \geq \alpha\, b_i,\ \forall (i,j) \in \Pi^2, \\ & (t_j - t_i) - q_{j,i}\, g_{i,j} \leq g_{i,j} - \alpha\, b_j,\ \forall (i,j) \in \Pi^2, \\ & q_{j,i} \in \mathbb{Z},\ \forall (i,j) \in \Pi^2, \\ & \mathbf{t} \in \mathcal{T}. \end{array}$$

Let

$$d_{ij}(\mathbf{t}) = \min\left(\frac{(t_j - t_i)\%g_{i,j}}{b_i}, \frac{(t_i - t_j)\%g_{i,j}}{b_j}\right), \tag{4.1}$$

which represents the minimum evolution between partitions i and j. In other words, it represents the factor by which both partition durations can be multiplied without violating the temporal feasibility of the schedule. Consequently, the scheduling problem can be stated as follows

$$\begin{array}{ll} \text{maximize} & \min\limits_{i,j \neq i} d_{ij}(\mathbf{t}), \\ \text{subject to} & \mathbf{t} \in \mathcal{T}. \end{array} \tag{OPT-1}$$

In the following section, a fast algorithm for generating offset vectors is discussed. As will be shown numerically, the main merit of this algorithm is that it is much quicker than the exact method for solving the MILP proposed in Chapter 3 while at the same time it generates offset vectors with modest relative errors.

4.1.1 Uniprocessor best-response

The best-response algorithm is inspired from an algorithm of the same name in Non-cooperative Game Theory (*cf.* Section 2.6.3.1). In a game, the best-response of a player is defined as its optimal strategy conditioned on the strategies of the other players. It is, as the name suggests, the best response that the player can give for a given strategy of the others. The best-response algorithm then consists of players taking turns to adapt their strategy based on the most recent known strategy of the others.

The proposed best-response algorithm converts the optimization problem (OPT-1) into the following game. Think of partitions as players. The game is assumed to be played sequentially with players taking turns in some fixed order until the strategies (offsets) of the players converge. An emphasis should be made out on the fact that, in each turn, exactly one player computes its best-response while the others keep their strategies unchanged.

Let t_i^n denote the strategy of player i at the beginning of the nth iteration. Given that it is the turn of player i in this iteration, it computes its offset so as to maximize its relative distances with the other partitions, that is, player i solves the following problem :

$$\begin{aligned} &\text{maximize } \alpha_i(x) = \min_{j \neq i} d_{i,j}(x, \mathbf{t}_{-i}^n) \\ &\text{subject to } \ x \in \mathcal{T}_i, \end{aligned} \tag{SCHD-i}$$

where in standard Game Theory notation, $\mathbf{t}_{-i}^n = (t_1^n, t_2^n, \ldots, t_{i-1}^n, t_{i+1}^n, \ldots, t_N^n)$ is the vector of offsets of players other than player i. The player i then sets t_i^{n+1} to that value of x that gives the solution. In case the best-response is not unique, then it retains the smallest offset from the set of best-responses. Note that player i solves the same problem as (OPT-1) except that it takes into account only the terms that are affected by its offset. Figure 4.1 demonstrates the response α_1 of player 1 based on its strategy x. Player 1 chooses the offset x which gives a best-response (maximum). The example of Figure 4.1 has players with unitary time budgets ($b_i = 1, \forall i$), this leads to a simpler representation of $d_{i,j}$, that is

$$d_{i,j} = \min\left((t_j - t_i)\%g_{i,j}, (t_i - t_j)\%g_{i,j}\right)$$

Remark 7 *In a usual non-cooperative game each player seeks to maximize its own objective function. In that setting the natural objective of a partition would be to determine the offset that maximizes the factor by which it can increase its own duration, which would amount to solving the problem*

$$\begin{aligned} &\textit{maximize } \min_{j \neq i} \left(\frac{(t_j - x)\%g_{i,j}}{b_i} \right) \\ &\textit{subject to } \ x \in \mathcal{T}_i. \end{aligned}$$

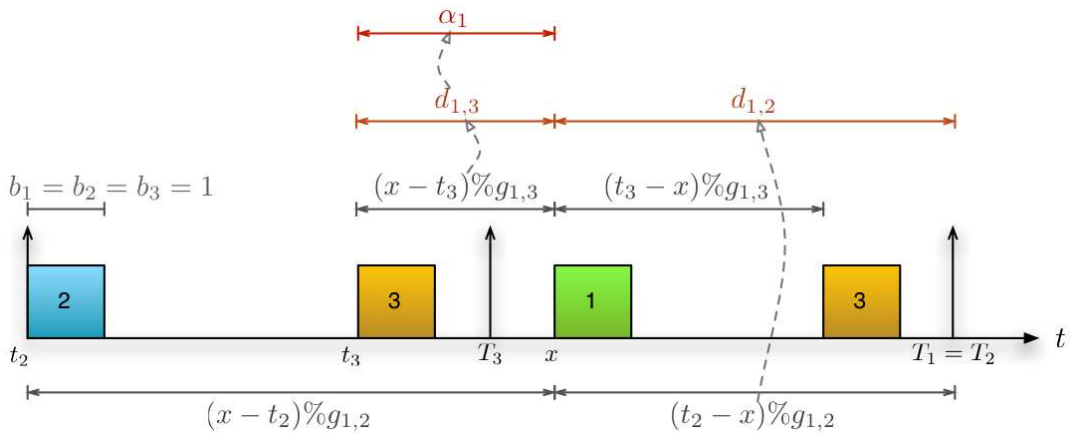


Figure 4.1: Strategy of player 1, given the fixed strategies of players 2 and 3. Each choice of an offset x gives rise to a certain response $\alpha_1(x)$

The objective function in (SCHD-i) *is not defined as such, but it takes into consideration other partitions' interest whenever a given one changes its strategy.*

In the following, define

$$\alpha_i^n = \min_{j \neq i} d_{i,j}(\mathbf{t}^n) \tag{4.2}$$

$$\mathcal{S}_i(\mathbf{t}_{-i}^n) = \operatorname*{argmax}_{x \in \mathcal{T}_i} \min_{j \neq i} d_{i,j}(x, \mathbf{t}_{-i}^n), \tag{4.3}$$

where α_i^n is the utility (payoff) of player i after the nth iteration, and $\mathcal{S}_i(\mathbf{t}_{-i}^n)$ is the set of all the best-responses of player i.

It shall be assumed that if the best-response of a player i does not improve its current utility α_i^n, then the player does not change its strategy, i.e. $t_i^{n+1} = t_i^n$. This assumption although not restrictive will allow to prove the convergence of the algorithm.

The pseudocode and flowchart for the best-response algorithm are given in Algorithm 5 and Figure 4.2 respectively. In step 4 of the algorithm, $n\%N + 1$ gives the index of the player whose turn it is in the nth iteration.

4.1.2 Properties of the best-response algorithm

Two important properties of the best-response algorithm are proven in this section. The first property states that the algorithm converges. Indeed, for recursive algorithms like the best-response, convergence of the algorithm is a desired property. There however need not be a unique equilibrium point. The second property states that one or more of the equilibrium points correspond to the solutions of (OPT-1). Consequently, if the initial point is chosen appropriately, the best-response algorithm will converge to

Algorithm 5 Uniprocessor best-response

Require: $\mathbf{t}^0$
$n \leftarrow 0$
repeat
  for $i = 1$ to N **do**
    if $i = n\%N + 1$ **and** $\max_x \min_{j \neq i} d_{i,j}(x, \mathbf{t}^n_{-i}) > \alpha^n_i$ **then**
      $t^{n+1}_i \leftarrow \min \operatorname{argmax}$ (SCHD-i)
    else
      $t^{n+1}_i \leftarrow t^n_i$
    end if
  end for
  $n \leftarrow n + 1$
until $\mathbf{t}^n = \mathbf{t}^{n-N}$.
return $\mathbf{t}^n$

an optimal solution.

Lemma 8 *Assume that at iteration n, player i updates its strategy. If $\alpha^{n+1}_i > \alpha^n_i$, then for all $j \neq i$*

$$\alpha^n_j < \alpha^n_i \Rightarrow \alpha^{n+1}_j = \alpha^n_j \tag{4.4}$$
$$\alpha^n_j = \alpha^n_i \Rightarrow \alpha^{n+1}_j \geq \alpha^n_j \tag{4.5}$$
$$\alpha^n_j > \alpha^n_i \Rightarrow \alpha^{n+1}_j > \alpha^n_i \tag{4.6}$$

Proof. To be more concise, d^n_{ji} is used instead of $d_{ji}(\mathbf{t}^n)$. Observe that at iteration n only player $i = n\%N + 1$ updates its strategy, and thus $\mathbf{t}^{n+1}_{-i} = \mathbf{t}^n_{-i}$. Assume that $\alpha^{n+1}_i > \alpha^n_i$. Consider a partition $j \neq i$ such that $\alpha^n_j < \alpha^n_i$. Since $t^{n+1}_k = t^n_k$ for all $k \neq i$,

$$\alpha^{n+1}_j = \min\left(\min_{k \neq i,j} d^{n+1}_{jk}, d^{n+1}_{ji}\right) = \min\left(\min_{k \neq i,j} d^n_{jk}, d^{n+1}_{ji}\right) = \alpha^n_j,$$

where the last equality is obtained thanks to the following inequalities:

$$\min_{k \neq i,j} d^n_{jk} = \alpha^n_j < \alpha^n_i < \alpha^{n+1}_i \leq d^{n+1}_{ji}.$$

Consider now a partition $j \neq i$ such that $\alpha^n_j = \alpha^n_i$. Observe that $d^{n+1}_{ij} \geq \alpha^{n+1}_i > \alpha^n_i = \alpha^n_j$. In addition

$$\min_{k \neq i,j} d^{n+1}_{jk} = \min_{k \neq i,j} d^n_{jk} \geq \alpha^n_j. \tag{4.7}$$

It thus can be concluded that

$$\alpha^{n+1}_j = \min(\min_{k \neq i,j} d^{n+1}_{jk}, d^{n+1}_{ij}) \geq \alpha^n_j.$$

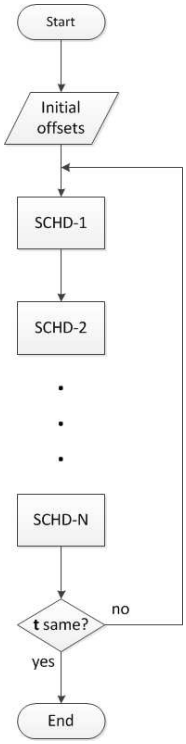


Figure 4.2: Flowchart for the best-response algorithm.

Finally, consider a partition $j \neq i$ such that $\alpha_j^n > \alpha_i^n$. Again $d_{ij}^{n+1} \geq \alpha_i^{n+1} > \alpha_i^n$. Furthermore equation (4.7) holds and implies that $\min_{k \neq i,j} d_{jk}^{n+1} > \alpha_i^n$. Thus $\alpha_j^{n+1} > \alpha_i^n$. ■

For each vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_N)$, let $\hat{\boldsymbol{\alpha}}$ denote the vector obtained from $\boldsymbol{\alpha}$ by sorting the values in the increasing order. Define $\boldsymbol{\alpha} \succ \boldsymbol{\beta}$ as $\hat{\boldsymbol{\alpha}}$ greater than $\hat{\boldsymbol{\beta}}$ in the lexicographic order, i.e.

$$\exists k \leq N \quad \hat{\alpha}_i = \hat{\beta}_i \; \forall i < k \quad \text{and} \quad \hat{\alpha}_k > \hat{\beta}_k. \tag{4.8}$$

In order to show that the best-response algorithm converges, it is first shown that the vector $\boldsymbol{\alpha}^n = (\alpha_1^n, \ldots, \alpha_N^n)$ at iteration n increases in the lexicographical sense, and that this vector is bounded for all n. These two properties are sufficient for the convergence of a given sequence.

The following is a demonstration that the vector $\boldsymbol{\alpha}$ increases in the lexicographic sense when a player updates its strategy.

Proposition 9

$$\alpha_i^{n+1} > \alpha_i^n \Rightarrow \boldsymbol{\alpha}^{n+1} \succ \boldsymbol{\alpha}^n \tag{4.9}$$

Proof. Note that since $\alpha_j^{n+1} = \alpha_j^n$ for all j such that $\alpha_j^n < \alpha_i^n$, the vectors $\hat{\boldsymbol{\alpha}}^n$ and $\hat{\boldsymbol{\alpha}}^{n+1}$ have a common prefix (maybe of length 0). Consider the multiplicity of the value α_i^n in the vector $\hat{\boldsymbol{\alpha}}^n$. According to Lemma 8, the multiplicity of this value decreases by at least one when partition i changes its strategy. Since $\alpha_i^{n+1} > \alpha_i^n$ and $\alpha_j^{n+1} > \alpha_i^n$ for all j such that $\alpha_j^n > \alpha_i^n$, it concludes the proof. ■

Next, in order to show that the vector $\boldsymbol{\alpha}^n$ is bounded from above, it is sufficient to show that α_i^n is bounded from above.

Lemma 10 *For all $i \in \Pi$ and $n \geq 1$,*

$$\alpha_i^n \leq \min_{j \neq i} \frac{g_{i,j}}{b_i + b_j}.$$

Proof. From (4.2), $\forall j \neq i$,

$$\begin{aligned}
\alpha_i^n &\leq d_{i,j}(\mathbf{t}^n), \\
&\leq \min\left(\frac{(t_i - t_j)\%g_{i,j}}{b_j}, \frac{(t_j - t_i)\%g_{i,j}}{b_i}\right) \\
&\leq \min\left(\frac{(t_i - t_j)\%g_{i,j}}{b_j}, \frac{g_{i,j} - (t_i - t_j)\%g_{i,j}}{b_i}\right),
\end{aligned}$$

which yields $\alpha_i^n b_j < (t_i - t_j)\%g_{i,j}$ and $\alpha_i^n b_i < g_{i,j} - (t_i - t_j)\%g_{i,j}$. Consequently, $\alpha_i^n(b_i + b_j) < g_{i,j}$. ■

Theorem 11 *The best-response algorithm converges.*

Proof. The sequence $\boldsymbol{\alpha}^n$ is increasing in the lexicographic order. Since it is bounded from above, the conclusion is that it converges. ■

It is now shown that the best-response algorithm converges in a finite number of iterations.

Lemma 12 *Let $\Delta = \min_{j,k} \frac{1}{lcm(b_j,b_k)}$. Given that partition i changes its strategy, the value of the minimum step by which partition i can increase α_i^n is Δ.*

Proof. Note that α_i^n is of the form $\frac{m_n}{b_j}$ for some integer m_n and some partition j which could be the same as partition i. Thus,

$$\begin{aligned}
\alpha_i^{n+1} - \alpha_i^n = \frac{m_{n+1}}{b_j} - \frac{m_n}{b_k} &= \frac{m_{n+1}b_k - m_n b_j}{b_j b_k} \\
&= \frac{l \cdot \text{gcd}(b_j, b_k)}{b_j b_k} = \frac{l}{\text{lcm}(b_j, b_k)},
\end{aligned}$$

for some integer l.

Given that partition i changes its strategy, $\alpha_i^{n+1} - \alpha_i^n > 0$, l is strictly positive. Therefore, $\alpha_i^{n+1} - \alpha_i^n \geq \frac{1}{\text{lcm}(b_j,b_k)} \geq \Delta$. ■

Define

$$\alpha_{max} = \max_i \min_{j \neq i} \frac{g_{i,j}}{b_i + b_j}.$$

Lemma 13 *For two integers $N, K \in \mathbb{N}$, the number of vectors $\mathbf{v} = (v_1, v_2, \ldots, v_N)$ such that $v_j \leq v_{j+1}, j = 1, \ldots, N-1$, and $v_j \in \{1, \ldots, K\}, \forall j = 1, \ldots, N$, is $\binom{N+K}{K}$.*

Proof. Let $F(i, N)$, with $i < K$, denote the number of vectors $\mathbf{v} = (v_1, v_2, \ldots, v_N)$ with $v_j \in \{K - i, K - i + 1, \ldots, K\}$ and $v_j \leq v_{j+1}, \forall j$. Clearly $v_1 \geq K - i$, and hence, $F(i, N)$ can be interpreted from $v_1 \geq K - i + 1$ and $v_1 = K - i$, or in other words as the number of

- vectors $v' = (v'_1, v'_2, \ldots, v'_N)$ such that $v'_j \in \{K - i + 1, \ldots, K\}$ and $v'_j \leq v'_{j+1}, \forall j$, and
- vectors $v'' = (v''_1, v''_2, \ldots, v''_{N-1})$ such that $v''_j \in \{K - i, \ldots, K\}$ and $v''_j \leq v''_{j+1}, \forall j$.

In other words,

$$F(i, N) = F(i - 1, N) + F(i, N - 1).$$

Setting $F(i, N) = \binom{N+i}{i}$ can be verified from the following,

$$\begin{aligned} F(i-1,N) + F(i,N-1) &= \binom{N+i-1}{i-1} + \binom{N-1+i}{i} &&= \frac{(N+i-1)!}{N!\,(i-1)!} + \frac{(N+i-1)!}{(N-1)!\,i!} \\ &= \frac{(N+i-1)!}{(N-1)!\,(i-1)!}\left(\frac{1}{N} + \frac{1}{i}\right) &&= \frac{(N+i)!}{N!\,i!} \\ &= F(i,N) \end{aligned}$$

and hence the proof is concluded. ■

Proposition 14 *The best-response algorithm converges in at most $\binom{N+K}{K}N$ iterations where $K = \lceil \alpha_{max} \Delta^{-1} \rceil$.*

Proof. The sequence $\boldsymbol{\alpha}^n$ is lexicographically ordered and is bounded from above by the vector

$$\boldsymbol{\alpha}_{max} = (\alpha_{max}, \alpha_{max}, \ldots, \alpha_{max}).$$

Additionally, from Lemma 12, the minimum step size of α_i^n is Δ. Hence, the maximum number of possible vectors $\boldsymbol{\alpha}^n$ is the same as the number of vectors $\mathbf{v} \in \mathbb{Z}^N$ such that $(K, K, \ldots, K) \succeq \mathbf{v} \succeq \mathbf{0}$ and $v_j \leq v_{j+1}, \forall j$. According to Lemma 13, the number of such vectors is $\binom{N+K}{K}$.

The worst-case scenario for the convergence of the algorithm occurs when in N consecutive iterations exactly one player changes its strategy so that exactly one of the $\binom{N+K}{K}$ possible vectors is traversed in these iterations.

Thus, the maximum number of iterations is upper bounded by $\binom{N+K}{K}N$. ■

Remark 15 *The upper bound obtained is a conservative estimate which is exponential in the number of partitions. However, in the numerical experiments presented in Section 4.4, the algorithm always converged in a few tens of iterations.*

It should be noted that there can be multiple equilibrium points. In Game Theory, an equilibrium point is also known as a Nash Equilibrium Point after John Nash who made major contributions to the theory of Non-Cooperative Games.

Theorem 16 *There exists at least one equilibrium point that is also a solution of* (OPT-1).

Proof. Let $\mathbf{t}^0$ be a solution of (OPT-1). From Proposition 9, $\hat{\boldsymbol{\alpha}}^n$ is non-decreasing lexicographically, that is $\alpha_1^n \geq \alpha_1^0$. Since $\mathbf{t}^0$ has been assumed to be an optimal offset vector, α_1^0 is maximal. Thus, $\alpha_1^n = \alpha_1^0$, and consequently, $\mathbf{t}^n$ is also an optimal offset vector for all n. From Theorem 11, the sequence $\mathbf{t}^n$ converges. Hence, it can be concluded that there is at least one equilibrium point that is also a solution of (OPT-1). ■

4.1.3 Computing the best-response

The best-response of partition i can be computed using linear search (at least when the offsets are restricted to integers) which requires $O(T_i)$ computations. In the following, a method to reduce the computational complexity of the best-response is proposed.

In the rest of this section the following assumption is considered.

Assumption 1 *The offsets can take on non-integral values, i.e.* $\mathcal{T}_i = [0, T_i)$.

Let

$$\mathcal{I}_i(\mathbf{t}_{-i}) = \bigcup_{(j,k)\in(\Pi\setminus\{i\})^2} \left\{ x : \frac{(x - t_j)\%g_{i,j}}{b_j} = \frac{(t_k - x)\%g_{i,k}}{b_i} \right\}$$

To better interpret this set, consider Figure 4.3 where partition i's turn is up and it is the one changing its strategy. Considering two other fixed partitions j and k as shown in the same figure, the strategy of i affects the evolution coefficient between these partitions (i, j and k). In this example, the evolution coefficient is the minimum between $\frac{(t_k-x)\%g_{i,k}}{b_i}$ and $\frac{(x-t_j)\%g_{i,j}}{b_j}$ whose evolutions as a function of i's offset x are represented by the solid red and dashed blue lines respectively. Evidently the best-response is going to lie on an intersection point where every other value of x yields a lower value of $\min(\frac{(t_k-x)\%g_{i,k}}{b_i}, \frac{(x-t_j)\%g_{i,j}}{b_j})$. For the overall best-response, considering only intersection points $\forall (j,k) \in (\Pi \setminus \{i\})^2$ should suffice, this is demonstrated in Theorem 17.

Theorem 17 $\mathcal{S}_i(\mathbf{t}_{-i}) \subset \mathcal{I}_i(\mathbf{t}_{-i}) \subset \mathcal{T}_i$. *In other words, instead of using linear search, it suffices to consider points in* $\mathcal{I}_i(\mathbf{t}_{-i})$ *when computing the best-response of player* i.

Proof. Assume on the contrary that

$$\frac{(t_j - t_i^*)\%g_{i,j}}{b_i} \neq \frac{(t_i^* - t_k)\%g_{i,k}}{b_k}, \quad \forall j,k \neq i, \tag{4.10}$$

where t_i^* denotes one of the best-responses of partition i.

Denote by $\alpha_i(x, t_{-i}) = \min_{k\neq i} d_{ik}(x, t_{-i})$ the response of partition i with its offset set to x.

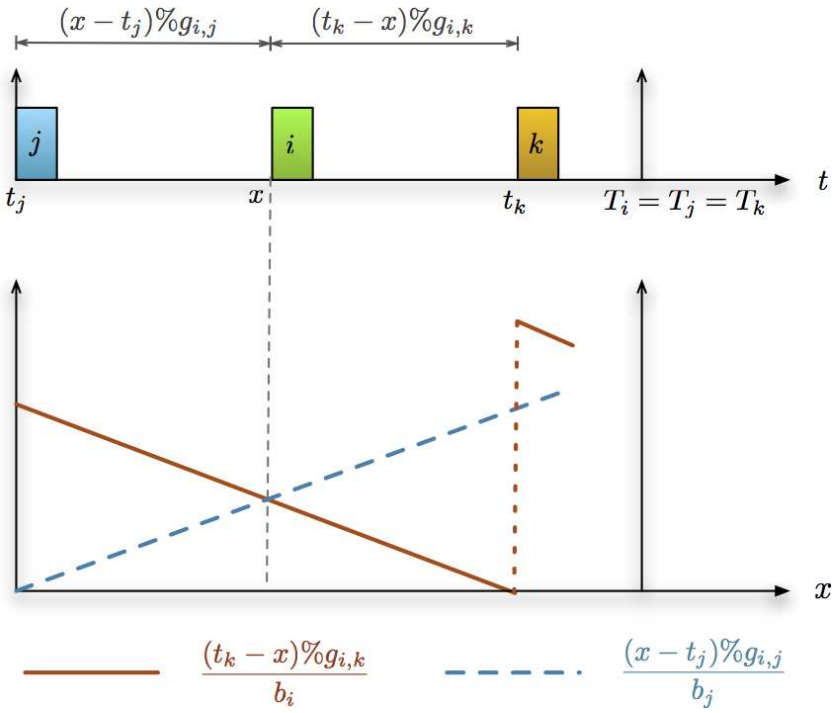


Figure 4.3: Partition i changes its offset in the presence of two other partitions j and k. The best-response for maximizing the evolution factors for partitions i and j is on the intersections between the lines on which these factors evolve.

Let $j \neq i$ be such that $\alpha_i(t_i^*, t_{-i}) = d_{ij}(t_i^*, \mathbf{t}_{-i})$. By assumption, $\frac{(t_j - t_i^*)\%g_{i,j}}{b_i} \neq \frac{(t_i^* - t_j)\%g_{i,j}}{b_j}$. Now assume that $\frac{(t_i^* - t_j)\%g_{i,j}}{b_j}$ is the smaller of the two terms. Thus for all $k \neq i$

$$\frac{(t_i^* - t_j)\%g_{i,j}}{b_j} \leq \frac{(t_i^* - t_k)\%g_{i,k}}{b_k}, \tag{4.11}$$

$$\frac{(t_i^* - t_j)\%g_{i,j}}{b_j} < \frac{(t_k - t_i^*)\%g_{i,k}}{b_i}. \tag{4.12}$$

Equation (4.11) comes from the definition of j as the partition achieving the minimum in (4.2). The strict inequality in (4.12) is obtained using (4.10).

With $k = j$ in (4.12), $(t_i^* - t_j)\%g_{i,j}/b_j < (t_j - t_i^*)\%g_{i,j}/b_i$, which implies that $(t_j - t_i^*)\%g_{i,j} > 0$. With (4.11), it implies that $(t_i^* - t_k)\%g_{i,k} > 0$ for all $k \neq i$ and thus that there exists $q_{ik} \in \mathbb{Z}$ such that $t_i^* \in (t_k + q_{ik}g_{i,k}, t_k + (q_{ik} + 1)g_{i,k})$.

Let $z = \min_{k \neq i} t_k + (q_{ik} + 1)g_{i,k}$. The function $x \to \min_{k \neq i}(x - t_k)\%g_{i,k}/b_k$ is continuous and strictly increasing on the interval (t_i^*, z). Therefore

$$\min_{k\neq i} \frac{(x-t_k)\%g_{i,k}}{b_k} > \frac{(t_i^*-t_j)\%g_{i,j}}{b_j} \quad \forall x \in (t_i^*, z). \tag{4.13}$$

In addition, the function $x \to \min_{k\neq i}(t_k - x)\%g_{i,k}/b_i$ is continuous and strictly decreasing on the interval (t_i^*, z). Thus, (4.12) implies that there exists $\epsilon > 0$ such that

$$\min_{k\neq i} \frac{(t_k-x)\%g_{i,k}}{b_i} > \frac{(t_i^*-t_j)\%g_{i,j}}{b_j} \quad \forall x \in (t_i^*, t_i^*+\epsilon). \tag{4.14}$$

From (4.13) and (4.14), $\alpha_i(x, \mathbf{t}_{-i}) > \alpha_i(t_i^*, \mathbf{t}_{-i})$ can be concluded for all $x \in (t_i^*, t_i^*+\epsilon)$, which is clearly a contradiction.

The proof in the case $(t_j - t_i^*)\%g_{i,j}/b_i < (t_i^* - t_j)\%g_{i,j}/b_j$ is symmetric. ■

Thus, the solution of (SCHD-i) can be obtained by restricting the search over the set $\mathcal{I}_i(\mathbf{t}_{-i})$. Since, the original problem is defined only for integer values of the offsets, the best-response algorithm described in Algorithm 5 will search for best-response only on floor and ceiling integers of points in $\mathcal{I}_i(\mathbf{t}_{-i})$ instead of searching all the T_i points.

4.1.4 Computing the intersection points

An algorithm to compute the elements of $\mathcal{I}_i(\mathbf{t}_{-i})$ is given, and if possible, its cardinality.

An offset $x \in \mathcal{I}_i(\mathbf{t}_{-i})$ satisfies

$$t_j - x = mg_{i,j} + (t_j - x)\%g_{i,j}, \tag{4.15}$$

$$x - t_k = ng_{i,k} + (x - t_k)\%g_{i,k}, \tag{4.16}$$

$$\frac{(t_j-x)\%g_{i,j}}{b_i} = \frac{(x-t_k)\%g_{i,k}}{b_k}, \tag{4.17}$$

for some m and n in $\mathbb{Z}$. From which the following can be concluded

$$t_j - t_k - ((t_j - x)\%g_{i,j})\left(1 + \frac{b_k}{b_i}\right) = lg_{i,j,k}, \tag{4.18}$$

where $g_{i,j,k} = \gcd(g_{i,j}, g_{i,k})$, and for some l in $\mathbb{Z}$.

Since $(t_j - x)\%g_{i,j} \in [0, g_{i,j})$, $(x - t_k)\%g_{i,k} \in [0, g_{i,k})$ and the equality (4.17) is true,

$$(t_j - x)\%g_{i,j} \in \left[0, \min\left(g_{i,j}, g_{i,k}\frac{b_i}{b_k}\right)\right),$$

that is,

$$t_j - t_k - ((t_j - x)\%g_{i,j})\left(1 + \frac{b_i}{b_k}\right) \in (t_j - t_k - \mu_{j,k}, t_j - t_k]\,. \tag{4.19}$$

where $\mu_{j,k} = \min\left(\frac{g_{i,,j}}{b_i}, \frac{g_{i,k}}{b_k}\right)(b_i + b_k)$. From (4.18) and (4.19), $t_j - t_k - \left((t_j - x)\%g_{i,j}\right)\left(1 + \frac{b_i}{b_k}\right)\Big)$ can potentially be any integer multiple of $g_{i,j,k}$ in the interval $(t_j - t_k - \mu_{j,k}, t_j - t_k]$.

Let $L(j,k)$ be the number of multiples of $g_{i,j,k}$ in the interval $(t_j - t_k - \mu_{j,k}, t_j - t_k]$, that is,

$$L(j,k) = \left\lfloor \frac{t_j - t_k}{g_{i,j,k}} \right\rfloor - \left\lfloor \frac{t_j - t_k - \mu_{j,k}}{g_{i,j,k}} \right\rfloor. \tag{4.20}$$

From (4.18) and (4.20),

$$(t_j - x)\%g_{i,j} = \frac{t_j - t_k - l g_{i,j,k}}{1 + \frac{b_k}{b_i}}, \text{ for } l = \left\lfloor \frac{t_j - t_k}{g_{i,j,k}} \right\rfloor, \ldots, \left\lfloor \frac{t_j - t_k - \mu_{j,k}}{g_{i,j,k}} \right\rfloor + 1,$$

that is,

$$(t_j - x)\%g_{i,j} = \frac{(t_j - t_k)\%g_{i,j,k} + l g_{i,j,k}}{1 + \frac{b_k}{b_i}}, \text{ for } l = 0, \ldots, L(j,k) - 1. \tag{4.21}$$

Note that $(t_j - x)\%g_{i,j} < \min\left(g_{i,j}, g_{i,k}\frac{b_i}{b_k}\right)$ as is necessary for (4.17) to be satisfied.

Finally, in order to determine x, the computation of m is needed. For this purpose, let $\hat{m}$ and $\hat{n}$ be the Bézout coefficients of the pair $g_{i,j}$ and $g_{i,k}$, that is

$$\hat{m} g_{i,j} + \hat{n} g_{i,k} = g_{i,j,k}.$$

These coefficients can be determined using the extended Euclid algorithm [40]. Then,

$$l\hat{m} g_{i,j} + l\hat{n} g_{i,k} = l g_{i,j,k}.$$

Let $c_{i,j,k}$ be the least common multiple of $g_{i,j}$ and $g_{i,k}$. From the Bachet-Bézout theorem $m = l\hat{m} + q\frac{c_{i,j,k}}{g_{i,j}}$, for some $q \in \mathbb{Z}$. Thus, for a given l, q is needed in order to compute x.

Let

$$q(l) = \max\{q : t_j - l\hat{m} - q c_{i,j,k} - (t_j - x)\%g_{i,j} > 0\}.$$

That is, $q(l)$ is that value of q that results in the smallest non-negative x that satisfies (4.15) and (4.16). This smallest value of t will be denoted by $\tau(l)$. Thus,

$$\tau(l) = (t_j - l\hat{m} g_{i,j} - (t_j - x)\%g_{i,j})\%c_{i,j,k}.$$

If $\tau(l)$ is an intersection point, then so is $\tau(l) + r c_{i,j,k}$. Thus, for each l, there are $\lfloor \frac{T_i - \tau(l)}{c_{i,j,k}} \rfloor + 1$ values of r for which $\tau(l) + r c_{i,j,k} \in \mathcal{T}_i$ which give the same intersection heights.

Algorithm 6 provides a summary on the above steps for computing the elements of the set $\mathcal{I}_i$.

Algorithm 6 Computing the set of intersection points

Require: : $\mathbf{t}_{-i}$
  $\mathcal{I}_i(\mathbf{t}_{-i}) = \emptyset$
  for all $j \neq i$ **do**
    for all $k \neq i$ **do**
      for $l = \left\lfloor \frac{t_j - t_k - \min\left(\frac{g_{i,j}}{b_i}, \frac{g_{i,k}}{b_k}\right)(b_i + b_k)}{g_{i,j,k}} \right\rfloor + 1$ to $\left\lfloor \frac{t_j - t_k}{g_{i,j,k}} \right\rfloor$ **do**
        $\tau(l) = (t_j - l\hat{m}g_{i,j} - \frac{t_j - t_k - lg_{i,j,k}}{1 + \frac{b_k}{b_i}}) \% c_{i,j,k}$
        for $r = 0$ to $\left\lfloor \frac{T_i - \tau(l)}{c_{i,j,k}} \right\rfloor$ **do**
          $\mathcal{I}_i(\mathbf{t}_{-i}) \leftarrow \mathcal{I}_i(\mathbf{t}_{-i}) \cup (\tau(l) + rc_{i,j,k})$
        end for
      end for
    end for
  end for

An upper bound on the cardinality of $\mathcal{I}_i(\mathbf{t}_{-i})$ can be computed as

$$|\mathcal{I}_i(\mathbf{t}_{-i})| \leq \sum_{j \neq i} \sum_{k \neq i} \frac{\min\left(\frac{g_{i,j}}{b_i}, \frac{g_{i,k}}{b_k}\right)(b_i + b_k)}{g_{i,j,k}} \frac{T_i}{c_{i,j,k}}$$
$$= \sum_{j \neq i} \sum_{k \neq i} \frac{\min\left(\frac{g_{i,j}}{b_i}, \frac{g_{i,k}}{b_k}\right)(b_i + b_k) T_i}{g_{i,j} g_{i,k}}$$

If it is assumed that the partitions are at least schedulable in pairs, that is $(b_i + b_j) \leq g_{i,j}, \forall i, j$, then a sufficient condition for this algorithm to check fewer points than linear search is $b_i > N^2$, where N is the total number of partitions.

Remark 18 *When the offsets are restricted to integers, several intersection points between two integers could potentially be computed. And, for all these intersection points the best-response algorithm shall check for only the two integers closest to them. Although, theoretically, there could be instances where there could be more intersection points than the period of the partition, in practice, this will not have much impact on the efficiency of the best-response algorithm.*

4.2 Multiprocessor Scheduling

In this section, the best response algorithm is further extended to the multiprocessor scheduling problem. The schedule is now not only represented by the temporal scheduling of partitions $\mathbf{t}$, but also by the processing module allocation for each of the partitions, represented by the binary vector $\mathbf{a}$. The associated MILP formulation for the problem can be found in Section 3.2.4.

Again the main merit here of the best-response algorithm is, as was the case in the uniprocessor setting, the ability to solve quite large instances for reasonable amounts of time and reduced relative errors with respect to optimal solutions. Furthermore, as was already noted, the algorithm is not restricted to harmonic periods, and hence allow further flexibility with partition period definition.

In the multiprocessor setting, at its turn a partition sequentially computes its best response on each of the processors. It then selects the processor which improves its current evolution margin the most. Evidently, a partition's strategy is now closely related to not only its offset, but also its allocation to a module. Just as in the uniprocessor case, it shall be assumed that a partition does not switch its module and change its offset unless it can strictly improve its evolution margin (or, its utility).

Following the definition (4.2), for the multiprocessor setting define

$$\alpha_i^n = \min_{\{j \neq i, a_{i,k}^n = a_{j,k}^n \forall k\}} d_{i,j}(\mathbf{t}^n), \tag{4.22}$$

which is the evolution margin of the partition i with respect to those that are scheduled on the same module as itself for a given offset vector $\mathbf{t}^n$ and allocation vector $\mathbf{a}^n$. Also, define $\mathbf{a}_i^n$ to be the allocation vector of partition i after the nth iteration.

The pseudocode for the multiprocessor best-response algorithm is given in Algorithm 7. The condition in step 7 verifies if it possible to introduce partition i on module k (if previously $a_{i,k}^n = 0$), that is to say that the system resource constraints are satisfied ($\mathcal{A}$ and Θ from Section 3.2).

Lemma 19 *The results demonstrated in Lemma 8 are still applicable in the multiprocessor case. In other words, assuming at iteration n player i updates its strategy, the following are still valid $\forall j \neq i$, and whatever the allocation.*

$$\alpha_j^n < \alpha_i^n \Rightarrow \alpha_j^{n+1} = \alpha_j^n \tag{4.23}$$

$$\alpha_j^n = \alpha_i^n \Rightarrow \alpha_j^{n+1} \geq \alpha_j^n \tag{4.24}$$

$$\alpha_j^n > \alpha_i^n \Rightarrow \alpha_j^{n+1} > \alpha_i^n \tag{4.25}$$

Proof. For the sake of simplicity, $d_{i,j}$ can not be written unless i and j occupy the same module. Let E^n denote the event that players i and j occupy the same module at the end of iteration n. Subsequently, $\overline{E}^n$ denotes the event of i and j being on different modules. Assuming i changes its strategy ($\alpha_i^{n+1} > \alpha_i^n$ and $t_j^{n+1} = t_j^n \, \forall j \neq i$), the following events are possible $\forall j \neq i$,

1. $E^n \cap E^{n+1}$: in this case the proof is similar to that of the uniprocessor setting.

2. $E^n \cap \overline{E}^{n+1}$: player i was on the same module as j but then changed its allocation. In this case,

$$\alpha_j^{n+1} = \min_{k \neq i,j} d_{jk}^{n+1} = \min_{k \neq i,j} d_{jk}^n \geq \min\left(\min_{k \neq i,j} d_{jk}^n, d_{ij}^n\right) = \alpha_j^n.$$

   If $\alpha_j^n < \alpha_i^n$, then since $d_{ij}^n \geq \alpha_i^n > \alpha_j^n = \min\left(\min_{k \neq i,j} d_{jk}^n, d_{ij}^n\right)$, then $\alpha_j^n = \min_{k \neq i,j} d_{jk}^n = \alpha_j^{n+1}$.

Algorithm 7 Multiprocessor best-response

Require: $\mathbf{t}^0, \mathbf{a}^0$
$n \leftarrow 0$
repeat
  for $i = 1$ to N **do**
    if i = n % N + 1 **then**
      $\mathbf{a}_i^{n+1} \leftarrow \mathbf{0}$
      for $k = 1$ to M **do**
        if $\mathbf{a}_i^{n+1} \cup \mathbf{a}_{-i}^{n+1} \in \mathcal{A} \cap \Theta$ for $a_{i,k}^{n+1} = 1$ **then**
          $z \leftarrow \max_x \min_{\{j:j\neq i, a_{j,k}=1\}} d_{i,j}(x, \mathbf{t}_{-i}^n)$
          if $z > \alpha_i^n$ **then**
            $\alpha_i^n \leftarrow z$
            $c \leftarrow k$
            $t_i^{n+1} \leftarrow \min \operatorname{argmax} \min_{\{j:j\neq i, a_{j,k}=1\}} d_{i,j}(x, \mathbf{t}_{-i}^n)$
          end if
        end if
      end for
      $a_{i,c}^{n+1} = 1$
    else
      $t_i^{n+1} \leftarrow t_i^n$
      $\mathbf{a}_i^{n+1} \leftarrow \mathbf{a}_i^n$
    end if
  end for
  $n \leftarrow n + 1$
until $\mathbf{t}^n = \mathbf{t}^{n-N}$ **and** $\mathbf{a}^n = \mathbf{a}^{n-N}$.
return $\mathbf{t}^n$

If $\alpha_j^n = \alpha_i^n$, then as defined above $\alpha_j^{n+1} \geq \alpha_j^n$.

Finally, if $\alpha_j^n > \alpha_i^n$, then $\alpha_j^{n+1} \geq \alpha_j^n > \alpha_i^n$.

3. $\overline{E}^n$ and $\overline{E}^{n+1}$: the module on which j is found is not modified, and hence systematically the equality is satisfied in (4.23) and (4.24). If however $\alpha_j^n > \alpha_i^n$, then $\alpha_j^{n+1} > \alpha_i^n$, since the utility of j has not changed.

4. $\overline{E}^n$ and E^{n+1}: at the end of the $(n+1)$th iteration, i entered j's module on which it was not before. If $\alpha_j^n < \alpha_i^n$, then $\alpha_j^{n+1} = \alpha_j^n$ due to the following

$$d_{ij}^{n+1} \geq \alpha_i^{n+1} > \alpha_i^n > \alpha_j^n = \min_{k \neq i,j} d_{jk}^n,$$

leading to $\alpha_j^{n+1} = \min_{k \neq i,j} d_{jk}^n$.

If $\alpha_j^n = \alpha_i^n$, then the equality in (4.24) ($\alpha_j^{n+1} \geq \alpha_j^n$) is satisfied since

$$d_{ij}^{n+1} \geq \alpha_i^{n+1} > \alpha_i^n = \alpha_j^n = \min_{k \neq i,j} d_{jk}^n,$$

and hence $\alpha_j^{n+1} = \min\left(\min_{k \neq i,j} d_{ij}^n, d_{ij}^{n+1}\right) = \min_{k \neq i,j} d_{kj}^n = \alpha_j^n$.

Finally, if $\alpha_j^n > \alpha_i^n$, then from $d_{ij}^{n+1} \geq \alpha_i^{n+1} > \alpha_i^n$ and $\min_{k \neq i,j} d_{jk}^{n+1} = \min_{k \neq i,j} d_{jk}^n = \alpha_j^n > \alpha_i^n$ then

$$\alpha_j^{n+1} = \min\left(\min_{k \neq i,j} d_{jk}^{n+1}, d_{ij}^{n+1}\right) > \alpha_i^n.$$

■

Subsequent results can be demonstrated, though not presented hereafter, as was done in the uniprocessor case. The following theorem states two important properties of the best-response algorithm that are also valid in the multiprocessor case.

Theorem 20

1. *The multiprocessor best-response algorithm converges*.
2. *There exists at least one equilibrium point that is optimal*.

Proof. The proofs of the two properties are similar to those of the uniprocessor case. ■

It should be also stated that the upper bound on the number of required iterations remains valid.

4.2.1 Initial allocation in the multiprocessor setting

In the uniprocessor scheduling, an initial schedule can be obtained easily, e.g. randomly fixing partition offsets. In the multiprocessor case however, randomly choosing an offset and allocation vectors does not guarantee respecting resource constraints.

A manner to obtain an initial schedule ($\mathbf{t}^0$ and $\mathbf{a}^0$) is using a greedy algorithm (*cf.* Section 2.6.2.2). In an implementation of this greedy algorithm, partitions are ordered, e.g. in the decreasing order of $\frac{b_i}{T_i}$, and then one by one, each partition is introduced and allocated the module on which it has its best-response. In this manner, partition are only placed on modules where the resource constraints are respected.

This greedy algorithm gives a fairly good starting point when resource constraints are somehow flexible. In case these constraints are tight, this algorithm might fail. A solution for this problem is the utilization of linear formulations for obtaining a feasible allocation, that is the computation of an allocation vector $\mathbf{a}^0$. The offset vector $\mathbf{t}^0$ can then be chosen at random. The formulation is as follows,

$$\text{maximize} \sum_{i\in\Pi, k\in\mathcal{P}} a_{i,k} \tag{4.26}$$

subject to

$$\sum_{k\in\mathcal{P}} a_{i,k} \leq 1, \qquad \forall i \in \Pi, \tag{4.27}$$

$$\sum_{i\in\Pi} a_{i,k} \leq H_k, \qquad \forall k \in \mathcal{P}, \tag{4.28}$$

$$\sum_{i\in\Pi} a_{i,k}\, m_i \leq M_k, \qquad \forall k \in \mathcal{P}. \tag{4.29}$$

The objective in (4.26) along with constraints (4.27) maximizes the number of partition allocations to modules. If the objective is not equivalent to the total number of partitions, then no feasible schedule, including all partitions, exists. Constraints (4.28) and (4.29) prevent exceeding the memory and partition capacities of modules.

The communication constraints are not included so as to keep the presentation clear and simple. Their addition however should not be of great concern, as the constraints of Section 3.2.3 can be easily incorporated. Furthermore, in the best-response algorithm, a partition is prohibited from changing its strategy if delay constraints for the processing chains are violated.

4.3 Multi-start with bayesian stopping rules

In both the uniprocessor and multiprocessor settings, the quality of the algorithm's solution greatly depends on the starting point (initial schedule), from which partitions change strategies till an equilibrium is reached. In order to augment the probabilities of finding an optimal solution, multi-start methods [115] can be performed. The aim of multi-start is the discovery, up to a certain extent, of the majority of equilibrium points or regions of attraction in the system by departing from several starting

points. A region of attraction is defined as the set of states that, when applied as starting points for an optimization algorithm, lead to a unique local optimum (equilibrium in the best-response algorithm). This allows enhancing the proposed best-response algorithm and give some statistical information on the obtained results. Bayesian stopping rules [30] are used to stop the multi-start procedure where several runs are made, one after the other. It should be noted that important results from [30] are hereafter presented without going into much details and proofs.

Let the function f denote the best-response algorithm which, starting from an initial sample (offset and allocation vectors $\mathbf{t}^0$ and $\mathbf{a}^0$) leads to an equilibrium characterized by $\mathbf{t}^*$ and $\mathbf{a}^*$. The local minima concept in [30] is hence replaced by the notion of equilibrium points. The methods represented in this section do not guarantee finding the optimal solution, but assuring a probability for this event that approaches 1 as the sample size goes to infinity (given that the real number of equilibria is not known). Stopping rules are applied to give a trade-off between reliability and computational effort. In other words, additional samples (starting points) are considered as long as a given stopping criterion is not satisfied. In this section consider the following notations:

- K: a random variable representing the real number of equilibria k.
- W: a random variable representing the observed number of equilibria w.
- S: a random variable representing the number of considered samples s.
- S_i: a random variable representing the occurrence of the i-th equilibrium.
- Ω: a random variable representing the total relative volume of the observed regions of attraction.
- Φ_i: a random variable representing the relative volume $\phi_i (i = 1, \ldots, k)$ for the i-th region of attraction.
- $\mathcal{S}$: the set of samples, i.e. all possible starting points.

In practice, $k, \phi_1, \ldots, \phi_k$ are always unknown. The sampled equilibria, however, clearly provide information about the values of these parameters. Since the starting points of Multi-start are uniformly distributed over $\mathcal{S}$, the i-th equilibrium point at each trial has a fixed probability of being found that is equal to the relative volume ϕ_i of its region of attraction. In order to apply Bayesian stopping rules, it is first assumed that the number of equilibria K is a priori equiprobable on $[1, \infty)$. Given $K = k$, it is also assumed that the relative sizes of the regions of attraction $\Phi_1, \ldots, \Phi_K$ follow a uniform distribution on the $(k-1)$-dimensional unit simplex

$$I_{k-1} = \left\{ (\phi_1, \ldots, \phi_k) | \phi_i \geq 0 (i = 1, \ldots, k), \sum_{i=1}^{k} \phi_i = 1 \right\}.$$

Hence the joint prior probability density function is given by

$$p(k, \phi_1, \ldots, \phi_k) \propto (k-1)!.$$

The following are some consequent results arising from Theorem 1 in [30],

- **Posterior expected value of the number of equilibria:**

$$E(K|\{s_1,\ldots,s_w\}) = \frac{w(s-1)}{s-w-2} \quad (s \geq w+3). \tag{4.30}$$

- **Posterior expected value of the relative volume of a region of attraction of an equilibrium point which has been found s_j times:**

$$E(\Phi_{s_j}|\{s_1,\ldots,s_w\}) = \frac{(s_j+1)(s+w)}{s(s-1)} \quad (s \geq w+2). \tag{4.31}$$

- **Posterior expected value of the total volume of the observed regions of attraction:**

$$E(\Omega|\{s_1,\ldots,s_w\}) = \frac{(s-w-1)(s+w)}{s(s-1)} \quad (s \geq w+2). \tag{4.32}$$

4.3.1 Stopping rules

The stopping rules indicate when to stop considering additional samples to perform more runs. A simple rule might be as follows,

$$E(K|\{s_1,\ldots,s_w\}) - \frac{1}{2} \leq w,$$

i.e. to stop if the optimal integer Bayesian estimate of the unknown number of equilibria is equal to the number of distinct equilibria observed. If one is not willing to continue the search for all equilibria, an appropriate stopping criterion may be to terminate the algorithm if the total relative volume of observed regions of attractions exceeds a prescribed value $t\,(0 < t < 1)$, that is,

$$E(\Omega|\{s_1,\ldots,s_w\}) \geq t.$$

It is preferable, however, to apply sequential stopping rules which also take into account the cost of the sampling. To do so, one has to impute a *termination loss* which attaches costs to the deviation of an estimated unknown quantity from its true value, as well as an *execution loss* corresponding to the cost of further experiments.

Consider hereafter two loss structures presented in [30] in order to minimize the *expected posterior loss* [52].

1. The termination loss is equal to a fixed constant c_1 if sampling is stopped before all minima have been discovered and 0 otherwise:

$$L_1 = \begin{cases} s + c_1 & \text{if } K > W, \\ s & \text{if } K = W. \end{cases}$$

Given $\{s_1,\ldots,s_w\}$ the posterior loss is equal to

$$E(L_1|\{s_1,\ldots,s_w\}) = c_1\left(1 - \prod_{i=1}^{w}\frac{s-1-i}{s-1+i}\right) + s. \tag{4.33}$$

This termination loss can only decrease if ultimately all equilibria are found.

2. The termination loss is proportional to the total relative volume of the unobserved regions of attraction:

$$L_2 = c_2(1 - \Omega) + s,$$

where c_2 is a predefined constant.

The posterior loss is equal to

$$E(L2|\{s_1, \ldots, s_w\}) = c_2 \frac{w(w+1)}{s(s-1)} + s. \tag{4.34}$$

This termination loss reflects that case where one is less interested in finding local minima with exremely small regions of attraction.

In the following $\{s_1, \ldots, s_w\}$ is replaced by (s, w).

Given a sample size s, the posterior loss after $s' > s$ observations is a random variable $E(L_j|(s', W))$. The purpose, then, is to find for each loss function the stopping rule that minimizes the expected sequence of posterior losses $\{E(L_j|(s', W))\}_{s'=s+1}^{\infty}$. Given a current pair (s, w) only two relevant outcomes can occur as a result of an additional local search: $(s, w) \rightarrow (s+1, w+1)$ or $(s, w) \rightarrow (s+1, w)$. The posterior probability that the next search will not result in the discovery of an unobserved equilibrium is equal to the posterior expected volume of the observed regions of attraction. Hence the sequence of posterior losses satisfies the recurrence relation [30]:

$$\begin{aligned} E(E(L_j|(s+1, W))|(s, w)) =& E(\Omega|(s, w))E(L_j|(s+1, w)) \\ &+ (1 - E(\Omega|(s, w)))E(L_j|(s+1, w+1)) \\ =& \frac{(s-w-1)(s+w)}{s(s-1)} E(L_j|(s+1, w)) \\ &+ \frac{w(w+1)}{s(s-1)} E(L_j|(s+1, w+1)). \end{aligned} \tag{4.35}$$

Therefore an appealing one-step stopping rule is to terminate if this conditional expected posterior loss of $s+1$ observations is greater than the current posterior loss $E(L_j|(s, w))$. This decision is based on the assumption that the search is immediately stopped once the next observation would have been performed.

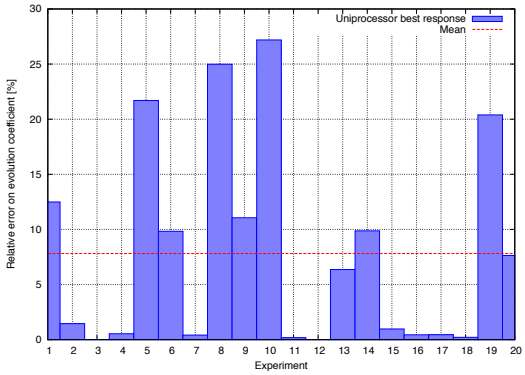
4.4 Results

In this section some experimentations to evaluate the performance of the best-response algorithms are demonstrated. Again, the exact MILP formulations for both the uniprocessor and multiprocessor problems are solved using the linear program solver CPLEX [81]. The machine used is based on an Intel® Core™2 Quad CPU Q6700 @ 2.66GHz with 4MB of cache and 4GB of system memory.

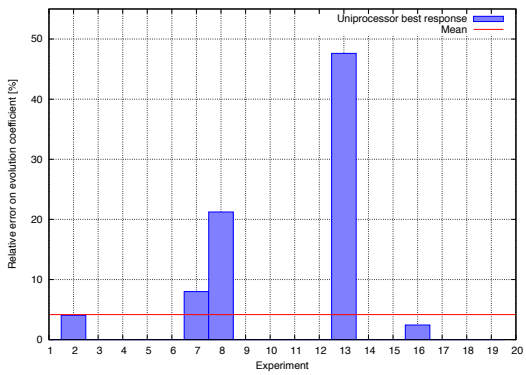
For example generation, unlike the earlier chapter, harmonic (H) periods were chosen uniformly from the set $\{1500, 3000, 6000, 12000, 24000\}$, whereas for non-harmonic (NH) periods partitions were

uniformly allocated one of five periods chosen from the set $\{2^x\, 3^y\, 50 : x \in [0,4], y \in [0,3]\}$, as was inspired from [56]. For partition time budgets, it was generated following an exponential distribution and averaging at about 20% of the partition's period. Memory and partition capacities for modules and memory requirements for partitions were generated so as not to greatly constrain the problem, in other words, they were generated to limit the modules with a twenty partition limit. The influence of memory resources on the scheduling problem is investigated in the benchmark of Section 4.4.1. Finally, among the architectural constraints (for multiprocessor problems), only partition exclusions were considered for the generated examples, i.e. couples of partitions that cannot execute on the same processor. In this case, 40% of partitions were chosen uniformly to be implicated in an exclusion constraint. Unless indicated otherwise, each experiment is run once starting from an initial solution supplied by the greedy algorithm discussed in Section 4.2.1.

The algorithm's performance is first evaluated for the uniprocessor setting (single run for each instance). Twenty uniprocessor examples, with fifteen partitions each, were generated with harmonic and non-harmonic periods respectively. As can be seen in Figure 4.4, the relative error on the optimized evolution coefficient α averaged at about 7.8% for the harmonic case and 4.2% for the non-harmonic one. This is interesting given that the average execution time for the best response algorithm was around 2.7 seconds versus more than 20 minutes for the exact method. The relatively high relative error for instance 13 in Figure 4.4b arises from the considered starting point (using greedy algorithm), as will be seen in Section 4.4.2.



(a) Harmonic periods.



(b) Non-harmonic periods.

Figure 4.4: Relative error on α for uniprocessor examples.

A particular situation arises when the problem becomes more complex and the exact method fails to prove optimality in an adequate amount of time. Consider for instance a non-harmonic example constituted of 20 partitions whose temporal attributes are indicated in Table 4.1–these are not generated as mentioned earlier but randomly with 12 different periods–, partition time budgets remain small compared to their periods.

For this example the best response algorithm gave $\alpha = 1.41$ in 2.83 seconds where the exact method failed to supply a value superior to $\alpha = 1.11$ within a one hour limit. In this case, the benefit of the

Table 4.1: A uniprocessor example with 20 partitions of general non-harmonic periods. LCM between periods is 756000

Partition	1	2	3	4	5	6	7	8	9
Time budget	10	30	30	10	10	10	10	10	30
Period	1200	1200	3600	1200	1200	1500	4200	1000	2000

10	11	12	13	14	15	16	17	18	19	20
10	10	45	40	40	60	80	30	10	60	40
4000	1200	2400	2000	4000	3000	3000	2700	200	1800	1800

proposed algorithm can be noticed. It is true that the final optimal solution may be greater than the obtained value but one has to take into account the time cost required to do so.

Another advantage of the best-response algorithm is the ability to obtain better mean evolution per partition, where unlike the exact method, the best-response algorithm does not stop until all partition strategies are unchanged. To illustrate this consider a uniprocessor example where the solution was $\alpha = 1.57$ (minimum corresponding to partition 12) for both methods. Figure 4.5 represents the evolution coefficient for each of the 15 partitions, as a result of both methods. The average evolution per partition is at 1.98 for the exact method versus 2.58 for the proposed algorithm.

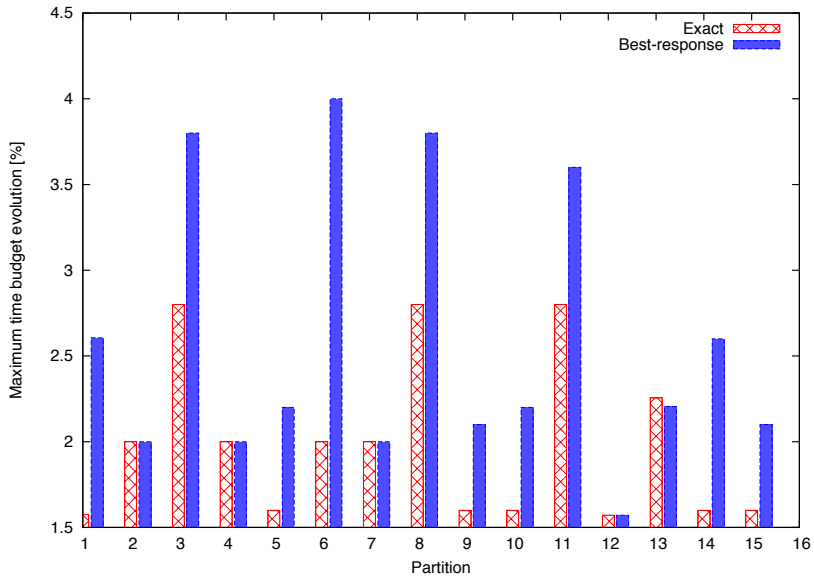


Figure 4.5: Evolution margin per partition for an example with 15 partitions.

For the multiprocessor scheduling problem, 4 experiment sets, each consisting of 100 instances, were generated. Instances of the first and second sets are constituted of 4 modules and 20 partitions with harmonic and non-harmonic periods respectively. Instances of the third and fourth sets are composed of 4 modules and 40 partitions with harmonic and non-harmonic periods respectively. The attributes were generated as indicated in the beginning of this section.

When solving the multiprocessor problem to optimality using the exact MILP formulation, a time limit of thirty minutes was defined for instances of the first and second sets. As for the third and fourth sets, this limit was increased to one hour and partition offsets were considered continuous ($t_i \in \mathbb{R}$). This decreases execution times significantly (*cf.* Table 3.1). To limit the minor difference in the optimal solution, all temporal attributes were divided by 100. For what concerns the proposed best-response algorithm, the instances were not modified and no time limit was imposed.

Table 4.2 demonstrates the relative error on the evolution coefficient α for the multiprocessor best-response algorithm with respect to the exact method. Execution times for the exact method are not included as it was already shown that exceeding certain limits, they may be inconvenient. In fact, most of the runs exceeded the time limit and hence were stopped. The solutions were nevertheless used for comparison. It is obvious that execution times as well as the quality of obtained solutions for the best-response algorithm are quite interesting. For all of the considered examples, harmonic and non-harmonic, mean relative error on the evolution coefficient α, as compared to the exact solution, remains below 20%. For the third set '4M40P-H', the relative error averages at about 8%, the reason why this average is lower than that of '4M20P-H' is that the exact method often required more than one hour to reach or even prove optimality, and hence, within this time limit, α obtained by the best response algorithm was superior to that of the exact method. The same analysis applies to '4M40P-NH'.

Table 4.2: Relative error on optimality for the multiprocessor best response algorithm. xMyP designates instances with x modules and y partitions.

Experiment set	Nature of periods	Mean relative error on α	Mean CpuTime
4M20P-H	harmonic	14.35%	0.65s.
4M20P-NH	non-harmonic	18.13%	1.04s.
4M40P-H	harmonic	7.91%	6.64s.
4M40P-NH	non-harmonic	10.58%	12.14s.

4.4.1 Large scale and industrial applications

The best-response algorithm is hereafter tested on examples depicting real world scenarios. In avionics, three types of partitions and modules actually exist, without discussing the nomenclature and side locations (modules are distributed between two sides), this leads to the definition of three distinct scheduling problems, one per type. Therefore, three sets of problems depicting those on an air-plane were generated. A set composed of 6 modules and 36 partitions '6M36P', another of 6 modules and 79 partitions

‘6M79P’ and a last one composed of 24 modules and 419 partitions ‘24M419P’. All periods were considered harmonic and temporal attributes were generated as before. Exclusions between partitions were neglected for these three sets. Table 4.3 demonstrates the results obtained.

Table 4.3: Three scheduling problems with sizes similar to those in avionic partition scheduling problems

Problem	Evolution coefficient α	CpuTime
6M36P	3.15	1.18s.
6M79P	1.48	37.48s.
24M419P	1.17	34.485mn.

Unfortunately, comparison with the exact formulation was not possible due to its limitations. The quality of an obtained solution has to be assessed based on its utility, a value of $\alpha \geq 1$ implies in essence a feasible schedule. Any greater value gives more flexibility when increasing time budgets. Execution times are nevertheless convenient, the largest problem with 419 partitions taking about one half of an hour.

The utility of the algorithm with a benchmark supplied by one of the industrial partners of the project is further investigated. An example architecture constituted of three sub-problems were supplied, the first of 12 modules and 36 partitions ‘12M36P’, the second of 12 modules and 112 partitions ‘12M112P’ and the last one of 48 modules and 636 partitions ‘48M636P’. Partition periods belong to the set $\{60000, 120000, 240000\}$ (in μseconds). Module segregation constraints (exclusions) were also supplied, totalling 176 constraints (implicating partitions of the architecture). Additional resource attributes were also supplied (e.g. RAM, NVM, etc.). Table 4.4 demonstrates the obtained results.

Table 4.4: Three scheduling problems supplied as a benchmark

Problem	Evolution coefficient α	CpuTime
12M36P	10	0.06s.
12M112P	3.12	4.53s.
48M636P	1.56	16.83mn.

The results appear to be, as before, convenient. A difference in execution times between ‘48M636P’ at 16 minutes and the previous one ‘24M419P’ (Table 4.3) at 34 minutes can be noticed. One reason can be related to the fact that in ‘48M636P’ the number of partitions per module averages at about 13 whereas in ‘24M419P’ this average is 17, and thus less computation per module in ‘48M636P’. The decreased number of distinct periods and simpler temporal attributes is another factor that may have played a role in decreasing the number of iterations required for the algorithm to converge.

Another experimentation carried out was the division of temporal attributes (periods and time budgets) by 1000, as usually a precision in the order of milliseconds should suffice. Surprisingly the algorithm converged for '48M636P' in about 3 minutes. This is caused from reducing the precision on the time horizon and hence reducing several computations throughout the algorithm.

For what concerns the effect of resources on the initial solution (starting point), a similar benchmark was also supplied but with more strict resources. The greedy algorithm failed to supply an initial allocation, and hence, in this case the exact formulation proposed in Section 4.2.1 is applied. The proposition of an initial allocation took less than one second, after which the algorithm continues as before.

4.4.2 Multi-start results

The multi-start method discussed in Section 4.3 is investigated for some of the experimentations. The Bayesian stopping rule in which the termination loss is related to total relative volume of the unobserved regions of attraction was used. This rule was shown to be the fastest in [30].

In the uniprocessor setting, partition offsets were generated uniformly ($t_i \in \mathcal{T}_i$) to form sample solutions. For each sample point the algorithm was run to obtain an equilibrium specified by the system evolution margin α. Interestingly applying the above indicated Bayesian stopping rule lead to a 0.25% and 0% average relative error on α for the examples of Figures 4.4a and 4.4b respectively, along with a mean execution time of 3 minutes which remains faster than the exact method. This means that the multi-start method succeeded in finding an optimal equilibrium in most of the cases. As indicated earlier, the elevated relative error for instance 13 in Figure 4.4b was due to the consideration of a bad starting point.

To demonstrate the difference between both termination losses, Figure 4.6 represents, for a given instance of the uniprocessor examples, the number of discovered and estimated number of equilibria as more runs are performed. The stopping rule related to the regions of attractions stopped the procedure after 37 runs (5 major equilibria in 96 seconds), whereas the one related to finding all equilibrium points stopped the procedure after 289 runs (12 equilibria in 743 seconds). Both found an optimal equilibrium point, with the difference that the former stopped at 5 equilibrium points with a 97.74% estimated total relative volume for the regions of attraction. This means that in case there are a colossal number of equilibria, stopping after the discovery of a sufficient total region of attraction might be more appropriate.

In the multiprocessor setting, partition offsets and module allocations were initialised uniformly in a manner respecting the various resource constraints (e.g. memory). Fifty instances of 4M20P-H and 4M20P-NH from Table 4.2 were considered. The single runs for these two sets of instances gave a mean relative error of 16.28% and 17.64% respectively, as compared to the exact method. The application of the aforementioned multi-start method reduced this relative error to 0.58% and 0.09% respectively in about 15 minutes. The relative error distributions are represented in Figure 4.7. For some instances the Bayesian stopping rule guaranteed discovering almost all of the regions of attraction, such as instance 20 of Figure 4.7b where a 99.8% total volume for the regions of attraction was estimated. Plotting the

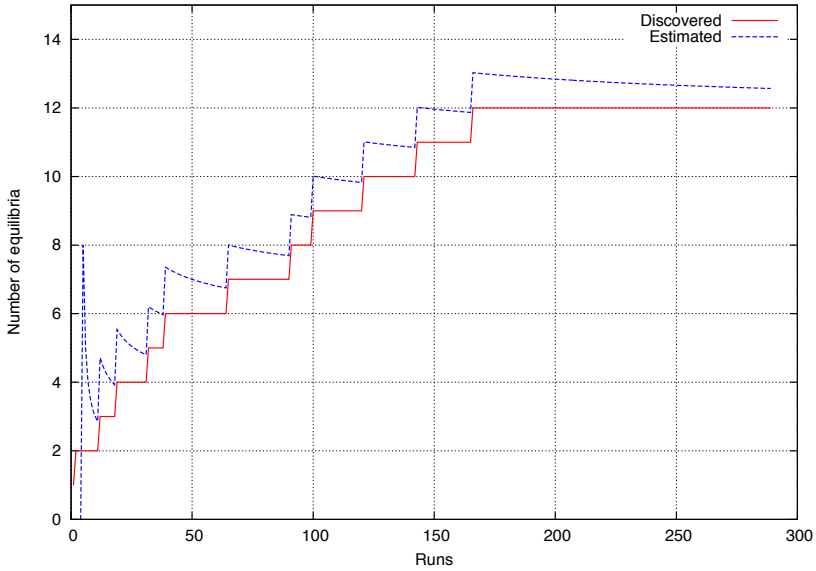
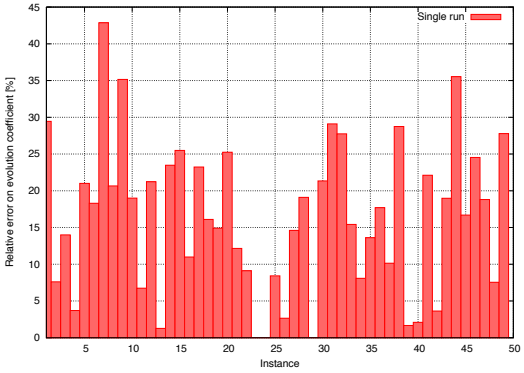


Figure 4.6: Evolution of the discovered and estimated number of equilibria for a uniprocessor instance.

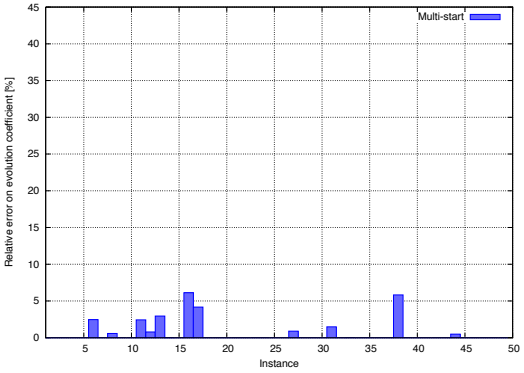
evolution of the estimated and discovered number of equilibria showed that almost all of equilibrium points were discovered (*cf.* Figure 4.8a). However, for instance 16 of Figure 4.7b, the runs where stopped after a 97% total estimated volume for the regions of attraction while providing a solution at about 6% relative error. As can be seen in Figure 4.8b, this is due to the fact that, although a relatively high probability for the discovered regions of attraction was estimated, several equilibrium points lying in small regions remain to be discovered. This can be overcome by continuing the runs until the estimated and discovered number of equilibria converge.

Finally, for the benchmark presented in Table 4.4, the application of the multi-start method on '48M636P' yielded an ameliorated solution at $\alpha = 2.0833$ instead of $\alpha = 1.56$. This solution was discovered in early runs but the method terminated after about 3 days with a total estimated volume of the regions of attraction that was greater than 99%. It should be noted that the lower bound on the number of required modules is 19 as inferred from $\sum_{i\in\Pi} \frac{b_i}{T_i}$. Hence dividing the number of actual modules with this lower bound gives an upper bound on the evolution coefficient, that is $\alpha = 2.564$. In other words, if all time budgets are multiplied by this coefficient, the best-case scenario is to to have a schedule with 100% loads on the 48 modules.

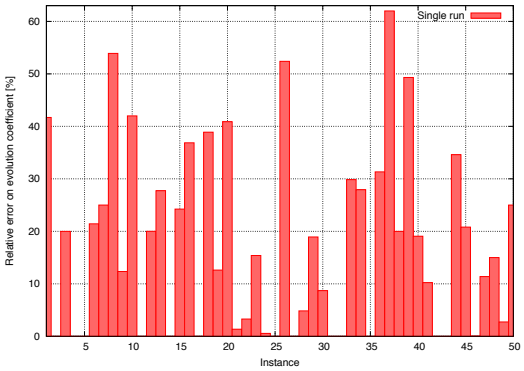
As a conclusion on the application of multi-start methods, and although a single run of the best-response algorithm yields interesting results in most of the cases, the solution quality can be ameliorated through further exploration of the solution space. As was noticed, optimal solutions are not always guaranteed to be found. Nevertheless, some probabilistic measures on the optimality of discovered solutions can be supplied, such as an estimation on the number of equilibria that remain undiscovered. In all, the application of multi-start methods appeared extremely beneficial given the associated computation times.



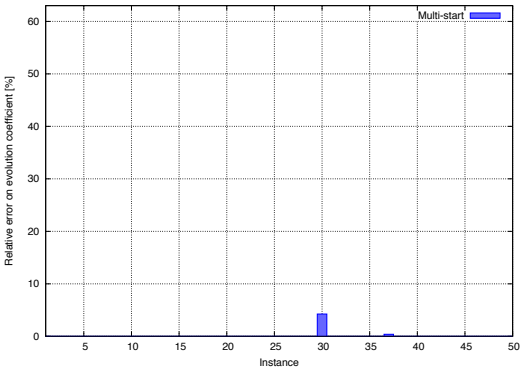
(a) Single runs for 50 instances of 4M20P-H.



(b) Multi-start on the 50 instances of 4M20P-H.



(c) Single runs for 50 instances of 4M20P-NH.



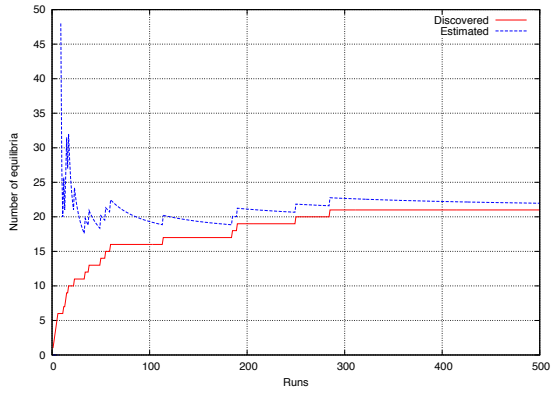
(d) Multi-start on the 50 instances of 4M20P-NH.

Figure 4.7: The application of multi-start methods on multiprocessor examples.

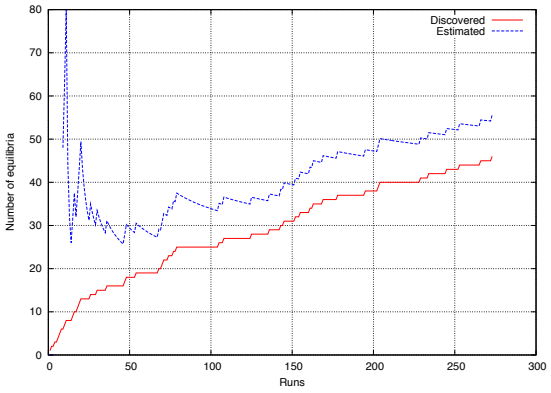
4.4.3 Processor minimization context

The adaptability of the best-response algorithm, in a module minimization context, is superficially demonstrated. Instead of seeking a maximum evolution coefficient on predefined modules, the problem is transformed into minimizing the number of required modules for scheduling the partitions, i.e. determine minimum P such that $\alpha \geq 1$. Without indulging into much detail, the modified best-response algorithm (MBR) starts with one module and tries to obtain $\alpha \geq 1$. If it fails, an extra module is activated and the procedure is redone. The algorithm terminates with P modules and an evolution coefficient $\alpha \geq 1$. The algorithm is compared to a fast ILP-based *Bin-Formulation* [56] (BF), and the approximation algorithm proposed in [91] for constrained processor scheduling (KS). Since BF is limited to harmonic periods, the comparison is only demonstrated for harmonic examples.

Three experiment sets, each consisting of 100 instances, where generated. Instances of the 3 sets are constituted of 20, 40 and 60 partitions respectively with harmonic periods and with no exclusion constraints defined. The various attributes were generated as indicated previously.



(a) An instance where almost all equilibrium points were discovered.



(b) An instance where several equilibrium points remain to be discovered.

Figure 4.8: Evolution of the discovered and estimated number of equilibria for multiprocessor instances.

Table 4.5 represents the optimality of the three methods, in addition to their respective computation times. Since BF is an exact formulation, a solution supplied within a 30 minutes time limit is specified as optimal. For the other two methods, their optimality is determined from BF whenever it is considered optimal (does not exceed time limit).

Table 4.5: Comparison between MBR, BF and KS.

Experiment set	Optimality			Average CpuTime		
	BF	MBR	KS	BF	MBR	KS
20P	100%	67%	42%	0.08s.	2.38s.	0.04s.
40P	97%	16%	13%	0.97s.	6.22s.	0.35.
60P	93%	11%	12%	3.4s.	33s.	1.51s.

As can be seen in this table, the best-response algorithm, although not representing the best results with its simple implementation, can be also adapted to processor minimization problems. Throughout all of the experimentations carried out the obtained number of modules was in worst cases 2OPT, where OPT denotes the optimal number of processors.

4.5 Conclusion

In this chapter a best-response algorithm for the non-preemptive and strictly periodic multiprocessor scheduling problem has been proposed. This game theoretic approach appeared to be extremely efficient in solving the scheduling problem, especially when the exact method fails. The quality of the supplied solutions appeared promising and execution times were highly convenient given the complexities dealt with.

Numerical experiments have shown that the solution space is separated into a certain number of regions of attraction. Started in any point of such a region, the best-response algorithm will converge to the associated equilibrium. In order to ensure a better solution quality, multi-start methods with Bayesian stopping rules can be implemented. This enhanced greatly the quality of the algorithm, where the chances of finding an optimum increases. In addition, statistical information, such as an estimation on total volume of the regions of attraction, can be supplied.

CHAPTER 5

Virtual Link routing

After the allocation of resources to partitions, which was handled in previous chapters, routing tables for data and message transmission need to be completed, as was indicated in Section 1.5.2. For this purpose, the Virtual Link (VL) routing problem is addressed. In other words, approaches for routing the VLs through AFDX switches are investigated. These approaches should take into consideration the congestion in the AFDX network and the end-to-end transmission delays of the VLs.

Some characteristics of Virtual Links are first discussed in Section 5.1 to then indicate the problematic and related work in Section 5.2. A formal definition for the problem is indicated in Section 5.3. In Section5.4, an exact method is discussed. A two-level heuristic is then proposed in Section 5.5. Some experimentations are presented in Section 5.6 whereas Section 5.7 concludes this chapter.

5.1 Virtual Links

In AFDX networks, Virtual Links, which can be thought of as multicast trees [146], have the interest of routing isolated packets of data from a unique End System (interfacing a source Avionic Subsystem) to a predetermined set of other End Systems (destined Avionic Subsystems). In other words, the AFDX switches are configured to direct frames with the same Virtual Link ID through a predefined tunnel in the network. This data tunneling requires a reserved amount of bandwidth, which is defined by system integrators, on each link it traverses. Figure 5.1 demonstrates a Virtual Link originating at End System 1 and delivering data to destination End Systems 2, 3 and 4.

As multiple VLs share the physical links' bandwidth, preventing the traffic of one VL from interfering with traffic of other VLs on the same physical link is essential. This is done by limiting the rate at which Ethernet frames can be transmitted and their respective frame sizes on a virtual link. Each virtual link is thus assigned two parameters (*cf*. Figure 5.2):

- Bandwidth Allocation Gap (BAG): it defines the minimum time interval between the starting bits of two successive AFDX frames, assuming zero jitter. BAG values range from 1 ms to 128 ms

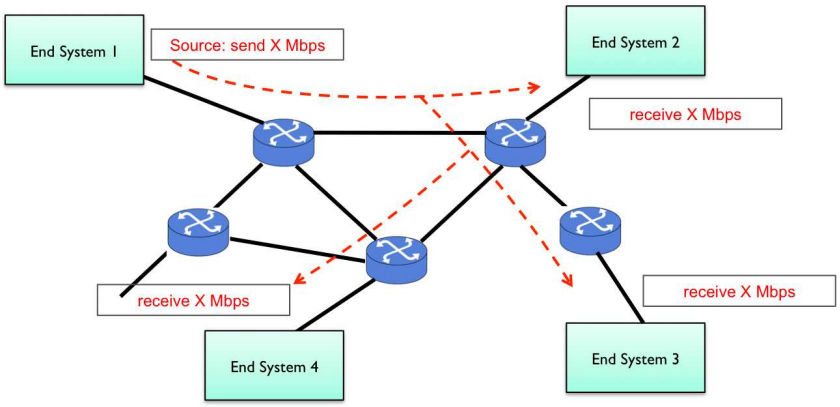


Figure 5.1: A virtual link originating at one End System and sending data to 3 others.

(values that are a power of 2). In other words, the BAG defines the maximum frequency at which frames are sent in a VL.

- Maximum Frame Size (MFS): it describes the maximum size, in bytes, of the transmitted Ethernet frames. MFS values can range from 64 bytes to a maximum of 1518 bytes.

The Virtual Link bandwidth can be hence calculated from the following equation,

$$bw = \frac{(67 + MFS)}{BAG}, \tag{5.1}$$

where 67 bytes are added to the MFS to represent the frame header.

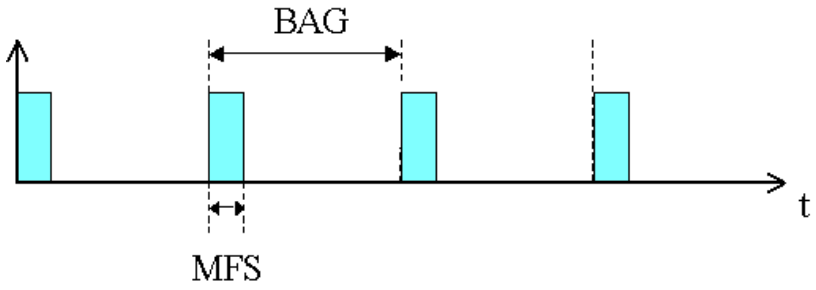


Figure 5.2: A VL can send frames with a maximum size of MFS bytes every BAG ms.

5.2 Problem description and related work

Virtual Link definition and routing are highly related to partition scheduling on the processing modules. Communication requirements are defined by system designers from specification phases. VLs, how-

ever, cannot be routed before scheduling the partitions and deducing implicated End Systems. Once the partition mapping is fixed, it becomes more obvious what VLs need to be defined and routed. System designers proceed by configuring routing tables in the AFDX switches. In other words, paths or multicast trees are determined in a manner respecting system resources and quality requirements, such as fair load balancing.

As was done in previous chapters, the automation of VL routing in the network is proposed. In this chapter, the partition allocation and scheduling decisions are considered to be already made, and hence the VLs and their characteristics are known.

The problem considered amounts to finding one and only one multicast tree for each VL. In the special case where only one VL exists, the problem becomes related to the computation of a minimum cost tree, known as a Steiner tree [69, 79]. This Steiner tree problem is NP-complete [136]. Another special case is obtained when each VL has a single destination, yielding a single-path routing problem. This problem is also known to be NP-complete [127] (*cf.* the same reference for node-link and link-path formulations of this problem).

Not much propositions can be found for routing several multicast demands at once under resource constraints. The problem was handled in a different context, that is minimizing a single multicast tree cost under certain constraints, if any. The authors in [136] look into finding multicast trees for multicast traffic requests that arrive one-by-one, while minimizing the maximum link utilization. Many others also propose multi-objective multicast routing algorithms for a single multicast traffic request in an already loaded network [44, 155].

In the avionic domain, VL routing can be thought of as an offline multicast tree allocation problem. The proposed methodology is closely related to that in [64] and [33], where the authors propose a 3-level optimization approach for single path routing. Their work formed an inspiration for the heuristic proposed in Section 5.5, with the main difference being the multicast aspect of the demands.

5.3 Formal definition of the problem

Let $\mathcal{N} = \{1, \ldots, N\}$ be a set of N nodes representing the AFDX network ($n \in \mathcal{N}$ may correspond to an AFDX switch or End System). Denote by $\mathcal{E}$ the set of directed edges (links) between nodes such that no edge exists between two End System nodes, that is to say, End Systems are interconnected through AFDX switches only. The set $\mathcal{C} = \{c_e \in \mathbb{R} : e \in \mathcal{E}\}$ represents link capacities.

Consider a set $\mathcal{V} = \{1, \ldots, V\}$ of V Virtual Links to be deployed in the AFDX network. Each VL $v \in \mathcal{V}$ is characterized by the following:

- a source node (End System) $src(v) \in \mathcal{N}$,
- a set of destination nodes (End Systems) $dest(v) \subseteq \mathcal{N} \setminus \{src(v)\}$, and the corresponding number of destinations $K_v = |dst(v)|$,
- and a bandwidth b_v, according to equation (5.1).

The VL routing problem amounts to finding one and only one tree for each VL.

In this thesis, the optimization associated to the Virtual Link routing problem is considered to be the minimization of the maximum link load in the network (links actually represent switch interfaces), that is,

$$\text{Minimize} \left[\rho = \max_{e \in \mathcal{E}} \left(\frac{y_e}{c_e} \right) \right],$$

where y_e represents the total load on edge (link) e. It should be noted that this choice was considered for the following reasons:

1. The network topology is fixed and hence equipment reduction, such as number of switches, can not be performed.
2. The load reduction was preferred, and not the end-to-end delay, as the delay arising in the network, and based on the analysis of a project partner [100], can be neglected against the partition periods and execution durations. Hence this delay is of insignificant effect in the inter-partition transmission delay discussed in Section 3.2.3.
3. This optimization was appealing for the project partners, among which, one in the avionic domain.

Additional or alternate optimization criteria is left for future work, though some delay related constraints can be exploited in the heuristic proposed in Section 5.5.

5.4 An exact node-link formulation

In this section an exact node-link formulation [2] based on Mixed Integer Linear Programming (MILP) is presented.

For every node $n \in \mathcal{N}$, let $\Gamma^+(n) \subseteq \mathcal{E}$ denote the set of incoming links to n and $\Gamma^-(n) \subseteq \mathcal{E}$ the set of outgoing links from n as can be seen in Figure 5.3.

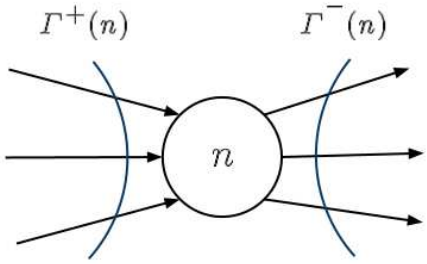


Figure 5.3: Links to and from node n.

The exact node-link formulation is as follows,

$$\begin{aligned}
&\min \rho \\
&s.t. \\
&\sum_{e\in\Gamma^+(n)} x_v^e - \sum_{e\in\Gamma^-(n)} x_v^e = h_{n,v} && ,\forall n, v, && (5.2)\\
&y_v^e\, M \geq x_v^e && ,\forall v, e, && (5.3)\\
&y_v^e \leq x_v^e && ,\forall v, e, && (5.4)\\
&\sum_{e\in\Gamma^+(n)} y_v^e \leq 1 && ,\forall n, v, && (5.5)\\
&y_e = \sum_{v\in\mathcal{V}} y_v^e\, b_v && ,\forall e, && (5.6)\\
&y_e \leq c_e\, \rho && ,\forall e, && (5.7)\\
&y_v^e \in \{0,1\} && ,\forall v, e, && (5.8)\\
&x_v^e \in \{0,\ldots,K_v\} && ,\forall v, e, && (5.9)
\end{aligned}$$

where x_v^e can be thought of as the number of destinations VL v is addressing via link e. The equality in constraint (5.2) indicates that the difference in the number of addressed destinations by VL v between before and after entering a node n should be equivalent to $h_{n,v}$, which is given by

$$h_{n,v} = \begin{cases} -K_v & \text{if } i = src(v), \\ 1 & \text{if } i \in dst(v), \\ 0 & \text{otherwise.} \end{cases}$$

Evidently the source node (End System) should address all of the destinations. An intermediate node (switch) should transfer whatever is addressed by the VLs from its input links to its output links. Finally a destination node (End System) should be the last in the chain as it cannot re-route information, and only receives what is addressed to it from VL v and hence the value of $h_{n,v} = 1$.

The constant value M in (5.3) is considered as a large number. Consequently, the boolean y_v^e indicates whether link e is traversed by VL v or not, i.e. if $x_v^e > 0$ then $y_v^e = 1$ indicating the passage of v in this link, otherwise it can be either 0 or 1 but will be set to 0 due to constraint (5.4). Constraint (5.5) ensures that each VL is routed along a tree where no node is visited more than once. Constraints (5.6) and (5.7) define the link loads and the maximum utilization rate of the links, respectively. Constraints (5.8) and (5.9) represent the domains for the decision variables y_v^e and x_v^e, respectively.

As a result of the preceding node-link formulation, trees for the various VLs can be assigned based on the decision variables y_v^e. The length of such trees is, however, not guaranteed and can hence take unnecessary routes in the network.

5.5 Two-level VL routing algorithm

Another well known formulation for single-path routing is the link-path formulation. Instead of investigating network nodes and demands passing through them, all path possibilities for joining sources

and destinations are considered. As a final solution, one path is assigned to each demand in a manner satisfying a certain criterion. A modified formulation for the VL tree allocation problem is as follows,

$$\min \ \rho \qquad \text{(OPT-VL)}$$

$$s.t.$$

$$\sum_{T \in \mathcal{T}^v} x_T = 1 \qquad ,\forall v \in \mathcal{V}, \qquad (5.10)$$

$$y_e = \sum_{v \in \mathcal{V}} \sum_{T \in \mathcal{T}^v} \delta_e^T \, x_T \, b_v \qquad ,\forall e \in \mathcal{E}, \qquad (5.11)$$

$$y_e \leq c_e \, \rho \qquad ,\forall e \in \mathcal{E}, \qquad (5.12)$$

$$x_T \in \{0,1\} \qquad ,\forall T \in \mathcal{T}^v, \forall v \in \mathcal{V}, \qquad (5.13)$$

where $\mathcal{T}^v$ represents the set of all possible trees for VL $v \in \mathcal{V}$. The variable x_T is a boolean decision variable on whether a tree $T \in \mathcal{T}^v$ is chosen for VL v. And the boolean variable δ_e^T indicates whether a given tree T traverses edge $e \in \mathcal{E}$, that is,

$$\delta_e^T = \begin{cases} 1 & \text{if } T \text{ traverses } e, \\ 0 & \text{else.} \end{cases}$$

Constraint (5.10) indicates that only one tree should be assigned for each VL. Equation (5.11) indicates the loads consumed on each link given the trees that pass through it. Constraint (5.12) is similar to Constraint (5.7) from the node-link formulation in Section 5.4.

Clearly, determining all tree possibilities for the VLs requires an exhaustive search. A turn-around may be to compute a predetermined set of candidate trees for each VL. This can be done based on a certain cost for trees, where amongst all possible trees, those of minimal cost are preferred. In what follows, a tree cost is equivalent to the sum of delays on the links it traverses. Delay on a link $e \in \mathcal{E}$, based on the M/M/1 queueing model [89], is,

$$D(y_e, c_e) = \frac{1}{c_e - y_e}. \qquad (5.14)$$

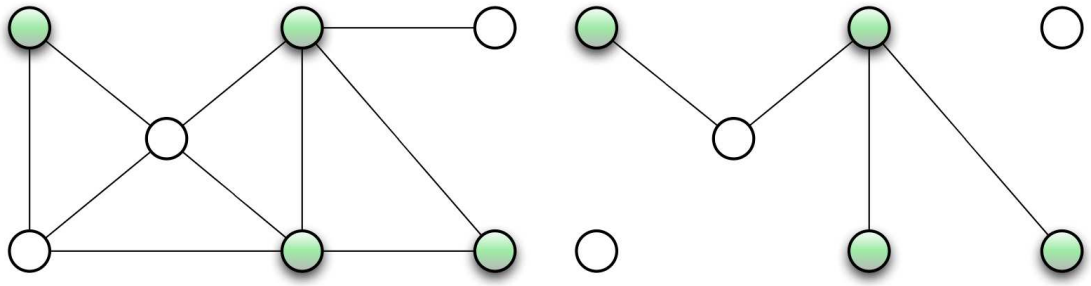
Although the hypothesis of the M/M/1 queueing model is typically violated in real networks, this delay function (5.14) represents a useful measure of performance in practice, principally because it expresses qualitatively that congestion sets in when a flow (traffic) y_e approaches the corresponding link capacity c_e. Nevertheless, it should be emphasized that the proposed optimization approach is not restricted to this delay function, yet it helps in selecting minimal length trees and balancing the switch loads.

The minimization of a tree cost is referred to as the Steiner tree problem.

5.5.1 Steiner tree problem

The Steiner problem amounts to finding a subtree spanning a set of vertices in a graph and having a minimal length, i.e. the sum of all link weights in this subtree is minimal. Steiner trees are superficially

similar to Minimum Spanning Trees (MST) with the difference being that, in contrary to an MST, a Steiner tree only interconnects a predefined set of nodes (called Steiner points) and not all. Figure 5.4b demonstrates a Steiner tree interconnecting a predefined set of Steiner points in a graph represented by Figure 5.4a.



(a) The complete graph. (b) Tree interconnecting Steiner points.

Figure 5.4: Steiner trees and Steiner points (green).

The interest in this chapter is in the Steiner problem for directed graphs [122]. Several proposals can be found for this subject such as [26].

In a directed graph with a set of nodes $\mathcal{N}$ and a set of edges $\mathcal{E}$ with edge costs $\mathcal{D} = (\Delta_e)$, let S be a set of Steiner points with s being the source Steiner point and $dst(s)$ the set of destination Steiner points. The Steiner problem can be formulated as follows:

$$\min \sum_{e \in \mathcal{E}} \delta_e \, \Delta_e \tag{5.15}$$

s.t. (5.16)

$$\sum_{e \in \Gamma^+(s)} \delta_e = 0, \tag{5.17}$$

$$\sum_{e \in \Gamma^-(s)} \delta_e > 0, \tag{5.18}$$

$$\sum_{e \in \Gamma^+(i)} \delta_e > 0, \quad \forall i \in dst(s), \tag{5.19}$$

$$\sum_{e \in \Gamma^-(j)} \delta_e = 0, \quad \forall j \in dst(s), \tag{5.20}$$

$$\sum_{e \in \Gamma^+(i)} \delta_e - \sum_{e \in \Gamma^-(i)} \delta_e = 0, \quad \forall i \in \mathcal{N} \setminus dst(s), \tag{5.21}$$

where δ_e is a boolean variable indicating if the Steiner tree traverses edge $e \in \mathcal{E}$.

As was indicated in Section 5.2, the Steiner tree problem is NP-complete. Approximation algorithms or heuristics are then preferred in this case. The algorithm used to solve the Steiner problem in this chapter is a modified version of the one discussed in [94], changes were made so that it would

be applied to directed graphs. It has a worst case time complexity of $O(|S|\,|\mathcal{N}|^2)$. The heuristic is summarized as follows (refer to Figure 5.5 for an example):

1. Starting with the initial graph $G = (\mathcal{N}, \mathcal{E})$ (*cf.* Figure 5.5a) representing the topology (modules + switches), construct the directed graph $G_1 = (\mathcal{N}_1, \mathcal{E}_1)$ (*cf.* Figure 5.5b) in which $\mathcal{N}_1 = S$ and $\mathcal{E}_1$ represents edges between Steiner points with the exception that the source Steiner point s has only outgoing edges, i.e. $\mathcal{E}_1 = \{e : e \in \Gamma^+(i), \forall i \in S \backslash \{s\}\}$. The associated edge costs represent shortest paths between the Steiner points in the graph G (the Dijkstra algorithm is used).

2. Find the directed minimal spanning tree T_1 of G_1 (*cf.* Figure 5.5c) of source s and as roots nodes in $dst(s)$. If several trees are found, choose an arbitrary one.

3. Replace every edge in T_1 with its corresponding shortest path from graph G. Graph G_S is formed.

4. Find once more the minimal spanning tree T_S of graph G_s (*cf.* Figure 5.5d). Again, if several trees are found, choose an arbitrary one.

5. The consequent Steiner tree T_H is formed by removing edges from T_S so that all leaves are Steiner points.

In this thesis, the cost Δ_e associated to link $e \in \mathcal{E}$, arising from routing additional demand b through this link, is taken to be the variation in the delay defined by (5.14), that is,

$$\Delta_e(b) = D(y_e + b, c_e) - D(y_e, c_e). \tag{5.22}$$

It is hereafter discussed in Sections 5.5.2 and 5.5.3 a two-level optimization approach: ILST (Iterative Loading of Steiner Trees) for computing tree candidate sets, and VLRO (Virtual Link Routing Optimization) which is basically (OPT-VL) applied on the predetermined candidate sets.

5.5.2 Iterative Loading of Steiner Trees (ILST)

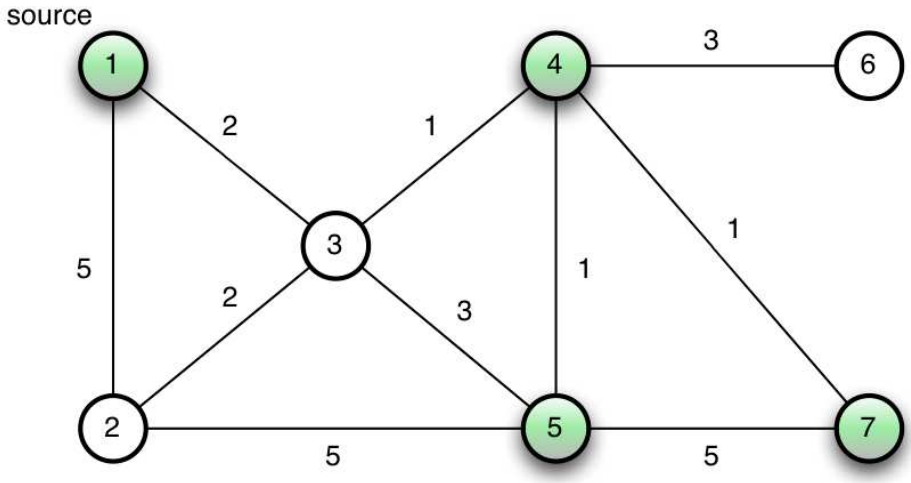
As was indicated earlier, each VL $v \in \mathcal{V}$ requires a tree to be configured for it to traverse the network. Possibilities for such a tree, and depending on the network topology, may be numerous. The efficiency of a VL routing algorithm may be reduced significantly by this great number. Candidate sets of trees have to be then computed for each VL before proceeding with load optimization, i.e. find candidate tree sets that lead to link load optimization without an exhaustive discovery.

Let $\mathcal{T}^v$ now be the set of tree candidates for VL $v \in \mathcal{V}$. Hence,

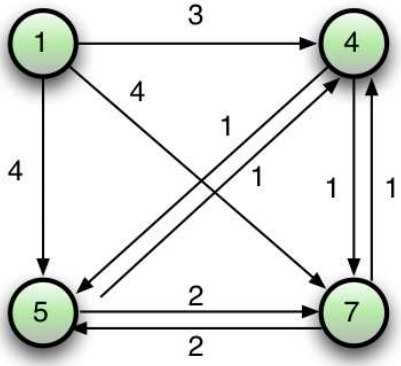
$$\mathcal{T}^v = \{T_1^v, \ldots, T_{p(v)}^v\},$$

where T_i^v is tree candidate i for VL v out of $p(v)$ candidates.

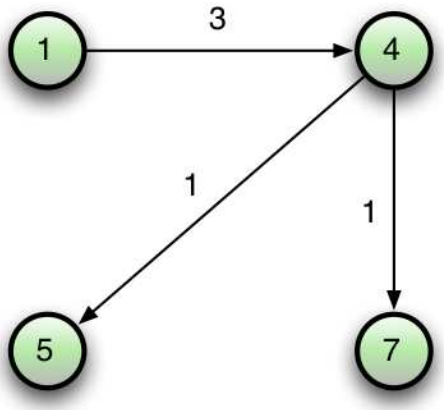
The general idea of the tree selection algorithm, is to progressively load the network with fractions of demands (VL bandwidth requirements) in R iterations (e.g. $R = 5$). At each iteration j, a candidate Steiner tree T_j^v is found for each VL $v \in \mathcal{V}$, given that $1/R$ of the demand (b_v/R) is to be routed. At the beginning of each iteration j, $\mathcal{V}_j = \mathcal{V}$. During iteration j, the following steps are repeated V times ($V = |\mathcal{V}|$):



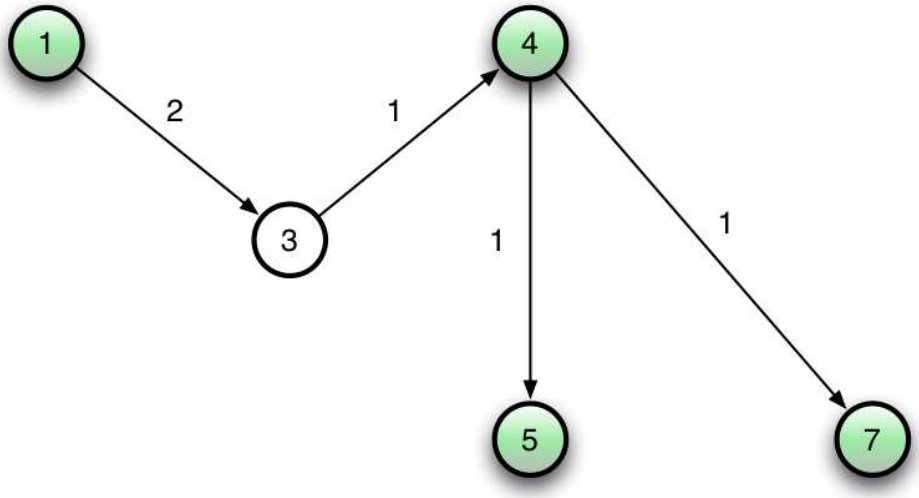
(a) Original graph G. Green hashed nodes are Steiner points, in which node 1 is source. All edges are bidirectional.



(b) Graph G_1 with Steiner points as nodes. Edge costs are equivalent to shortest paths between nodes from graph G.



(c) Minimal spanning tree T_1 of G_1.



(d) Edges in T_1 are replaced with shortest paths from G. No need to eliminate edges as final Steiner tree is obtained.

Figure 5.5: An example on obtaining a Steiner tree.

1. Select a VL $v \in \mathcal{V}_j$ according to a uniform distribution.
2. Compute the minimum Steiner tree T_j^v required to route b_v/R of v's bandwidth in the graph defined by the network nodes, and add it to the set $\mathcal{T}^v$.

   The associated cost for edge $e \in \mathcal{E}$ is defined by (5.22), that is $\Delta_e\left(\frac{b_v}{R}\right)$.
3. Do $\mathcal{T}^v = \mathcal{T}^v \cup \{T_j^v\}$ and $\mathcal{V}_j = \mathcal{V}_j - v$.
4. Update link loads y_e by propagating demand b_v/R along edges traversed by tree T_j^v.

This tree selection algorithm allows to greatly reduce the number of trees since it yields at most R Steiner trees per VL (the same tree can be obtained several times).

5.5.3 Virtual Link Routing Optimization (VLRO)

Now that a reduced set of candidate trees is computed for each VL, the optimization (OPT-VL) can be carried out to minimize the maximum link load in the network.

In this manner, tree possibilities that may be delay-wise costly are filtered out. It might be possible, however, that a solution characterized by the optimality in maximum link load is eliminated. Nevertheless, it will be shown in Section 5.6 that, under certain circumstances, the supplied solution has an objective similar to the one supplied by the node-link formulation. In other cases, and depending on the topology, the approach helps in reducing execution times while obtaining an acceptable result.

5.6 Results

In this section, some experimentations to evaluate the performance of the proposed methods are presented. The MILP formulations in Sections 5.4 and 5.5 are solved using CPLEX [81]. The machine used is similar to the one from Section 4.4 in Chapter 4.

As can be seen in Figure 5.6, the considered topology consists of 7 AFDX switches and 6 End Systems. The aim is to depict as much as possible the AFDX topology found on board aircrafts, though the final number of End Systems may be much larger. This simple example however allows a preliminary analysis of the quality of the algorithm proposed in Section 5.5.

All links are full duplex at 100Mbps. Given $nbVL$ virtual links, all End Systems are evenly attributed a number of VLs in which they act as a source (e.g. if $nbVL = 12$ then each End System in the considered topology acts as source for 2 VLs). For each VL, the destinations are chosen so that 1 to 5 End Systems (other than the source) are uniformly selected. Bandwidth requirements are chosen based on an exponential distribution with an average of 125Kbps (so that for large examples link capacities is not exceeded). In addition, the bandwidth for each VL is limited to the interval $[1/nb_{src}, 100/nb_{src}]$Mbps, where nb_{src} is the average number of VLs per source. Several examples are generated with $nbVL = \{100, 300, 500, 700, 1000\}$.

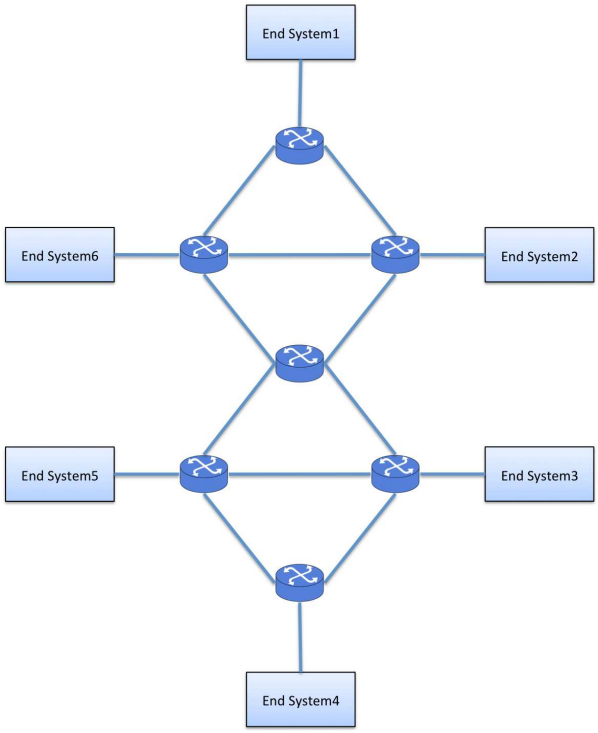


Figure 5.6: Topology considered for experimentations. Set of 7 AFDX switches and 6 End Systems.

For each case of $nbVL$, 100 instances were generated and solved with both, the exact node-link formulation (NL) and the two-level heuristic (2LH). Table 5.1 shows the execution times taken to solve the problems. In addition, the percentage of instances whose objective was computed to optimality via 2LH is also indicated.

As can be seen from Table 5.1, 2LH always gave an optimal solution which means that, for the considered topology at least, no optimal solutions were filtered out during the algorithm's initial phase (ILST). Execution times were however in favour for the exact NL. It should be noted that the heuristic may supply solutions that are not optimal for what concerns the maximum link utilization. This was not the case in the experimentations due to the small size and symmetry of the considered topology.

Table 5.2 presents the average delay arising in the network from the choices made (tree allocations). This delay D_{avg} is computed using Little's law and the M/M/1 queueing model,

$$D_{avg} = \frac{1}{\sum_v b_v} \sum_e \frac{y_e}{c_e - y_e}$$

It is obvious that the heuristic 2LH gave optimal solutions along with tree allocations yielding significantly lower delays in the network than those in NL. Figure 5.7 represents the average delay in the network for the instances with 1000 VLs. For instance 51, as an example, 2LH gave an optimal solution

Table 5.1: Computation times for NL and 2LH.

nbVL	CpuTime		Percentage optimal
	NL	2LH	instances (2LH)
100	1.52s.	2.45s.	100%
300	2.34s.	6.66s.	100%
500	3.85s.	11.01s.	100%
700	5.77s.	15.75s.	100%
1000	8.92s.	22.08s.	100%

with an average network delay that is about 8% better than that from NL (difference of about 14ms). Similar graphs can be drawn for the other instances (different $nbVLs$).

Table 5.2: Mean average delays arising in the network.

nbVL	Mean average delay [ms/Mb]	
	NL	2LH
100	94.82	90.64
300	100.23	94.89
500	110.91	104.15
700	128.03	119.99
1000	162.75	150.87

For further comparison, a benchmark with 4894 VLs was tested (corresponds to the benchmark of Section 4.4.1). The topology consisted of 72 End Systems, 7 switches and 176 links. The node-link formulation (NL) rapidly consumed the machine's memory resources and thus failed to solve the problem due the huge number of variables and constraints. The 2-level heuristic (2LH), however, managed to solve the problem within 34 minutes.

5.7 Conclusion

In this Chapter, the problem of VL routing in an AFDX network has been considered. The problem is NP-complete and has not been much treated. Two methods for the off-line allocation of multicast trees to VLs have been proposed. The first is based on a node-link formulation that is able to efficiently solve realistic problems to optimality, for what concerns minimizing maximum link utilization. This formulation, for relatively huge problems, may fail due to the resource depletion by the great number of variables and constraints generated. The other approach is based on a formulation similar to that of the link-path one in single-path routing problems, along with the pre-computation of a predetermined set

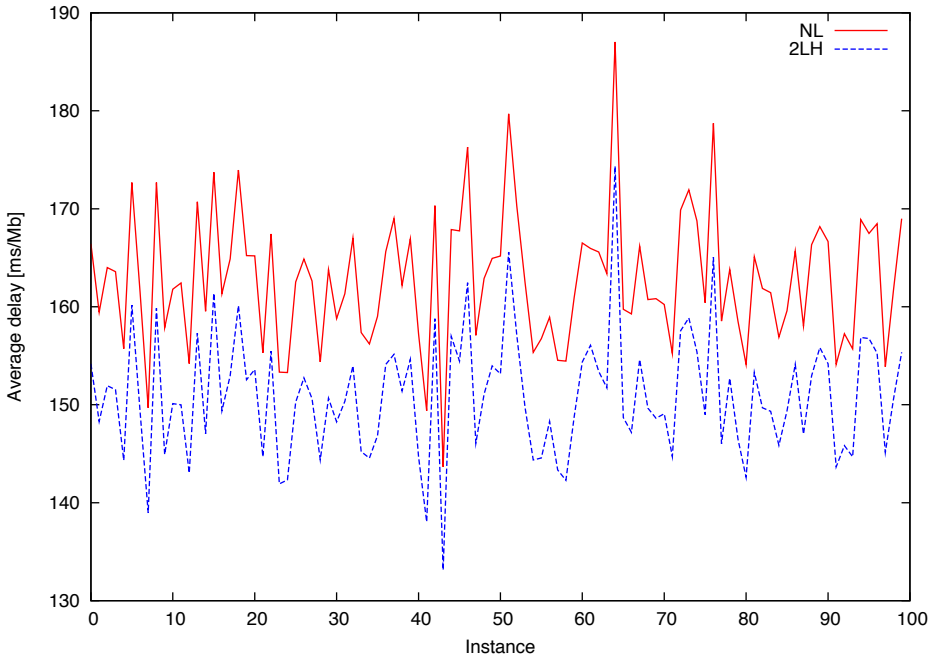


Figure 5.7: Average delays for instances with 1000 VLs.

of candidate trees. These candidate sets of trees help reduce the search space and hence ameliorate run times. This two-level heuristic, though may not always lead to an optimal minimization in maximum link utilization, allows integrating some criteria for choosing the candidate multicast trees. Criteria such as end-to-end delay may be implemented to filter out undesirable solutions.

Conclusion

The main motivation behind this thesis was the resource allocation in IMA-based avionic systems. Two key issues were identified and tackled: the scheduling of avionic applications (represented by partitions) on available resources and the routing of message flows (virtual links) in the avionic AFDX network. Indeed, these two matters represent system designers' major concerns in integration and configuration phases. This arises from the need to fill out configuration tables and parameters in a manner guaranteeing the proper functioning of the system, which may become cumbersome given the multitude of requirements and considerations. Additionally, the introduction of new components, be it hardware or software, may necessitate a re-verification of the entire configuration. Therefore, the work presented in previous chapters amount to automating such decision making processes.

The scheduling problem was first addressed, for harmonic and non-harmonic periods alike. A first approach was based on an exact Mixed Integer Linear Programming formulation which incorporates the system's various constraints. These constraints are either related to resource capacities such as the processing modules' memory, or to safety requirements such as prohibiting an application from executing on some of the modules. The decision problem, regarding the existence of a feasible schedule and allocation of available resources to the partitions, was transformed into an optimization one to provide partitions with extra temporal execution potentials. The optimal solution was characterized by a coefficient by which all partition execution times can be multiplied without rendering the schedule infeasible. A value greater than 1 for this coefficient not only implies the feasibility of the scheduling problem, but also gives an idea on the evolution margin for partition temporal executions, whenever required. The exact formulation, though ensures optimality, was shown to be inefficient for large instances. It even fails to converge in a sufficient amount of time for fairly limited problems, such as those with 4 modules and 40 partitions. Yet it may be used to assess the optimality of proposed algorithms on fairly limited instances.

After this exact resolution, an efficient heuristic was introduced. This heuristic, called best-response algorithm, was inspired from Game Theory and was made tailored to the specific scheduling problem. In this algorithm, partitions take turns in changing strategies (modules and offsets) to increase evolution potentials in which they are implicated. This successfully yielded convenient results in interesting

amounts of time. In comparison to the exact method, the algorithm was characterized by small relative errors on the optimization criterion considered in this thesis and almost always provided feasible solutions. It even exceeded the limitations for large scale instances, with tens of modules and hundreds of partitions, where the exact method failed to even supply any solution. As an example, a benchmark supplied by a project partner was solved in under 3 minutes and, though can not be compared to the optimal solution, the algorithm yielded a feasible solution respecting all imposed constraints. Another advantage of this heuristic is the additional evolution potential supplied to the partitions. The exact method converges after the minimum evolution potential in the system, which is usually related to one or several partitions, is maximized. This exact resolution disregards any additional potential for the other partitions. However, the best-response algorithm converges when partition strategies remain unchanged, that is the algorithm continues beyond the aforementioned objective, assuming the optimum is found, until no more evolution potential can be fairly supplied to the partitions.

The quality of the best-response algorithm was also shown to be ameliorated if a suitable starting point/solution is considered. For this, the application of multi-start methods with Bayesian stopping rules has greatly reduced the relative errors on the solutions. It was noticed that several equilibria, of various volumes for their regions of attraction, may exist. Choosing a starting point in a bad region of attraction may lead to a not so convenient solution. The statistical estimations associated with the Bayesian stopping rules allows quantifying the confidence on the optimality of obtained solutions. A 98% estimated total relative volume for the regions of attraction, for example, indicates a high probability of having an optimal solution discovered. This however does not guarantee finding an optimal solution all of the time. The obtained results were nevertheless satisfactory in most of the cases.

The last part of the thesis handles the routing problem in the AFDX network. This network is based on routing Ethernet frames through isolated data tunnels referred to as virtual links. The virtual links, which are represented by multicast routing trees, are deployed for exchanging data between the avionic applications. Hence, their routing is carried out after allocating the partitions on the processing modules. Again, the decision here has to follow a certain criterion which, given that in this thesis the network topology can not be modified and the network delay will be relatively negligible as compared to total the end-to-end transmission delay (partition execution durations and periods), was considered the minimization of the maximum link load.

For this routing problem, two methods were proposed. The first is an exact node-link formulation, and the second is a two level heuristic, which filters as a first step multicast trees that exceed a certain length (delay-wise), and then solves a path-link formulation to solve the proposed optimization. To demonstrate the performance of these algorithms, a topology similar to an avionic one is considered, where the number of switches is limited. Several number of virtual links to be routed were considered. The two algorithms successfully solved the problem to optimality in adequate run times. Though not necessarily optimal, the heuristic can sometimes give sub-optimal solutions in case optimal routes are filtered out. However, one advantage is that it might provide better average delays in the network. Another advantage is that additional or alternate filtering rules can be implemented in the heuristic's first phase. For an industrial example, and as part of the benchmark considered for the partition scheduling problem, an avionic topology with almost 5000 virtual links was considered. The exact formulation failed in this example to give a solution as the colossal number of variables and constraints exceeded

the offered system memory of the machine used for the runs. The two-level heuristic, on the other hand, solved the problem within half an hour. This last result sheds light on a limitation for the exact node-link formulation.

Perspectives

The proposed best-response algorithm, though producing interesting results, can be further enhanced beyond the scope of this thesis. Although it was proven that this algorithm converges and an optimal solution can be obtained, future work should include the refined analysis of the convergence time as well as the analysis of the error made by this heuristic. For the latter point, the goal is to derive an upper bound on the so-called Price of Anarchy [95], i.e. the ratio between the global cost obtained by the worst Nash equilibrium and a global optimal solution. This is needed to evaluate the quality of the algorithm for large instances, where the exact method fails in supplying a solution. Also, started in any point, the best-response algorithm will converge to an associated equilibrium. Although the application of multi-start methods with Bayesian rules enables discovering more of the regions of attraction, it starts from the assumption that the equilibria are equiprobable. This can be viable if no information is available on the regions of attraction, which is the case here. However, it seems interesting to analyse the number of the regions of attraction as well as their measures in order to devise a more precise prior probability density function for the problem, and hence enhancing the performance of the stopping rules.

Getting down to specifics, the scheduling problem discussed in this thesis disregards reconfiguration considerations. For such purposes, alternate schedules should be provisioned to guarantee the robustness of the systems, e.g. a module failure may necessitate the reallocation of partitions. In addition, the feedback discussed in Section 1.4, between the thesis' resource allocation and the latency checker, can be further reinforced. Methods, such as Benders' decomposition [27], can be implemented to divide the global resource allocation problem and gradually modify delay related parameters, such as the inter-module delay introduced in Section 3.1.1, and which was considered predefined.

For what concerns the proposed virtual link routing algorithms, future work should involve further amelioration of the algorithm used for computing candidate trees in the heuristic's first phase. This includes investigating other approaches for the Steiner tree problem which may be more efficient. Optimization wise, it may be of interest to consider multi-objective problems where a compromisation between link loads and end-to-end delays is searched. Furthermore, although the hypothesis on the M/M/1 queueing model provides a useful measure of performance, it may be more efficient to implement the deterministic aspects of the AFDX network.

Another research direction worth pursuing concerns the construction of the virtual links themselves. This constitutes an intermediary decision making phase between the scheduling and routing ones. A set of messages are to be deployed into one or separate virtual links. This includes the definition of key attributes such as the BAG and the MFS. These attributes should lead to the transmission of rate fixed frames in a manner respecting the delay limits between the production and consumption of messages. This can be hence achieved while minimizing the bandwidth reserved for the virtual links, for example,

without violating any system imposed constraints.

In a more general setting, the best-response algorithm appears promising for minimization or maximization problems. Hence, its behaviour under the various circumstances should be analysed. The addition of precedence between executing tasks is an example of additional constraints that are not considered in this thesis, yet are substantial under different contexts.

APPENDIX A

Maximal independent set and maximum clique problems

In this appendix, the Maximum Independent Set (MIS) and Maximum Clique (MC) problems are briefly introduced. An algorithm for computing an MC, and from which an MIS can be deduced, is presented.

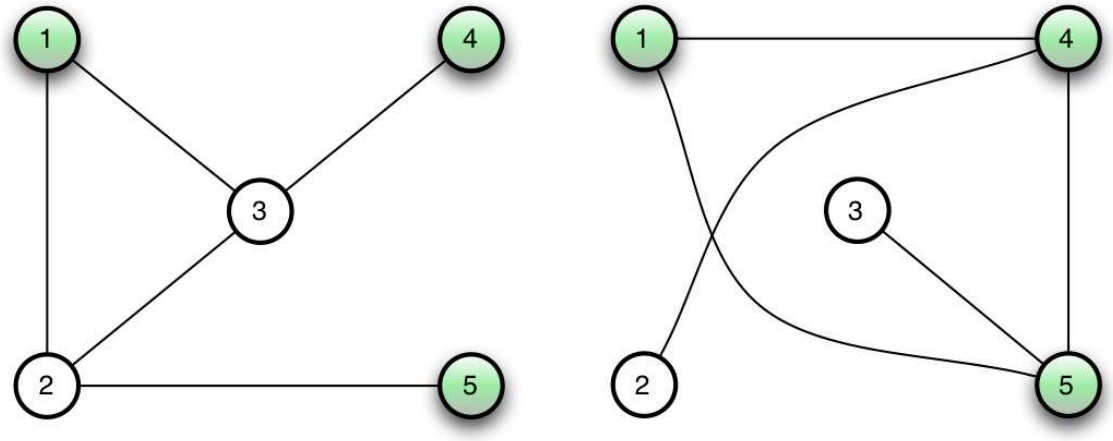
Given a graph $G = (N, A)$, with N and A being the sets of nodes and arcs respectively. An independent set is a set of nodes $S \subseteq N$ such that if $u, v \in S$, then $(u, v) \notin A$. A maximal independent set is an independent set to which no more nodes can be added without violating the independence property. As an example, Figure A.1a represents a graph G formed of 4 nodes in which green nodes form a maximum independent set.

The complementary graph $G' = (N, A')$ of G (such that $\forall u, v \in N$, $(u, v) \in A'$ iff $(u, v) \notin A$) allows the transformation of the maximal independent set problem into a maximum clique one.

For the graph G', a clique is a subset C of nodes such that each pair of nodes in C is connected by an edge. The MC problem is one of the first shown to be NP-hard. The MIS and the MC problems are essentially equivalent and have been extensively studied in graph theory and combinatorial optimization [138]. Figure A.1b demonstrates the complementary graph G' of G where green nodes represent a maximum clique, and which can be considered as an MIS in G.

Consequently, finding a maximum clique in G' gives nodes that form a maximal independent set in G. Algorithm 8 represents an old algorithm introduced by Carraghan and Pardalos [36] for solving the maximum clique problem. The set of nodes adjacent to node v is denoted by $N(v)$. The variable max, which is global, gives the size of a maximum clique when the algorithm terminates.

It should be noted that newer and faster algorithms exist such as [124], but for sake of simplicity the preceding one was presented.



(a) Graph G with an MIS.

(b) Graph G' with an MC.

Figure A.1: A graph G and its complementary G'.

Algorithm 8 Maximum clique algorithm

```
1: max := 0;
2: clique(N,0);
3:
4: procedure clique(U,size);
5: if |U| = 0 then
6:    if size > max then
7:       max := size;
8:       new record, save it;
9:    end if
10:   return;
11: end if
12: while |U| ≠ ∅ do
13:   if size + |U| ≤ max then
14:      return;
15:   end if
16:   i := min{j|v_j ∈ U};
17:   U := U − {v_i};
18:   clique(U ∩ N(v_j),size + 1);
19: end while
20: return;
```

APPENDIX B

List of publications

Related to the thesis:

- **Strictly periodic scheduling on an IMA-based avionic platform.**
  A. AL SHEIKH, O. BRUN, P.E. HLADIK, B. PRABHU. In 15th Austrian-French-German conference on Optimization (AFG'11), Toulouse, France, September 2011.

- **A best-response algorithm for multiprocessor periodic scheduling.**
  A. AL SHEIKH, O. BRUN, P.E. HLADIK, B. PRABHU. In proceedings of the 23rd Euromicro Conference on Real-Time Systems (ECRTS 2011), Porto, Portugal, July 2011.

- **Ordonnancement de tâches sous contrainte de périodicité stricte.**
  A. AL SHEIKH, O. BRUN, P.E. HLADIK. In Congrès Annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2011), Vol. I, Saint-Étienne, France, March 2011.

- **Partition scheduling on an IMA platform with strict periodicity and communication delays.**
  A. AL SHEIKH, O. BRUN, P.E. HLADIK. In proceedings of the 18th International Conference on Real-Time and Network Systems (RTNS 2010), Toulouse, France, November 2010.

- **Decision support for task mapping on IMA architecture.**
  A. AL SHEIKH, O. BRUN, P.E. HLADIK. In proceedings of the 3rd Junior Researcher Workshop on Real-Time Computing (JRWRTC 2009), Paris, France, October 2009.

Other:

- **Flow-level modelling of TCP traffic using GPS queueing networks.**
  O. BRUN, A. AL SHEIKH, J.M. GARCIA. In proceedings of the 21st International Teletraffic Congress (ITC'21), Paris, France, September 2009.

BIBLIOGRAPHY

[1] I. Ahmad, S. Ranka, and S. Khan. Using game theory for scheduling tasks on multi-core processors for simultaneous optimization of performance and energy. In *IEEE International Symposium on Parallel and Distributed Processing, 2008 (IPDPS 2008)*, pages 1–6, 2008.

[2] R. Ahuja, T. Magnanti, J. Orlin, and K. Weihe. *Network flows: theory, algorithms, and applications*. Prentice hall Englewood Cliffs, NJ, 1993.

[3] Airlines electronic engineering committee (AEEC). Design Guidance for Integrated Modular Avionics. *ARINC specification 651*, 1991.

[4] Airlines electronic engineering committee (AEEC). Backplane Data Bus. *ARINC specification 659*, 1993.

[5] Airlines electronic engineering committee (AEEC). Mark 33 Digital Information Transfer System (DITS). *ARINC specification 429, (parts 1, 2, 3)*, 2001.

[6] Airlines electronic engineering committee (AEEC). Aircraft data network, Part 1: Systems concepts and overview. *ARINC specification 664*, 2002.

[7] Airlines electronic engineering committee (AEEC). Aircraft data network, Part 7: Avionics Full Duplex Switched Ethernet (AFDX) Network. *ARINC specification 664*, 2005.

[8] Airlines electronic engineering committee (AEEC). Avionics application software standard interface. *ARINC specification 653, (part 1)*, 2006.

[9] A. Al-Sheikh, O. Brun, and P.-E. Hladik. Decision support for task mapping on IMA architecture. In *Proceedings of the 3rd Junior Researcher Workshop on Real-Time Computing (JRWRTC2009)*, pages 31–34, 2009.

[10] A. Al-Sheikh, O. Brun, and P.-E. Hladik. Partition scheduling on an IMA platform with strict periodicity and communication delays. In *Proceedings of the 18th International Conference on Real-Time and Network Systems (RTNS 2010)*, pages 179–188, 2010.

[11] A. Al-Sheikh, O. Brun, and P.-E. Hladik. Ordonnacement de tâches sous contrainte de périodicité stricte. In *Congrès Annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2011)*, volume I, pages 55–56, 2011.

[12] A. Al-Sheikh, O. Brun, P.-E. Hladik, and B. J. Prabhu. A best-response algorithm for multiprocessor periodic scheduling. In *Proceedings of the 23rd Euromicro Conference on Real-Time Systems (ECRTS 2011)*, 2011.

[13] A. Al-Sheikh, O. Brun, P.-E. Hladik, and B. J. Prabhu. Strictly periodic scheduling on an IMA-based avionic platform. In *15th Austrian-French-German conference on Optimization (AFG'11)*, 2011.

[14] R. Alena, J. Ossenfort, K. Laws, A. Goforth, and F. Figueroa. Communications for integrated modular avionics. In *2007 IEEE Aerospace Conference*, pages 1–18, 2007.

[15] B. Andersson and J. Jonsson. Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition. In *RTCSA*, page 337. Published by the IEEE Computer Society, 2000.

[16] F. Authority. 178B, Software considerations in airborne systems and equipment certification. *DO-178B/ED-12B, Radio Technical Commission for Aeronautics*, 1992.

[17] A. Avižienis, J. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, pages 11–33, 2004.

[18] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *International Parallel and Distributed Processing Symposium*, page 9, 2003.

[19] K. Baker. *Introduction to sequencing and scheduling*. Wiley, New York, 1974.

[20] T. Baker and A. Shaw. The cyclic executive model and Ada. *Real-Time Systems*, 1(1):7–25, 1989.

[21] P. Baptiste and V. Timkovsky. Shortest path to nonpreemptive schedules of unit-time jobs on two identical parallel machines with minimum total completion time. *Mathematical Methods of Operations Research*, 60(1):145–153, 2004.

[22] C. Barnhart, E. Johnson, G. Nemhauser, M. Savelsbergh, and P. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[23] S. Baruah. The non-preemptive scheduling of periodic tasks upon multiprocessors. *Real-Time Systems*, 32(1):9–20, 2006.

[24] S. Baruah and S. Chakraborty. Schedulability analysis of non-preemptive recurring real-time tasks. In *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, page 149, 2006.

[25] S. Baruah, N. Cohen, C. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.

[26] J. Beasley. An algorithm for the Steiner problem in graphs. *Networks*, 14(1):147–159, 1984.

[27] J. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.

[28] G. Bernat, A. Colin, and S. Petters. WCET analysis of probabilistic hard real-time systems. In *23rd IEEE Real-Time Systems Symposium (RTSS 2002)*, pages 279–288, 2002.

[29] P. Bieber, J. Bodeveix, C. Castel, D. Doose, M. Filali, F. Minot, and C. Pralet. Constraint-based Design of Avionics Platform–Preliminary Design Exploration. *4th European Congress on Embedded Real Time Software (ERTS 2008)*, 2008.

[30] C. Boender and A. Rinnooy Kan. Bayesian stopping rules for multistart global optimization methods. *Mathematical Programming*, 37(1):59–80, 1987.

[31] R. Boute. The Euclidean definition of the functions div and mod. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 14(2):144, 1992.

[32] P. Brucker and S. Kravchenko. *Complexity of mean flow time scheduling problems with release dates*. PhD thesis, Universität Osnabrück, Fachbereich Mathematik/Informatik, 2004.

[33] O. Brun and J. Garcia. Ressource allocation in communication networks. In *Proceedings of the 5th IEEE International Conference on High-Speed Networks and Multimedia Communications (HSNMC'02)*, pages 229–233, 2002.

[34] A. M. Campbell and J. R. Hardin. Vehicle minimization for periodic deliveries. *European Journal of Operational Research*, 165(3):668 – 684, 2005.

[35] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. In *Handbook on Scheduling Algorithms, Methods, and Models*, 2004.

[36] R. Carraghan and P. Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9(6):375–382, 1990.

[37] P. Chanet and V. Cassigneul. How to control the increase in the complexity of civil aircraft on-board systems. *AGARD, Aerospace Software Engineering for Advanced Systems Architectures*, 1993.

[38] W. Chang-Jun and X. Yu-Geng. Modeling and analysis of single machine scheduling based on noncooperative game theory. *Acta Automatica Sinica*, 2005.

[39] E. Clarke. Model checking. In *Foundations of Software Technology and Theoretical Computer Science*, pages 54–56. Springer, 1997.

[40] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[41] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri. Ordonnancement temps réel: cours et exercices corrigés. *Hermès science publications*, 2000.

[42] F. Cottet and Z. Mammeri. *Scheduling in real-time systems*. Wiley, 2002.

[43] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed systems: concepts and design*. Addison-Wesley Longman, 2005.

[44] J. Crichigno and B. Barán. A multicast routing algorithm using multiobjective optimization. *Telecommunications and Networking-ICT 2004*, pages 63–74, 2004.

[45] F. Cristian. Understanding fault-tolerant distributed systems. *Communication of the ACM*, 34(2):56–78, 1993.

[46] Cucu, L. *Ordonnancement non préemptif et condition d'ordonnançabilité pour systèmes embarqués à contraintes temps réel*. PhD thesis, Universit de Paris Sud, Spcialit lectronique, 2004.

[47] K. Danne and M. Platzner. An EDF schedulability test for periodic tasks on reconfigurable hardware devices. In *Proceedings of the 2006 ACM SIGPLAN/SIGBED conference on Language, compilers, and tool support for embedded systems*, page 102, 2006.

[48] G. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960.

[49] T. Davidovic, L. Liberti, N. Maculan, and N. Mladenovic. Mathematical programming-based approach to scheduling of communicating tasks. Technical Report G-2004-99, Cahiers du GERAD, 2004.

[50] J. Decotignie. Ethernet-based real-time and industrial communications. *Proceedings of the IEEE (Special issue on industrial communication systems)*, 93(6):1102–1117, 2005.

[51] D. Decotigny. *Une infrastructure de simulation modulaire pour l'évaluation de performances de systèmes temps-réel*. PhD thesis, Université de Rennes, 2003.

[52] M. DeGroot. *Optimal statistical decisions*. Wiley, 2004.

[53] N. Deshpande and S. K. *Distributed Systems*. Technical Publications, 2009.

[54] C. Dürr and K. Nguyen. Non-clairvoyant scheduling games. *Algorithmic Game Theory*, pages 135–146, 2009.

[55] F. Eisenbrand, N. Hähnle, M. Niemeier, M. Skutella, J. Verschae, and A. Wiese. Scheduling periodic tasks in a hard real-time environment. *Automata, Languages and Programming*, pages 299–311, 2010.

[56] F. Eisenbrand, K. Kesavan, R. Mattikalli, M. Niemeier, A. Nordsieck, M. Skutella, J. Verschae, and A. Wiese. Solving an avionics real-time scheduling problem by advanced IP-methods. *Algorithms–ESA 2010*, pages 11–22, 2010.

[57] J.-P. Elloy. Editorial: Quelle informatique est donc nécessaire pour automatiser en temps réel. *Technique et Sciences Informatique*, 7(5):395–396, 1988.

[58] M. Felser. Real-time ethernet-industry prospective. *Proceedings of the IEEE*, 93(6):1118–1129, 2005.

[59] T. Ferguson. Linear programming: A concise introduction. *Website. Available at http://www.math.ucla.edu/˜tom/LP.pdf*, 2011.

[60] N. Fisher and S. Baruah. The feasibility of general task systems with precedence constraints on multiprocessor platforms. *Real-Time Systems*, 41(1):1–26, 2009.

[61] B. Forouzan and S. Fegan. *Local area networks*. McGraw-Hill, 2003.

[62] C. Fraboul and F. Martin. Modeling advanced modular avionics architectures for early real-time performance analysis. In *Euromicro Conference on Parallel, Distributed, and Network-Based Processing*, pages 181–188, 1999.

[63] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, 1991.

[64] J. Garcia, A. Rachdi, and O. Brun. Optimal LSP Placement with QoS Constraints in Diff-Serv/MPLS Networks. *Providing Quality of Service in Heterogeneous Environments*, page 11, 2003.

[65] M. Garey and D. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4:397, 1975.

[66] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman & Co. New York, 1979.

[67] F. Gärtner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys*, 31(1):1–26, 1999.

[68] L. George, P. Muhlethaler, and N. Rivierre. Optimality and non-preemptive real-time scheduling revisited. Technical report, Rapport de Recherche RR-2516, INRIA, 2006.

[69] E. Gilbert and H. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968.

[70] R. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958.

[71] J. Goossens. *Scheduling of hard real-time periodic systems with various kinds of deadline and offset constraints*. PhD thesis, Université Libre De Bruxelles, Faculté des Sciences, 1999.

[72] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Systems*, 25(2):187–205, 2003.

[73] J. Goossens and C. Macq. Limitation of the hyper-period in real-time periodic task set generation. In *Proceedings of the RTS Embedded System (RTS'01)*, pages 133–148, 2001.

[74] N. Guan, Z. Gu, Q. Deng, S. Gao, and G. Yu. Exact schedulability analysis for static-priority global multiprocessor scheduling using model-checking. In *Proceedings of the 5th IFIP WG 10.2 international conference on Software technologies for embedded and ubiquitous systems*, pages 263–272, 2007.

[75] M. Harbour, J. García, J. Gutiérrez, and J. Moyano. Mast: Modeling and analysis suite for real time applications. In *13th Euromicro Conference on Real-Time Systems*, pages 125–134, 2001.

[76] K. Hoffman and M. Padberg. LP-based combinatorial problem solving. *Annals of Operations Research*, 4(1):145–194, 1985.

[77] P. Holman. *On the implementation of Pfair-scheduled multiprocessor systems*. PhD thesis, University of North Carolina, Chapel Hill, the Department of Computer Science, 2004.

[78] J. Huysseune and P. Palmer. NEVADA-PAMELA-VICTORIA: Towards the definition of new aircraft electronic systems. *Air & Space Europe*, 3(3-4):180–183, 2001.

[79] F. Hwang and D. Richards. Steiner tree problems. *Networks*, 22(1):55–89, 1992.

[80] J. Hyman, A. Lazar, and G. Pacifici. Real-time scheduling with quality of service constraints. *IEEE Journal on Selected Areas in Communications*, 9(7):1052–1063, 1991.

[81] ILOG CPLEX. `http://www.ilog.com/products/cplex/`, 2011.

[82] ISO, 1993. Road Vehicles–Interchange of Digital Information–Controller Area Network (CAN) for High Speed Communication. *ISO 11898:1993*, 1993.

[83] P. Jalote. *Fault-Tolerance in Distributed Systems*. Prentice-Hall, 1994.

[84] G. Jones and J. Jones. *Elementary number theory*. Springer Verlag, 1998.

[85] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390, 1986.

[86] L. Kantorovich. Mathematical methods of organizing and planning production. *Management Science*, 6(4):366–422, 1960.

[87] O. Kermia. *Ordonnancement temps réel multiprocesseur de tâches non-préemptives avec contraintes de précédence, de périodicité stricte et de latence*. PhD thesis, Université Paris XI, UFR scientifique d'Orsay, 2009.

[88] O. Kermia and Y. Sorel. A rapid heuristic for scheduling non-preemptive dependent periodic tasks onto multiprocessor. In *Proceedings of ISCA 20th international conference on Parallel and Distributed Computing Systems, PDCS*, volume 7, 2007.

[89] L. Kleinrock. *Queueing systems, volume I: theory*. Wiley Interscience, 1975.

[90] Kocik, R. *Sur l'optimisation des systèmes distribués temps réel embarqués: application au prototypage rapide d'un véhicule électrique semi-autonome*. PhD thesis, Université de Rouen, Spécialité informatique industrielle, 2000.

[91] J. Korst. *Periodic multiprocessor scheduling*. PhD thesis, Eindhoven university of technology, Eindhoven, the Netherlands, 1992.

[92] J. Korst, E. Aarts, Lenstra, and J. Karel. Scheduling periodic tasks. *Informs Journal on Computing*, 8:428–435, 1996.

[93] J. Korst, E. Aarts, and J. K. Lenstra. Scheduling periodic tasks with slack. *Informs Journal on Computing*, 9:351–362, 1997.

[94] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta informatica*, 15(2):141–145, 1981.

[95] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th annual conference on Theoretical Aspects of Computer Science*, pages 404–413, 1999.

[96] J. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical society*, pages 48–50, 1956.

[97] Y. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)*, 31(4):406–471, 1999.

[98] A. Land and A. Doig. An automatic method for solving discrete programming problems. *50 Years of Integer Programming 1958-2008*, pages 105–132, 2010.

[99] J.-C. Laprie. Sûreté de fonctionnement informatique: concepts de base et terminologie. Technical report, LAAS-CNRS, Toulouse, France, 2004.

[100] M. Lauer, J. Ermont, F. Boniol, and C. Pagetti. Latency and freshness analysis on IMA systems. In *Proceedings of the 16th IEEE international conference on Emerging technologies & factory automation (ETFA 2011)*, 2011.

[101] M. Lauer, J. Ermont, C. Pagetti, and F. Boniol. Analyzing end-to-end functional delays on an IMA platform. In *Proceedings of the 4th international conference on Leveraging applications of formal methods, verification, and validation - Volume Part I*, ISoLA'10, pages 243–257, 2010.

[102] E. Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Algorithmic aspects of combinatorics*, 2:75–90, 1978.

[103] E. Lee. MIL-STD-1553 Tutorial. Technical report, Technical Report 3.41, CONDOR Engineering, 2000.

[104] J. Leung and M. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information processing letters*, 11(3):115–118, 1980.

[105] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982.

[106] L. Liberti. Automatic reformulation of bilinear MINLPs. Technical Report 2004.24, DEI, Politecnico di Milano, July 2004.

[107] L. Liberti and M. Drazic. Variable neighbourhood search for the global optimization of constrained NLPs. In *Proceedings of Global Optimization*, pages 1–5, 2005.

[108] C. Lin and C. Liao. Makespan minimization for multiple uniform machines. *Computers & Industrial Engineering*, 54(4):983–992, 2008.

[109] C. Liu. Scheduling algorithms for multiprocessors in a hard real-time environment. *JPL Space Programs Summary 37-60*, 2:28–37, 1969.

[110] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[111] J. Lopez, J. Diaz, and D. Garcia. Minimum and maximum utilization bounds for multiprocessor RM scheduling. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, page 67, 2001.

[112] J. Lopez, M. Garcia, J. Diaz, and D. Garcia. Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. *Euromicro-RTS*, page 25, 2000.

[113] T. Magnanti and R. Vachani. A strong cutting plane algorithm for production scheduling with changeover costs. *Operations Research*, 38(3):456–473, 1990.

[114] M. Marouf and Y. Sorel. Schedulability conditions for non-preemptive hard real-time tasks with strict period. In *Proceedings of the 18th International Conference on Real-Time and Network Systems (RTNS 2010), pp.50-58*, 2010.

[115] R. Martí. Multi-start methods. *Handbook of metaheuristics*, pages 355–368, 2003.

[116] P. Meumeu and Y. Sorel. Non-schedulability conditions for off-line scheduling of real-time systems subject to precedence and strict periodicity constraints. In *Proceedings of 11th IEEE International Conference on Emerging technologies and Factory Automation (ETFA06)*, 2006.

[117] A. Mihaela, D. Teselios, R. Prundeanu, and I. Popa. Duality in linear programming. *MPRA Paper*, 2010.

[118] A. Mok and M. Dertouzos. Multiprocessor scheduling in a hard real-time environment. In *Proceedings of the Seventh Texas Conference on Computing Systems*, 1978.

[119] Mok, A.K.L. *Fundamental design problems of distributed systems for the hard-real-time environment*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1983.

[120] R. Myerson. *Game theory: analysis of conflict*. Harvard University Press, 1997.

[121] J. Nash. Non-cooperative games. *Annals of mathematics*, pages 286–295, 1951.

[122] L. Nastansky, S. Selkow, and N. Stewart. Cost-minimal trees in directed acyclic graphs. *Mathematical Methods of Operations Research*, 18(1):59–67, 1974.

[123] M. Osborne and A. Rubinstein. *A course in game theory*. The MIT press, 1994.

[124] P. Ostergard. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1-3):197–207, 2002.

[125] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

[126] S. Pallier. *Analyse hors ligne d'ordonnançabilitć d'applications temps réel comportant des tâches conditionnelles et sporadiques*. PhD thesis, École Nationale Supérieure de Mécanique et d'Aérotechnique, Université de Poitiers, 2006.

[127] M. Pióro and D. Medhi. *Routing, flow, and capacity design in communication and computer networks*. Morgan Kaufmann, 2004.

[128] D. Powell. Failure mode assumption and assumption coverage. In *International Symposium on Fault-Tolerant Computing (FTCS-22)*, pages 386–395, 1992.

[129] R. Pressman. *Software engineering: a practitioner's approach*. McGraw Hill, 1992.

[130] W. Roux. *Une approche coherente pour la planification et l'ordonnacement de systèmes de production complexes*. PhD thesis, Université Paul Sabatier, Toulouse, France, 1997.

[131] J. Rushby. Critical system properties: Survey and taxonomy. *Reliability Engineering and Systems Safety*, 43(2):189–219, 1994.

[132] SAE ARP4754. Certification considerations for highly-integrated or complex aircraft systems. *Systems Integration Requirements Task Group AS-1C, Avionics Systems Division, Society of Automotive Engineers, Inc*, 1995.

[133] L. Sagaspe and P. Bieber. Constraint-based design and allocation of shared avionics resources. In *26th AIAA-IEEE Digital Avionics Systems Conference*, 2007.

[134] SATRIMMAP. `http://www.irit.fr/satrimmap/`, 2011.

[135] M. Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45(6):831–841, 1997.

[136] Y. Seok, Y. Lee, Y. Choi, and C. Kim. Explicit multicast routing algorithms for constrained traffic engineering. *IEEE 7th International Symposium on Computer and Communications*, 2002.

[137] M. Serrano and H. Boehm. Understanding memory allocation of scheme programs. In *Proceedings of the fifth ACM SIGPLAN international conference on Functional programming*, pages 245–256. ACM, 2000.

[138] A. Sharieh, W. Al_Rawagepfeh, M. Mahafzah, and A. Al Dahamsheh. An Algorithm for Finding Maximum Independent Set in a Graph. *European Journal of Scientific Research*, 23(4):586–596, 2008.

[139] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar: a flexible real time scheduling framework. In *Proceedings of the 2004 annual ACM SIGAda international conference on Ada*, pages 1–8, 2004.

[140] R. Sitters. Two np-hardness results for preemptive minsum scheduling of unrelated parallel machines. *Integer Programming and Combinatorial Optimization*, pages 396–405, 2001.

[141] C. Spitzer. *Digital avionics systems*. McGraw-Hill Inc., 1993.

[142] C. Spitzer. *The avionics handbook*. CRC Press, 2001.

[143] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, 1(1):27–60, 1989.

[144] J. Stankovic and K. Ramamritham. What is predictability for real-time systems? *Real-Time Systems*, 2(4):247–254, 1990.

[145] J. Stankovic, M. Spuri, M. Di Natale, and G. Buttazzo. Implications of classical scheduling results for real-time systems. *Computer*, 28(6):16–25, 1995.

[146] A. Tanenbaum. Computer Networks. *Prentice Hall*, 2003.

[147] S. Thanikesavan and U. Killat. Global scheduling of periodic tasks in a decentralized real-time control system. In *IEEE International Workshop on Factory Communication Systems*, pages 307–310, 2004.

[148] S. Thanikesavan and U. Killat. Static scheduling of periodic tasks in a decentralized real-time control system using an ilp. In *Proceedings of the 11th International Conference on Parallel and Distributed Systems - Workshops (ICPADS '05)*, page 639, 2005.

[149] S. Thanikesavan and U. Killat. A satisficing momip framework for reliable real-time application scheduling. In *Proceedings of the 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing (DASC '06)*, pages 187–194, 2006.

[150] K. Tindell. Deadline monotonic analysis. *Embedded Systems Programming*, 13(6):20–38, 2000.

[151] J. Ullman. NP-complete scheduling problems*. *Journal of Computer and System Sciences*, 10(3):384–393, 1975.

[152] C. Watkins and R. Walter. Transitioning from federated avionics architectures to Integrated Modular Avionics. In *Proceedings of the IEEE/AIAA 26th Digital Avionics Systems Conference (DASC'07)*, 2007.

[153] G. Woeginger. Exact algorithms for NP-hard problems: A survey. *Combinatorial Optimization-Eureka*, pages 185–207, 2003.

[154] J. Zhu, T. Lewis, W. Jackson, and R. Wilson. Scheduling in hard real-time applications. *IEEE Software*, 12(3):54–63, 1995.

[155] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE transactions on evolutionary computation*, 3(4):257–271, 1999.